Web Services in stark ressourcenlimitierten Umgebungen

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

vorgelegt von

Guido Moritz, geb. am 23.02.1983 in Kühlungsborn

Rostock, den 01.11.2012

urn:nbn:de:gbv:28-diss2013-0079-4

Gutachter:

- Prof. Dr. Dirk Timmermann (Universität Rostock, Fakultät für Informatik und Elektrotechnik, Institut MD)
- Prof. Dr. Heiko Krumm (Technische Universität Dortmund, Informatik, LS 4)
- PD Dr. Christian Werner (Technische Universität Braunschweig, Informatik, Institut für Betriebssysteme und Rechnerverbund)

Tag der Einreichung: 24.10.2012

Tag der Verteidigung: 26.04.2013

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit am Institut für Angewandte Mikroelektronik und Datentechnik der Universität Rostock. Zu großem Dank bin ich daher meinem Doktorvater Prof. Dirk Timmermann verpflichtet, der es mir durch die Anstellung am Institut ermöglichte, mein Promotionsvorhaben durchzuführen. Mein Dank gilt ebenfalls meinem Betreuer Dr. Frank Golatowski, der mich sicher durch das schwierige Fahrwasser eines Promotionsvorhabens navigierte. Beide Mentoren haben mir den notwendigen Freiraum gelassen, der zur Erbringung dieser Dissertation nötig war.

Von meinen Kollegen werde ich Elmar Zeeb stets als meinen kompetenten und hilfsbereiten Kollegen in Erinnerung behalten. Ohne die vielen interessanten Diskussionen in unserem Büro wären viele der Ideen in dieser Arbeit nie zustande gekommen. Aber auch die Gespräche mit den weiteren Kollegen und Mitgliedern der WS4D-Arbeitsgruppe am Institut haben meine Arbeit nachhaltig geprägt.

Mein Dank gilt weiterhin den ehemaligen Studenten Christian Lerche, Michael Rethfeldt, Maik Gotzmann und Benjamin Beichler, die mich im Rahmen von studentischen Arbeiten dabei unterstützt haben, die Implementierung, die in dieser Arbeit vorgestellt werden, hervorzubringen.

Widmen möchte ich diese Arbeit meiner Familie. Meine Mutter und mein viel zu früh verstorbener Vater haben mich stets in meinem Handeln und vor allem auf meinem Bildungsweg unterstützt. Durch meine liebevolle Freundin Susanne konnten weiterhin die orthographischen Unzulänglichkeiten eines Ingenieurs weitestgehend ausgeglichen werden, sodass diese Arbeit überhaupt lesbar wurde.

Inhaltsverzeichnis

Abkürzungsverzeichnis

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
AAL	Ambient Assisted Living
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
A-SOAP	Adaptive SOAP
BNF	Backus-Naur-Form
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environments
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DPWS	Devices Profile for Web Services
DTD	Document Type Definition
DTLS	Datagram Transport Layer Security
EUI	Extended Unique Identifier
Exalt	Experimental XML Archiving Library/Toolkit
EXI	Efficient XML Interchange
FFD	Full Functional Device
FI	Fast Infoset
FTP	File Transfer Protocol
GPL	GNU General Public License
H2M	Human-to-Machine
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol

IMAP	Internet Message Access Protocol
IoT	Internet of Things
IRI	Internationalized Resource Identifier
ITU	International Telecommunication Union
LTP	Lean Transport Protocols
M2M	Machine-to-Machine
MEP	Message Exchange Pattern
MSMQ	Message Queuing von Microsoft
MTU	Maximum Transmission Unit
NAT	Network Address Translation
OASIS	Organization for the Advancement of Structured Information Standards
POP	Post Office Protocol
PPM	Prediction by Partial Match
REST	Representational State Transfer
RFD	Reduced Functional Device
ROA	Ressourcenorientierte Architekturen
ROLL	Routing Over Low Power and Lossy Networks
RPL	IPv6 Routing Protocol for Low Power and Lossy Networks
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SICS	Swedish Institute of Computer Science
SMTP	Simple Mail Transfer Protocol
SOA	Service-orientierte Architekturen
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TI	Texas Instruments
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UNICODE	8 Bit UCS Transformation Format
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier

URL	Uniform Resource Locator
URN	Uniform Resource Name
UUID	Universally Unique Identifier
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WBXML	WAP Binary XML
WG	Working Group
WLAN	Wireless Local Area Network
WML	Wireless Markup Language
WS4D	Web Services for Devices
WS	Web Services
WSA	Web Services Architecture
WSDL	Web Services Description Language
WSE	Web Services Enhancements
WSRA	Web Services Resource Access
Xaust	XML compression with Automata and a Stack
XML	Extensible Markup Language
XML-Infoset	XML Information Set
XMLPPM	XML-Conscious PPM
XMPP	Extensible Messaging and Presence Protocol
xs:	XML-Schema

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Inhaltsübersicht

In den vergangenen 50 Jahren hat sich das Internet von einer einfachen Idee zu einem der wichtigsten und zugleich komplexesten technischen Systeme entwickelt. Als Beginn der Entwicklung des Internets kann das Jahr 1961 bezeichnet werden, in dem Leonard Kleinrock in seiner Dissertation [1] die mathematischen Grundlagen für paketvermittelte Kommunikation vorgestellt hat. Als Geburtsstunde des Internets wird dennoch meist das Jahr 1969 genannt, in dem das ARPANET [2] die ersten vier Internetknoten miteinander verband. Das ARPANET entwickelte sich schnell weiter. Nur fünf Jahre später wurde von Vincent Cerf und Bob Kahn mit dem Internet Transmission Control Program [3, 4] die Grundlage für ein globales Netzwerk geschaffen, dessen technische Basis fortan unter dem Akronym TCP/IP zusammengefasst wurde. Bis heute hat sich an der Paketvermittlung und an dem grundlegenden Design hinter TPC/IP nichts geändert. Beides kommt nach wie vor zum Einsatz.

Mit dem Einzug des Internets in die Haushalte am Ende des zwanzigsten Jahrhunderts hat sich die bis dahin bereits rasante Entwicklung des Internets weiterhin vervielfacht. War das Internet zuvor für militärische und wissenschaftliche Zwecke entworfen worden, entwickelte es sich nun zu einem weltweiten Kommunikationsnetz für Millionen Nutzer. Einen essenziellen Anteil an dieser Entwicklung hatten Betriebssysteme für PCs, die durch grafische Benutzeroberflächen die Bedienung von Computern für jedermann ermöglichten.

Bis Mitte der 90er Jahre des letzten Jahrhunderts wurde das Internet vornehmlich genutzt, um Nutzern den Zugang zu Informationen zu ermöglichen. Das Abrufen der Informationen erfolgte

meist über Webseiten. Schnell wurde klar, dass diese triviale Form des Informationsaustausches nicht allen Bedürfnissen der Nutzer genügte, die ebenfalls untereinander kommunizieren wollten. Daher wurde es nötig, dass die Nutzer selbst Informationen in das Netz hineingeben. Das Internet, welches sich durch die Einbeziehung der Nutzer zur Generierung von Inhalten selbst neu erfand, wird als *Web 2.0* bezeichnet. Für das *Web 2.0* entstand unter dem Begriff *Prosument* ein neuer Anwender des Internets. Der Begriff *Prosument* entsteht aus der Zusammensetzung der Wörter *Produzent* und *Konsument*. Tim Berners-Lee bezeichnet diese Verbindung von Menschen durch gezielten Informationsaustausch als *Read/Write Web* [5].

Schon seit langem hat die Menge der im Internet zur Verfügung stehenden Daten und Informationen bei weitem die Menge dessen überschritten, was vom menschlichen Gehirn erfasst werden kann. Daher stellt eine notwendige Erweiterung des *Web 2.0* die Art und Weise dar, wie ein Nutzer mit Informationen bedient wird. Das Internet muss als intelligentes Netzwerk die Daten für jeden Nutzer individuell aufbereiten. Für diese grundlegende Erweiterung findet sich in der Literatur inzwischen der Begriff des *Web 3.0*. Die hierfür notwendige technische Weiterentwicklung wird allgemein als *Semantik* zusammengefasst. *Semantiken* ermöglichen es, Informationen und Daten um eine Bedeutung und einen Kontext zu ergänzen. Hierdurch können die Daten mittels Software automatisch verarbeitet werden. Die Übergänge zwischen den Begrifflichkeiten *semantisches Web* und *Web 3.0* können nach Lee et al. und Markoff daher als fließend betrachtet werden (siehe [6, 7]).

In das *Web 3.0* fließen weiterhin Entwicklungen ein, die zusammengefasst als das *Internet der Dinge* (engl. Internet of Things; IoT) bezeichnet werden (siehe Abbildung 1.1). Das *Internet der Dinge* umfasst Bemühungen, bei denen physikalische Objekte ebenfalls Teil des Kommunikationsnetzes werden und ihre Daten weitergeben. Ob letztendlich das *semantisches Web* oder das *Internet der Dinge* in der Zukunft als die wichtigste Neuerung des *Web 3.0* betrachtet werden kann, muss sich erst noch herausstellen. Allerdings schließen sich die beiden Ausprägungen gegenseitig nicht aus, sondern können sich vielmehr ergänzen.

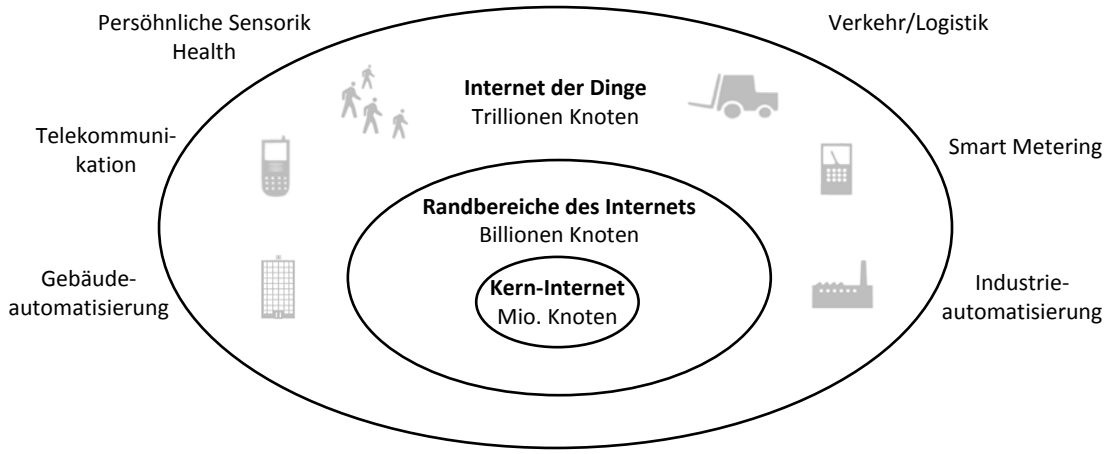

Abbildung 1.1 Internet der Dinge [8]

Bereits in naher Zukunft werden basierend auf vernetzten Komponenten intelligente Umgebungen entstehen. Diese bestehen aus diversen selbständig handelnden Systemen, die den Mensch als Individuum mit Informationen versorgen und wenn möglich im täglichen Leben unterstützen. Diese Visionen sind nicht neu, sondern wurden bereits 1991 von Marc Weiser als *ubiquitäres Computing* beschrieben [9, 10].

1.1 Problemstellung und Zielsetzung der Arbeit

Bei Visionären wie Marc Weiser bleibt oft die Frage unbeantwortet, mit welchen konkreten technischen Lösungen physikalische Objekte in eine intelligente Umgebung eingebunden werden können. Im Kern stellt sich die Frage, welche Protokolle und Technologien konkret genutzt werden können, um ein *Internet der Dinge* zu realisieren.

Geprägt durch frühere Entwicklungen gab es hinsichtlich der zur Verfügung stehenden Protokolle für ein *Internet der Dinge* bis vor kurzem die getrennten Bereiche des *Internets* und der *Dinge* bzw. *Geräte*. Für den Bereich des *Internets* existieren zahlreiche offene Standards, die meist anwendungs- und domänenunabhängig sind und ausschließlich die Vernetzung von Endpunkten zum Ziel haben. Aufgrund der Unabhängigkeit von einer Domäne sind diese Standards horizontal konzipiert. Ob z.B. Texte, Audio- oder Videodaten durch die Protokolle übertragen werden, ist in der Regel nicht relevant. In *gerätenahen* Anwendungen war bislang eine globale Vernetzung nicht vorgesehen, wodurch Technologien und Protokolle meist für einen gezielten vertikalen Anwendungsfall entwickelt wurden und folglich nicht in der Lage sind, auch andere Daten zu

übermitteln. Geschäftsmodelle von Geräteherstellern haben die Entwicklung von eher proprietären Lösungen weiterhin begünstigt, um die Kunden an den jeweiligen Hersteller zu binden.

Um *gerätenahe Umgebungen* und *Internetdienste* zu verschränken, befasst sich das Forschungsfeld des *Internets der Dinge* unter anderem mit der Fragestellung, wie *Internettechnologien* und *-protokolle* in *gerätenahen Anwendungen* genutzt werden können. Es gilt zu untersuchen, wie sich die existierenden Konzepte, Architekturen und Technologien aus der virtuellen Datenwelt des *Web 2.0* in die reale Welt der Geräte und physikalischen Objekte überführen lassen. Da die existierenden *Internettechnologien* nicht für die Vernetzung von *Geräten* entwickelt wurden, sind dedizierte Untersuchungen nötig, welche Protokolle sich für spezifische Lösungen am besten eignen und ob ggfs. Erweiterungen notwendig sind.

Höherwertige Dienste, die entweder im Internet oder in Unternehmen im Intranet eingesetzt werden, sind meist auf ressourcenstarken Servern implementiert, sodass Hardware- und Bandbreitenbeschränkungen keine nennenswerte Herausforderung darstellen. *Gerätenahe Netzwerkanwendungen*, wie beispielsweise netzwerkfähige Drucker, Scanner und Beamer, bei denen Internettechnologien genutzt werden, benötigen zwar gegenüber dem klassischen Internet geänderte Konzepte hinsichtlich der genutzten Vernetzungslösungen [11], sind aber dennoch nicht zwingend durch starke Ressourcenlimitierungen geprägt. Solche Limitierungen treten erst beim Einsatz von Kleinstgeräten auf, die meist als einfache Sensoren und Aktoren genutzt werden. Gründe für die Ressourcenknappheit ergeben sich aus der miniaturisierten Bauform, geringen Herstellungskosten und dem kabellosen Batteriebetrieb, welcher eine hohe Energieeffizienz erfordert. Je nach Anwendung und Szenario kann die Rechenkapazität, der begrenzte Speicher, die Bandbreite oder die zur Verfügung stehende Energie als wesentlichste Einschränkung angesehen werden.

Diese Arbeit konzentriert sich auf die genannten Kleinstgeräte, die starken Ressourcenbeschränkungen unterliegen. Diese Geräte sind bezüglich ihrer Anforderungen den *drahtlosen Sensornetzwerken* [12, 13] sehr ähnlich, konnten unter dieser Bezeichnung aber bislang keine nennenswerte Marktdurchdringung erreichen [14]. Bei den dieser Arbeit zugrundeliegenden Szenarien und genutzten Zielplattformen entsteht die Komplexität nicht durch den Umfang und die Leistungsfähigkeit der einzelnen Dienste selbst, sondern durch die interoperable Kombination und Komposition verschiedenster Dienste, die unabhängig von den zur Verfügung stehenden Ressourcen nahtlos miteinander kommunizieren und interagieren können. Ziel dieser Arbeit ist es,

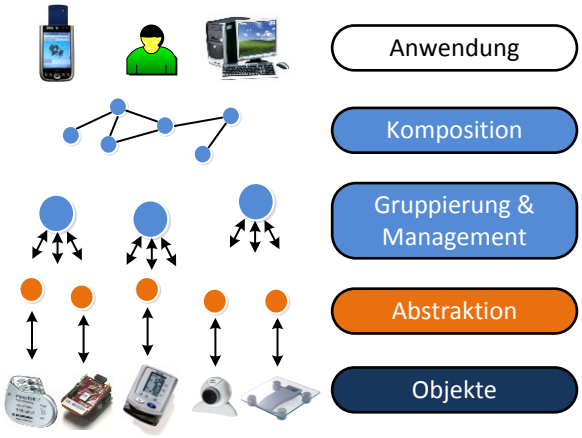

Abbildung 1.2 Dienstbasierte Umgebung [15]

die bereits in *höherwertigen Internetdiensten* und für *ressourcenstarke Geräte* eingesetzten Konzepte und Technologien hinsichtlich ihrer Eignung für *ressourcenarme Kleinstgeräte* zu untersuchen und eine ganzheitliche Vernetzungslösung hervorzubringen, bei der von den Objekten abstrahiert werden kann und diese in eine dienstbasierte Umgebung integriert werden können (siehe Abbildung 1.2).

Nach dem Moore‘schen Gesetz [16, 17], welches bis heute seine Gültigkeit behalten hat, verdoppelt sich alle 18 bis 24 Monate die Komplexität bzw. Integrationsdichte von integrierten Schaltkreisen. Eine Optimierung von Protokollen und Implementierungen wäre demnach innerhalb kürzester Zeit gegenstandslos, da auf den Zielplattformen wesentliche Fortschritte hinsichtlich der zur Verfügung stehenden Ressourcen zu erwarten wären. Das Moore‘sche Gesetzt darf im Bereich der stark ressourcenlimitierten Umgebungen aber nicht falsch interpretiert werden. In diesen Umgebungen hat die fortschreitende technische Entwicklung die Minimierung der Stromaufnahme der Komponenten zur Folge und führt weiterhin zu geringeren Produktionskosten. Eine Steigerung der Rechenleistung, der Speicherkapazität oder der Bandbreite ist in drahtlosen Sensornetzwerken bislang nicht zu beobachten [18]. Vielmehr bilden die ressourcenlimitierten Plattformen durch die neuartige Einbeziehung in das weltweite Datennetz mittels standardisierter Internettechnologien eine neue Kategorie von Computern. Diese neue Kategorie von Computern wurde bereits von Gordon Bell im Bell‘schen Gesetz formuliert [19]. Weiterführend hat Robert Metcalf im Metcalfe‘schen Gesetz [20, 21] eine Regel für die Erfassung des Wertes eines Netzwerkes in Abhängigkeit von der Anzahl der Teilnehmer formuliert. Nach Metcalf steigt der Nutzen eines Kommunikationssystems mit einer genügend hohen Anzahl von Teilnehmern proportional zum

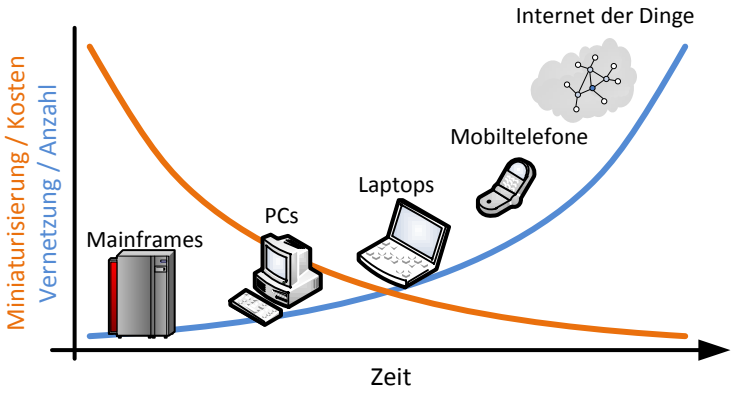

Abbildung 1.3 Entwicklung Internet der Dinge

Quadrat, da jeder neue Endpunkt in einem Netzwerk mit allen anderen Endpunkten kommunizieren kann.

Fasst man die drei Gesetzmäßigkeiten zusammen (siehe Abbildung 1.3), entsteht eine neue Klasse von Computern (Bell), die sich durch Miniaturisierung, geringen Energiebedarf und Kosteneffizienz auszeichnet (Moore) und durch die immens hohe Anzahl von interoperablen Endpunkten einen enormen Gesamtnutzen erzielt (Metcalf).

In diesem Kontext bringt diese Arbeit ein geeignetes *Anwendungsschichtprotokoll* für stark ressourcenlimitierten Umgebungen hervor. Als Grundlage dient das *Devices Profile for Web Services (DPWS)*. Das Protokoll DPWS ist derzeit in der Version 1.1 [22] bei der OASIS (Organization for the Advancement of Structured Information Standards) [23] standardisiert und ist unter anderem integraler Bestandteil von Windows Vista und Windows 7 [24]. DPWS ist konzeptionell sehr ähnlich zur Web Service Architecture (WSA) [25], wurde aber den spezifischen Anforderungen für den Einsatz in gerätenahen Umgebungen angepasst. Die Nutzung der Dienste bei DPWS ist identisch zu denen der WSA, wodurch DPWS-Geräte direkt in WSA-basierte Umgebungen eingebunden werden können [26–28]. DPWS basiert auf dem SOAP-Protokoll [29] des World Wide Web Consortium (W3C) und der damit einhergehenden Familie der WS-* Standards. Obwohl weder SOAP noch DPWS für den Einsatz in stark ressourcenlimitierten Umgebungen entworfen wurden, zeigt diese Arbeit, dass Anpassungen und Erweiterungen vorgenommen werden können, um den spezifischen Anforderungen gerecht zu werden.

Die notwendige Basis für den Einsatz von Internettechnologien stellt das Internet Protokoll (IP) auf der Vermittlungsschicht des OSI-Referenzmodells [30] dar. IP kann in den beiden verbreitetsten Versionen IPv4 [31] und IPv6 [32] ebenfalls nicht nativ in stark ressourcenlimitierten Umgebungen eingesetzt werden. Dieses Problems hat sich im Jahr 2006 die Internet Engineering Task Force (IETF) angenommen und die 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) Arbeitsgruppe ins Leben gerufen. Ergebnis aus der 6LoWPAN Arbeitsgruppe sind verschiedene Spezifikationen [33, 34], die den effizienten Einsatz von IPv6 in drahtlosen Sensornetzwerken ermöglichen. Die Netzwerkumgebungen, auf die sich diese Arbeit bezieht, werden daher im Folgenden zusammengefasst als 6LoWPANs bezeichnet. Die 6LoWPAN-Protokolle bilden die Basis für weitere Entwicklungen innerhalb der IETF, wie beispielsweise spezifische IPv6-Routinglösungen für vermaschte drahtlose Sensornetzwerke (Arbeitsgruppe ROLL; Routing Over Low Power and Lossy Networks [35, 36]) oder adaptive Protokolle auf der Anwendungsschicht (CoAP; Constrained Application Protocol [37]).

1.2 Architekturprinzipien und Basisarchitektur

In heutigen IP-basierten Internetanwendungen wird zwischen zwei Basisarchitekturen unterschieden. Ressourcenorientierte Architekturen (ROA) werden unter anderem durch Richardson und Ruby in [38] beschrieben. In ROAs werden verteilte Entitäten, wie zum Beispiel Daten, als abstrakte Ressourcen modelliert. Die Ressourcen können durch definierte Schnittstellen abgerufen und manipuliert werden. Dazu ist jede Ressource getrennt adressierbar. Eine konkrete Softwaretechnik zur Umsetzung einer ROA wurde von Fielding in seiner Dissertation [39] als Representational State Transfer (REST) beschrieben. Fielding stellt dar, auf welchen Prinzipien die Kommunikation im World Wide Web basiert, fasst diese als REST zusammen und erläutert deren Realisierung mittels des Hypertext Transfer Protocol (HTTP).

Im Gegensatz zu ROAs wird in Service-orientierten Architekturen (SOA) von komplexen Vorgängen und Funktionen durch Dienste abstrahiert. Die Dienste sind dabei nicht wie bei ROAs an spezielle Semantiken, Eingaben oder Ausgaben der Aufrufe gebunden. Der Begriff SOA wird oft im Zusammenhang mit Geschäftsprozessen genutzt, da durch die Kapselung der Funktionen mittels Dienste die Granularität und das Maß der Abstraktion nicht definiert sind. Dienste können einfache atomare Funktionen auf sehr niedrigen Ebenen, wie auch komplexe und verschachtelte

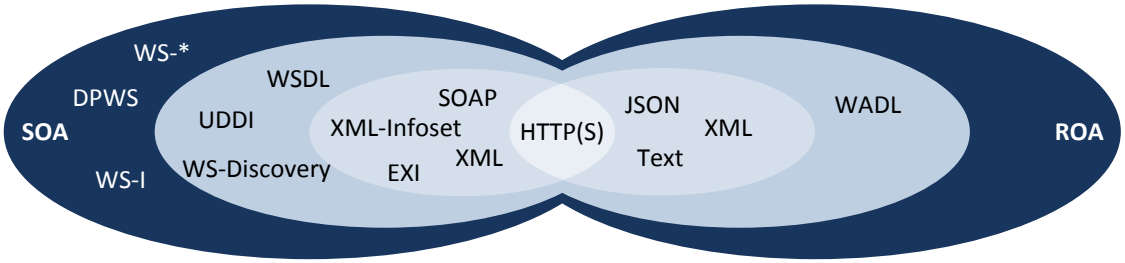

Abbildung 1.4 Abstrakte Gegenüberstellung möglicher Protokolle SOA und ROA

Geschäftsprozesse abbilden. Umfassende Einführungen zu SOAs wurden von Erl in [40] und von Melzer in [41] gegeben.

Die beiden genannten Basisarchitekturen unterliegen grundlegenden Prinzipien. Umsetzungen der Stile können auf diesen Prinzipien beruhen. SOAs und ROAs bilden aber keine Lösung für fachliche Probleme, noch existieren SOAs bzw. ROAs, die allgemeingültig angewendet werden können. Die Umsetzung muss immer mittels eines konkreten Protokolls erfolgen und an die konkrete Anwendung angepasst werden. So beschreibt REST keine Technologie oder ein Protokoll, sondern ist eine Softwaretechnik und stellt die Umsetzung einer ROA für den Einsatz im World Wide Web dar. Für die Definition von REST beschreibt Fielding die Prinzipien, auf denen HTTP beruht. REST wurde somit nach der Entstehung von HTTP davon abgeleitet.

Während REST und HTTP fälschlicherweise oft synonym verwendet werden, werden SOAs oft irrtümlich mit SOAP-basierten Web Services gleichgesetzt. Dabei ist mit SOAP-basierten Web Services lediglich die konkrete Umsetzung von SOAs möglich, die sich ebenfalls auf HTTP stützt. Ebenso bildet HTTP eine konkrete Umsetzung einer ROA (siehe Abbildung 1.4).

HTTP wird meist als schlankes Protokoll beschrieben, welches durch eine REST-basierte Realisierung den ROA-Prinzipien entspricht. Ebenso können ROAs auch mittels anderer Protokolle als HTTP umgesetzt werden und HTTP ist nicht auf die Realisierung einer ROA beschränkt. Mittels SOAP-basierter Web Services können SOAs realisiert werden. Die Web Services Resource Access (WSRA) Arbeitsgruppe [42] des W3C definiert SOAP-basierte Mechanismen zur Realisierung von ressourcenorientierten Web Services. Wie somit der Einsatz von SOAP nicht auf die Umsetzung einer SOA beschränkt ist, sind SOAs sind nicht auf SOAP beschränkt. Pautasso beschreibt in [43], dass der Vergleich von SOAs und REST dem Vergleich von Äpfeln mit Birnen ähnelt, da Basisarchitekturen, Softwaretechniken, Protokolle und Implementierungen oft vermengt und nicht losgelöst betrachtet werden.

Im Mittelpunkt einer Gegenüberstellung von SOAs und ROAs steht daher der Vergleich der Architekturprinzipien, nicht aber der Vergleich der Protokolle oder deren Implementierungen. Letztere haben einen erheblichen Einfluss auf die Leistungsfähigkeit der Gesamtlösung. Beim Vergleich der umsetzenden Protokolle muss beachtet werden, dass beide Stile nicht auf ein einzelnes Protokoll, beschränkt sind, wie auch ein einzelnes Protokoll verschiedene Stile abbilden kann.

Zusammengefasst haben die beiden Architekturstile, SOAs und ROAs, verschiedene Vor- und Nachteile. Weiterhin haben Protokolle, mit denen die Prinzipien der Stile umgesetzt werden können, ebenfalls Vor- und Nachteile. Es kann nicht pauschal beantwortet werden, welche der Lösungen bei einer konkreten Anwendung zu bevorzugen ist.

Diese Arbeit abstrahiert von weiterführenden Betrachtungen bezüglich der Auswahl einer geeigneten Basisarchitektur, sondern konzentriert sich auf die Hervorbringung einer ganzheitlichen Vernetzungsarchitektur. Dies schließt die Konzeption geeigneter Protokolle mit ein.

1.3 Inhaltsüberblick

Um die aufgezeigte Problemstellung zu bearbeiten, ist diese Arbeit in acht Kapitel unterteilt. In Abbildung 1.5 wird der Aufbau dieser Arbeit zu Veranschaulichung grafisch dargestellt. Die Abschnitte dieser Arbeit mit wesentlichen eigenen Beiträgen sind orange hervorgehoben.

Nach der Einleitung ordnet das Kapitel 2 die Inhalte dieser Arbeit spezifischen Anwendungsgebieten zu und motiviert so die technischen Ansätze. Hierzu werden beispielhaft Szenarien aus dem Bereich Ambient Assisted Living und der Gebäudeautomatisierung dargestellt.

Das Kapitel 3 zeigt die wesentlichsten Grundlagen und verwandte Arbeiten im dargestellten Themenumfeld auf. Die Beiträge aus diesem Kapitel konzentrieren sich auf den Einsatz von Internettechnologien zur Vernetzung in gerätenahen Umgebungen.

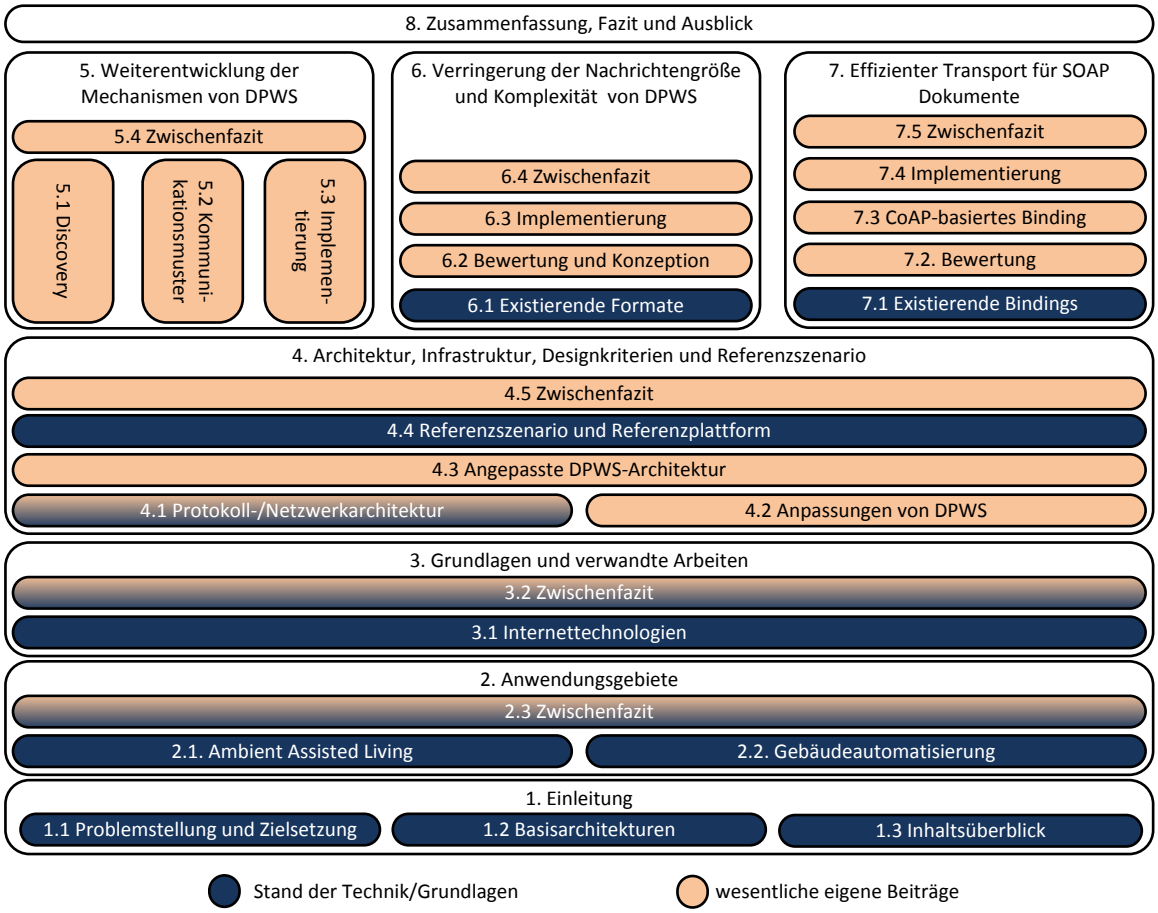

Abbildung 1.5 Inhaltsüberblick

Viele verwandte Arbeiten und Ansätze stellen lediglich einfache Optimierungen der Protokolle für spezifische Umgebungen oder dedizierte Anwendungen dar. Allerdings fehlt diesen Ansätzen oftmals eine grundlegende Architektur, um die entwickelten Lösungen allgemeingültig in heterogenen Umgebungen einzusetzen. Das Kapitel 4 dieser Arbeit konzipiert unter der Berücksichtigung der in diesem Kapitel ebenfalls diskutierten Prinzipien eine solche ganzheitliche Netzwerkinfrastruktur und Protokollarchitektur. Kapitel 4 gibt, basierend auf der zuvor entwickelten Architektur, einen genaueren Überblick über die einzelnen Teilschritte, die notwendig sind, um SOAP-basierte Web Services in stark ressourcenlimitierten Netzwerken wie 6LoWPANs einsetzen zu können.

Die Kapitel 5 bis 7 stellen die Hauptbeiträge dieser Arbeit dar. Dazu sind in Kapitel 5 vornehmlich Neuentwicklungen und Erweiterungen genereller Schnittstellenmechanismen und -funktionen für DPWS aufgezeigt, um dieses spezifische Geräteprofil in bandbreitenbeschränkten Netzwerken

anzuwenden. Die Kapitel 6 und 7 stellen Lösungen zur Verringerung der Nachrichtengröße sowie zum effizienten Transport der Nachrichten dar. Die neuartige Lösung zur Verringerung der Nachrichtengröße basiert auf dem Efficient XML Interchange (EXI) [44] Format des W3C. Mittels des in Kapitel 6 dargestellten Ansatzes lassen sich die Nachrichten auf unter 10 % (geringer als 100 Byte) der ursprünglichen Größe komprimieren. Der in Kapitel 7 definierte Transportmechanismus der Nachrichten basiert auf dem neuen Constrained Application Protocol (CoAP) der IETF und wird in dieser Arbeit erstmals ausführlich vorgestellt. Alle Beiträge der Kapitel 5 bis 7 sind durch vollständig funktionsfähige Implementierungen validiert. Details aus den Implementierungen und experimentelle Ergebnisse zur Bewertung werden ebenfalls in den einzelnen Kapiteln diskutiert.

Obwohl jedes einzelne Kapitel mit einem Zwischenfazit endet, werden in Kapitel 8 alle relevanten Ergebnisse dieser Arbeit nochmals zusammengefasst und ein Ausblick auf mögliche zukünftige Arbeiten gegeben.

2 Anwendungsgebiete

Inhaltsübersicht

Bedingt durch die Entwicklung des Internets als System, bei dem Nutzer Informationen abrufen können, hat sich für diese Nutzungsform die Bezeichnung Human-to-Machine (H2M) durchgesetzt. Der Mensch greift dabei auf definierte Benutzerschnittstellen zum Abrufen der Daten, z.B. aus Datenbanken oder anderen Informationsquellen, zurück. Die wichtigsten Aspekte bei der H2M-Kommunikation sind die Navigation durch die Dienste und Daten sowie die Darstellung der Informationen. Der Nutzer kann selbstständig entscheiden, welche Daten abgerufen und welche Dienste in einer Sequenz genutzt werden sollen. Weiterhin kann ein menschlicher Nutzer Informationen aus dem Kontext heraus meist selbstständig um fehlende Semantiken oder Strukturierungen der Daten ergänzen. Mit der Entwicklung des Web 2.0 wurden die Daten zunehmend von den Nutzern selbst generiert. Die Nutzungsform des Internets entspricht aber dennoch dem H2M-Muster.

Im gerätenahen Umfeld und auch in Geschäftsprozessen findet sich hingegen oft der Begriff der Machine-to-Machine (M2M) Kommunikation. Bei diesem Muster kommunizieren Endgeräte, wie beispielsweise Maschinen, Fahrzeuge, Sensoren und Aktoren direkt miteinander. Die Interaktion der Endpunkte untereinander erfordert dabei kein Eingreifen von menschlichen Nutzern, sondern läuft vollständig autonom ab. Dies beinhaltet, neben der effizienten und nahtlosen Interaktion verschiedenster heterogener Entitäten, auch das autonome Suchen und Finden von anderen geeigneten Diensten innerhalb der intelligenten Umgebung. Dadurch wird ein *‚Plug and Play‘* Verhalten innerhalb des Netzwerkes erreicht. Gerätenahe Dienste verfügen zusätzlich zur Fähigkeit der Kommunikation über die Möglichkeit, die physikalische Welt durch Sensorik zu erfassen oder

Aktorik zu beeinflussen [45]. Dies unterscheidet gerätenahe Dienste von höherwertigen Diensten, z.B. im Internet.

Durch die Bestrebung zu einem *Internet der Dinge* bzw. dem *Web 3.0* kommen für die M2M-Kommunikation zunehmend existierende Internettechnologien und -protokolle zum Einsatz. Dadurch werden die gerätenahen M2M-Anwendungen nahtlos interoperabel mit dem weltweiten Kommunikationsnetz und können die Daten mit anderen Maschinen bzw. Geräten, aber auch mit menschlichen Nutzern gleichermaßen teilen.

Die in dieser Arbeit entstandenen Lösungen sind domänenunabhängig in verschiedenen Umgebungen einsetzbar. Um mögliche Szenarien aufzuzeigen, werden in den nachfolgenden Unterkapiteln beispielhaft zwei Anwendungsgebiete beschrieben. Die Einbeziehung von höherwertigen Diensten aus dem Internet und der damit einhergehende Mehrwert durch die Vernetzung sind ebenfalls Bestandteil der Beschreibungen.

2.1 Ambient Assisted Living

Bereits in wenigen Jahren werden immer weniger Personen im arbeitsfähigen Alter für immer mehr ältere Personen sorgen müssen (siehe Abbildung 2.1). Motiviert vom diesem demografischen Wandel [46] existieren derzeit diverse nationale und internationale Bemühungen, die das Ziel haben, die persönliche Lebensqualität wie auch die volkwirtschaftliche Tragfähigkeit innerhalb einer alternden Gesellschaft aufrechterhalten zu können [47–51]. Der gesamte, sich mit dieser Thematik befassende Bereich, wird als Ambient Assisted Living (AAL) bezeichnet. Das Gebiet AAL erfährt, neben technischen Entwicklungen, auch weiterführende zum Teil interdisziplinäre Ausprägungen, die beispielsweise soziale oder ökonomische Betrachtungen beinhalten.

Der Bereich AAL setzt sich im Kern aus zwei Szenarien zusammen, die sich in Teilen überlappen. Zum einen sollen technische Assistenzsysteme genutzt werden, um Patienten bzw. Bewohner im alltäglichen Leben zu unterstützen. Diese Ausprägung beinhaltet Funktionen, wie beispielsweise das automatische Öffnen und Schließen von Türen. Diese Funktionen können im Allgemeinen als Komfortfunktionen bezeichnet werden. Mit steigendem Alter und unter dem Einfluss schwerer Erkrankungen werden diese Komfortfunktionen eine zunehmend essenzielle Voraussetzung für die Bewältigung des alltäglichen Lebens.

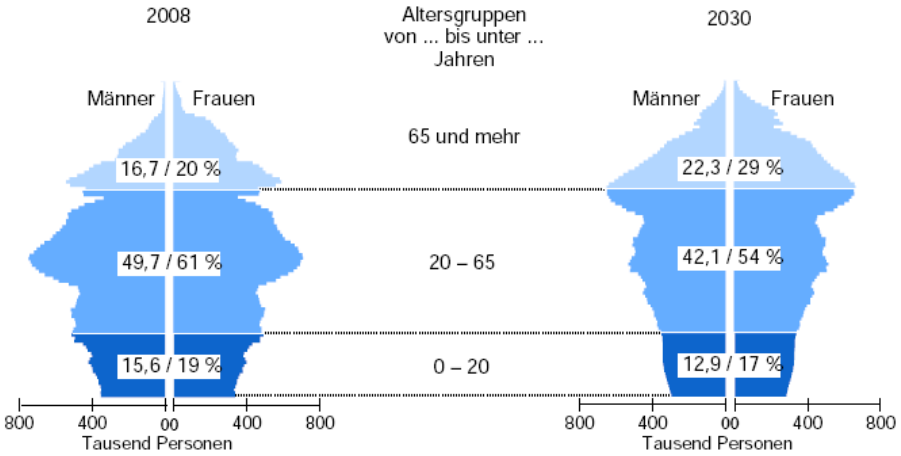

Abbildung 2.1 Bevölkerung in Deutschland in Tausend und Prozent [46]

Zum anderen werden technische Systeme genutzt, um Patienten und deren Krankheitsverlauf zu überwachen (engl. Monitoring). Um spezifische Vitaldaten zur Überwachung eines Krankheitsverlaufes beobachten zu können, müssen Patienten bisher oftmals Spezialisten aufsuchen. Die Spezialisten erfassen die Vitalparameter und werten diese aus. Ist der Patient durch entsprechende Messgeräte und -methoden in der Lage, die Messungen selbstständig zu Hause durchzuführen und die Resultate in Datenbanken sowie –auswertungssysteme im Internet zu übertragen, so kann dies zum einen zu einer Steigerung der Lebensqualität der Patienten führen. Zum anderen werden behandelnde Spezialisten entlastet. Weiterhin wird eine feingranulare Datenerfassung möglich. Die Überwachung von Patienten erfordert dabei eine nahtlose Interaktion der Geräte mit höherwertigen Internetdiensten. Beispiele für diese höherwertigen Dienste sind elektronische Krankenakten und weiterführende Auswertungsdienste [51] (siehe Abbildung 2.2).

Durch den Verbleib der Patienten in der häuslichen Wohnumgebung müssen Notfallsituationen autonom vom AAL-System erkannt und entsprechende Maßnahmen eingeleitet werden. Auch hierbei können höherwertige Dienste aus dem Internet eingesetzt werden, um im Notfall die relevanten Akteure der Alarmierungskette zu erreichen. Tritt die Notfallsituation ein, so können die vom Patienten bereits genutzten Datenentitäten und die darin enthaltenen krankheitsrelevanten Informationen über Internetdienste allen Spezialisten zugänglich gemacht werden, die an der Bewältigung der Notsituation beteiligt sind.

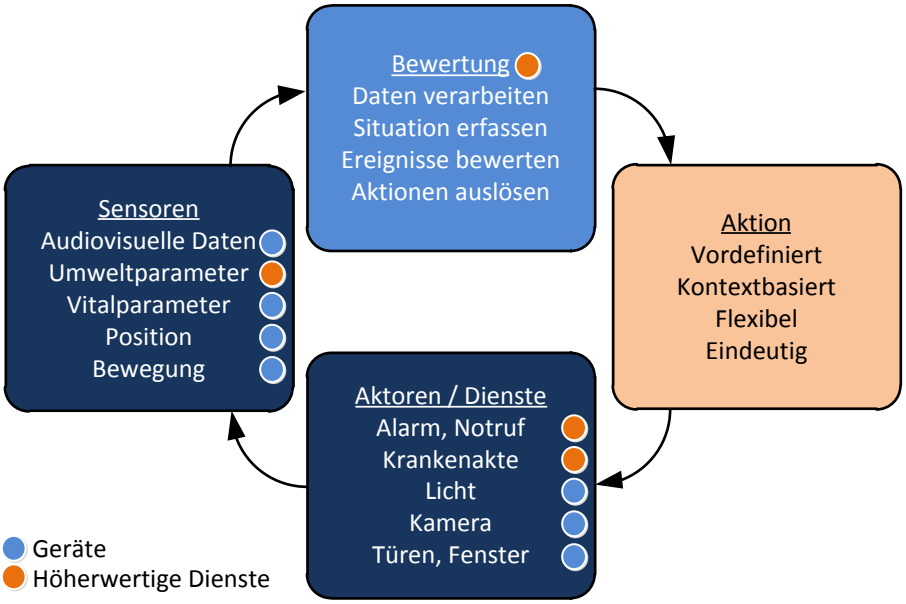

Abbildung 2.2 Integrierte AAL-Architektur

Die technischen Herausforderungen bei der Entwicklung eines Assistenzsystems liegen in der Dynamik bei der Vernetzung von Geräten und der Integration neuer Komponenten. Zudem beinhaltet jedes Einsatzszenario unterschiedlichste Anforderungen (z.B. Patiententyp und Krankheitsbild) und Geräte (z.B. Messung Vitalparameter, Position, audiovisuelle Daten). Das hohe Maß an Diversität und Heterogenität erfordert eine wohl definierte Gesamtarchitektur, bei der sich alle Geräte über einen einfachen Mechanismus in einen großen Geräte- und Dienstkomposition einbinden lassen. Dadurch lassen sich Szenarien realisieren, die aufgrund der Individualität der Problemstellungen des konkreten Patienten von Geräte- und Systemherstellern nicht berücksichtigt werden können.

Die entstehenden Geräteensembles im Bereich AAL sind hinsichtlich der Größe der entstehenden Netzwerke vergleichsweise klein und umfassen laut Schätzungen lediglich bis zu 100 Endpunkte. Da die Netzwerke meist auf das häusliche Wohnumfeld beschränkt sind, stellt die Sendereichweite bei drahtloser Übertragung ebenfalls keine nennenswerte Herausforderung dar. Alle Geräte und relevanten Endpunkte können in der Regel direkt erreicht werden. Vielmehr muss die eingesetzte Kommunikationsinfrastruktur dem hohen Maß an Heterogenität der Geräte und der Diversität der Anwendungen und Szenarien genügen. Weiterhin sind ein besonders Maß an Sicherheit und Zuverlässigkeit der eingesetzten Übertragungslösung von wesentlicher Bedeutung. Ferner stellen vor allem die am Körper getragenen Sensoren zur Überwachung der Vitalparameter eine nicht zu

vernachlässigende Herausforderung dar. Wegen des permanenten Mitführens am Körper des Patienten müssen diese miniaturisierte Bauformen aufweisen und durch die Energieversorgung mittels Batterie muss eine effiziente Form der Datenübertragung gewährleisten sein.

2.2 Gebäudeautomatisierung

Während die Domäne AAL vornehmlich auf den menschlichen Nutzer fokussiert ist, umfasst der Bereich der Gebäudeautomatisierung technische Prozesse in Gebäuden. Dies beinhaltet Aspekte wie die Überwachung, Steuerung und Regelung von Systemen. Im Kern umfasst die klassische Gebäudeautomatisierung Systeme, die unter der Abkürzung HLK (Heizung, Lüftung, Klimatechnik; engl. HVAC, Heating, Ventilation and Air Conditioning) zusammengefasst werden. Der Grund für die Fokussierung auf diese Anwendungen besteht in der Möglichkeit, durch die optimierten Steuerungs- und Regelungsprozesse dieser Systeme den Energieverbrauch von Gebäuden zu senken [52, 53] (vgl. Smart Grid).

In größeren Zusammenhängen betrachtet sind Gebäude Konsumenten in einem komplexen Netzwerk aus Energieerzeugern und Energieverbrauchern. Da Energie zum Zeitpunkt des Verbrauches in Echtzeit erzeugt wird und nicht in unbegrenztem Maß vorgehalten werden kann, werden derzeit Lösungen entwickelt, mit denen Erzeuger eine bessere Planbarkeit bei der Bereitstellung der benötigten Energiemengen erreichen. Die Grundlage hierfür bildet die informationstechnische Vernetzung der Geräte bzw. Systeme und somit der Energieverbraucher im Gebäude mit den Energielieferanten. Zum einen können so statistische Erhebungen vorgenommen werden, um die rechtzeitige Bereitstellung der Energie zu gewährleisten. Zum Zweiten können Energielieferanten bei einem sehr hohen Energiebedarf einzelner Abnehmer (z.B. Schwimmbäder) durch autonome Aushandlungsprozesse zwischen den energieverbrauchenden und den energieerzeugenden Entitäten Spitzen bezüglich der Abnahmemenge abfangen. Dies trägt im gesamten Netz zur Lastenreduzierung bei. Weiterhin sollen niedrige Energiepreise, die es in Zeiten von Energieangeboten gibt, den Verbraucher dazu anregen, Energie zu konsumieren [54]. Dies trägt zur Stabilisierung des gesamten Energienetzes und zur Reduzierung der vorsorglich vorgehaltenen Versorgungsreserven bei.

Abbildung 2.3 Vernetze Gebäude und höherwertige Dienste

Zukünftige Entwicklungen in der Domäne der Gebäudeautomatisierungen umfassen somit, neben der Vernetzung des heterogenen Geräteensembles, ebenfalls die Einbindung in ein intelligentes Netzwerk aus Energieverbrauchern, Energieversorgern und anderen Dienstleistern (siehe Abbildung 2.3). Die Energieversorger und Dienstleister bieten dazu höherwertige Dienste im Internet an, die die Überwachung und Steuerung der Geräte und Systeme im Gebäude ermöglichen.

Gerade in größeren Gebäuden bzw. Gebäudekomplexen ergibt sich schnell eine hohe Anzahl von Endpunkten, die Daten bzw. Dienste bereitstellen. Die hohe Anzahl und die Anforderung der Wirtschaftlichkeit erfordern niedrige Herstellungs-, Wartungs- und Installationskosten. Durch den großen Preisdruck müssen die eingesetzten Kommunikationsprotokolle auch auf kostengünstigen und somit ressourcenschwachen Plattformen eingesetzt werden können.

Kabelgebundene Lösungen können vor allem bei Neubauten eingesetzt werden, um eine höhere Bandbreite gegenüber kabellosen Lösungen zu erzielen. Bei der Renovierung von älteren Gebäuden oder wenn kabelgebundene Lösungen zu kostenintensiv sind, können ebenfalls kabellose Lösungen zum Einsatz kommen. Um bei kabelgebundenen Lösungen die Anzahl der benötigten Kabelverbindungen und somit die Kosten zu minimieren, teilen sich meist viele Endpunkte in einer flachen Hierarchie einen Kommunikationsbus. Hierdurch steht pro Teilnehmer nur eine geringe Datenrate zur Verfügung. Beim Einsatz von Funklösungen ist die Bandbreite ohnehin durch die hohe Anzahl der Endpunkte begrenzt, da sich diese das gleiche Medium zur Übertragung teilen müssen. In beiden Fällen stehen den eingesetzten Systemen somit lediglich sehr geringe Bandbreiten zur Verfügung. Energieeffizient müssen alle eingesetzten Systeme sein, da ein zu hoher Energieverbrauch einer sehr großen Anzahl von Sensoren und Aktoren für das gesamte Gebäude keine Energieeinsparungen zur Folge hätte.

Durch die Entwicklung der im vorherigen Abschnitt dargestellten AAL-Systeme entstehen zunehmend Überlappungen zwischen der Gebäudeautomatisierung und dem Bereich AAL. Solche Überschneidungen finden sich beispielsweise bei den bereits genannten AAL-Komfortfunktionen (z.B. sich autonom öffnende und schließende Fenster und Türen), die mit Alarmanlagen aus dem Bereich der Gebäudeautomatisierung verschränkt werden können.

2.3 Zwischenfazit

In verschiedenen bestehenden (z.B. Gebäudeautomatisierung) und entstehenden (z.B. Ambient Assisted Living) Anwendungsgebieten wird durch die nahtlose Konnektivität eines heterogenen Geräteensembles mit höherwertigen Diensten ein Mehrwert geschaffen. Der Mehrwert kann sich dabei auf gesundheitliche, betriebs- oder volkswirtschaftliche Aspekte beziehen. Die entstehenden Infrastrukturen können als *Internet der Dinge* bezeichnet werden.

Die Vision des *Internets der Dinge* wird erst möglich, wenn alle beteiligten Endpunkte des Kommunikationsnetzes unabhängig voneinander nahtlos miteinander kommunizieren und interagieren können (vgl. Ende-zu-Ende Kommunikation). Die in diesem Kapitel aufgezeigten Anwendungsgebiete beinhalten jedes für sich bereits ein hohes Maß an Heterogenität und Diversität bezüglich der Systeme, Geräte, Szenarien, Anforderungen und Anwendungen. Diese Heterogenität und Diversität wird bei der Entstehung des *Internets der Dinge* noch verstärkt, da die bislang untereinander getrennten Anwendungsgebiete und Domänen zahlreiche Verschränkungen erfahren. In einem *Internet der Dinge* steht somit nicht die Komplexität der angebotenen Dienste im Vordergrund, sondern die ressourcenunabhängige horizontale Kombination bislang getrennter, vertikaler Dienste und Bereiche. So können beispielsweise bestehende Sensoren der Gebäudeautomatisierung genutzt werden, um das Verhalten und die Gewohnheiten der Bewohner aufzuzeichnen. Bei ungewöhnlichen Abweichungen, wie dem Fehlen jeglicher Aktivitäten innerhalb des Wohnumfeldes, kann das AAL-System von einer Gesundheitsgefährdung ausgehen und autonom Reaktionsketten auslösen.

Proprietäre Kommunikationslösungen einzelner Geräte- oder Systemhersteller sind für die aufgezeigten Einsatzgebiete nicht geeignet. Vielmehr müssen die genutzten Protokolle auf offenen Standards basieren und ein hohes Maß an Flexibilität und Erweiterbarkeit aufweisen. Dennoch müssen die Protokolle auf verschiedenen Klassen von Plattformen eingesetzt werden können und somit von ressourcenstarken Systemen bis hin zu batteriebetriebenen Kleinstgeräten skalieren.

Bezüglich der batteriebetriebenen Kleinstgeräte resultieren die Ressourcenbeschränkungen nicht immer aus der begrenzten Energie (siehe Energy Harvesting [55]), sondern können sich aufgrund der Netzwerkarchitektur oder anderen nicht-technischen Anforderungen ergeben.

In dieser Arbeit wird eine Architektur konzipiert, die den genannten Anforderungen in einer autonom agierenden M2M-Umgebung genügt. Dazu wird im nachfolgenden Kapitel der aktuelle Stand der Technik aufgezeigt und eine Bewertung der zur Verfügung stehenden Technologien und Protokolle vorgenommen.

3 Grundlagen und verwandte Arbeiten

Inhaltsübersicht

Bei der Entwicklung von Protokollen zur Vernetzung haben die beiden Domänen des Internets und der gerätenahen Kommunikation gänzlich unterschiedliche Ausprägungen erfahren. Diese wurden vor allem durch die Geschäftsmodelle aus diesen beiden Domänen bestimmt.

Gerade im Bereich der Heim- und Gebäudeautomatisierung stellt die Möglichkeit und Leistungsfähigkeit der Gerätevernetzung oft ein Alleinstellungsmerkmal dar. Da die Vernetzungslösungen auf proprietären Standards basieren, führt dies zur Kundenbindung an den jeweiligen Hersteller. Je größer die Produktpalette des Geräteherstellers ist, desto mehr unterschiedliche Geräte stehen in einer vernetzten Umgebung zur Verfügung und desto eher wird dieser Gerätehersteller von den Kunden favorisiert. Mit den proprietären Lösungen und der teilweise fehlenden nahtlosen Anbindung externer Dienste lässt sich meist nur ein *Intranet der Dinge* und kein ganzheitliches *Internet der Dinge* realisieren. Es entstehen somit Netzwerkinfrastrukturen, bei denen Geräteensembles durch dedizierte Stellvertreter und Protokollumsetzer mit dem weltweiten Kommunikationsnetz verbunden werden müssen. Die Geräte werden aber nicht integraler Bestandteil des Internets. Je unterschiedlicher die grundlegenden Prinzipien der Protokolle und Datenstrukturen ist, umso schwergewichtiger sind die Protokollumsetzer. Im schlechtesten Fall muss bei jeder Änderung der Netzwerktopologie oder der Anwendung auch der Stellvertreter und die Protokollumsetzung angepasst werden. Dies führt zu unflexiblen Insellösungen und hemmt die Vision des *Internets der Dinge* nachhaltig (siehe Abbildung 3.1) [8].

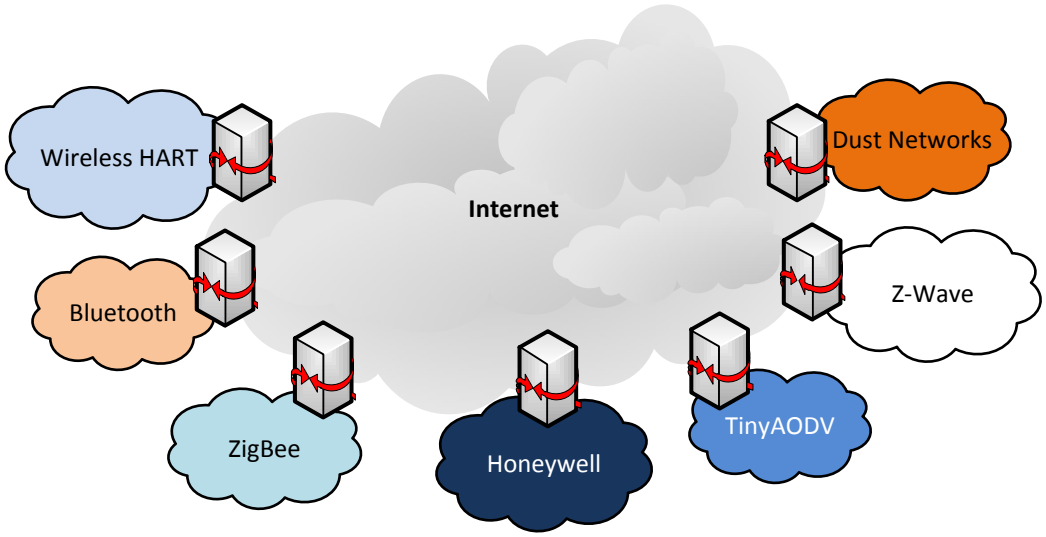

Abbildung 3.1 Insellösungen mit schwergewichtigen Protokollumsetzern

Im Internet stellt die domänenübergreifende Vernetzung von Nutzern und Diensten zwar eine Grundvoraussetzung dar, ist aber nicht zwingend Teil des Geschäftsmodells. Der Mehrwert wird durch die angebotenen Dienste und Daten selbst erzielt. Da die Protokolle nicht Teil des Geschäftsmodells sind, kommen im Internet eher offene Standards zum Einsatz [56].

Zur Vermeidung von Insellösungen und zur Realisierung einer nahtlosen Ende-zu-Ende Konnektivität zwischen Endpunkten [57], bei der auf schwergewichtige Stellvertreter verzichtet werden kann, wird eine Abstraktion vom konkreten Übertragungsmedium, der Netzwerktopologie und Änderungen innerhalb der Netzwerkinfrastruktur erforderlich. Im Internet wird hierfür das Internet Protokoll (IP) genutzt. Da IP auf der Vermittlungsschicht des OSI-Referenzmodells [30] angesiedelt ist, wird bereits auf einer sehr niedrigen Schicht die notwendige Homogenität für interoperable Kommunikation erreicht. An den Schnittstellen zwischen verschiedenen Netzwerken sind lediglich einfache Router nötig, die deutlich leichtgewichtiger sind als die zuvor beschriebenen komplexen Stellvertreter und Protokollumsetzer. Weiterhin stellt IP die Grundlage für ganzheitliche Basisarchitekturen in verteilten Umgebungen dar. Ein Beispiel für einen solchen ganzheitlichen Ansatz sind Service-orientierte Architekturen [40]. In Abbildung 3.2 ist die Interoperabilität, zum einen basierend auf dem Einsatz von IP und zum Zweiten basierend auf dem Einsatz von anwendungsspezifischen Protokollumsetzern (Gateways), in Form eines Protokollstapels, grafisch dargestellt. Während IP-basierte Infrastrukturen bereits über

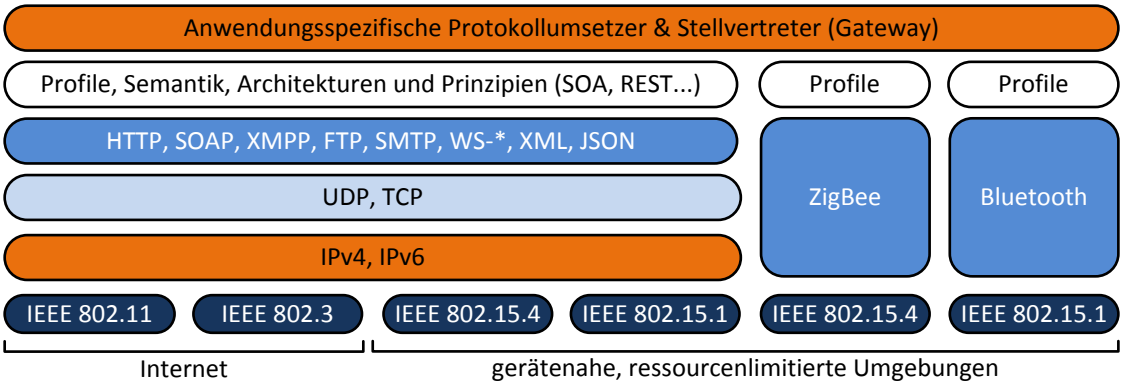

Abbildung 3.2 Interoperabilität zwischen Geräten und Internetdiensten

Homogenität auf den unteren Schichten verfügen, wird in Gateway-basierten Systemen diese Homogenität erst auf der Anwendungsschicht erreicht.

Das hohe Maß an Heterogenität und Diversität in zukünftigen Anwendungen lässt einen Trend erkennen, bei dem zur Vernetzung von Geräten Internettechnologien und –protokolle eingesetzt werden [11, 26, 58–60]. Auch hier bildet IP die Grundlage für Interoperabilität zwischen Geräten und höherwertigen Internetdiensten, auf der weiterführende Protokolle aufsetzen können.

Der in diesem Kapitel aufgeführte Stand der Technik bezieht sich daher auf die relevanten Internettechnologien. Der spezielle Fokus liegt auf stark ressourcenlimitierten Kleinstgeräten (vgl. drahtlose Sensornetzwerke). Bislang können diese Plattformen nur durch schwergewichtige und unflexible Protokollumsetzer bzw. Stellvertreter in höherwertigere, vernetzte Umgebungen integriert werden [8, 45]. Basierend auf den dargestellten Technologien wird in Kapitel 4 eine ganzheitlichere Netzwerkarchitektur entwickelt, bei der komplexe Gateway-Infrastrukturen vermieden werden.

3.1 Internettechnologien für die Gerätevernetzung

In vielen Szenarien ressourcenlimitierter Umgebungen werden die Plattformen durch Batterien mit Energie versorgt. Um den Energieverbrauch zu senken und somit die Lebensdauer des Gerätes innerhalb eines Ladezyklus‘ zu verlängern, müssen nicht benötigte Hardwarekomponenten so oft wie möglich in energiesparende Zustände versetzt werden. Die Kommunikation benötigt auf den Zielplattformen anteilig die meiste Energie während des Betriebes [61]. Dabei ist es nicht relevant, ob gerade Daten gesendet (TX) bzw. empfangen (RX) werden oder die Sende- und

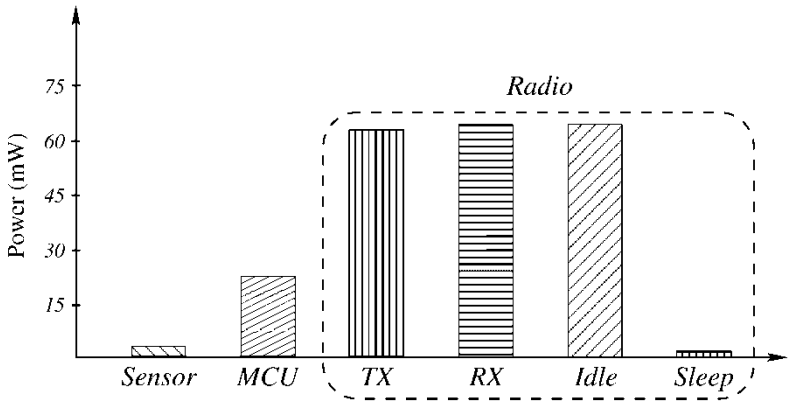

Abbildung 3.3 Aufteilung Energieverbrauch drahtlose Sensorknoten nach Akyildiz [12]

Empfangseinheit lediglich eingeschaltet ist (Idle). Der Energieverbrauch der Recheneinheiten (engl. Mikrocontroller Unit; MCU) bzw. des Speichers ist, wie in Abbildung 3.3 dargestellt, gegenüber allen drei Zuständen (RX, TX, Idle) in der Regel zu vernachlässigen [12]. Dies ist darin begründet, dass im Gegensatz zu den digitalen Komponenten der MCU, die analogen Verstärkerschaltungen für eingehende Signale der Empfangseinheiten immer Energie benötigen, sobald diese den energiesparenden Zustand verlassen.

Daher wurden im Forschungsbereich der drahtlosen Sensornetzwerke diverse Zugriffsverfahren auf den unteren Schichten des OSI-Referenzmodells entwickelt, die jeweils an spezifische Umgebungen und Netzwerktopologien angepasst sind [62, 63]. Die Zugriffsverfahren beinhalten verschiedenartige Mechanismen, um die Zeiten untereinander zu synchronisieren, in denen einzelne Endpunkte ihre Sende- und Empfangseinheiten aktivieren müssen bzw. für bestimmte Zeit deaktivieren können. Durch die Nutzung von Internettechnologien und somit IP für die Entkopplung der Zugriffs- bzw. Übertragungsschicht von höheren Schichten betrifft das Aktivieren bzw. Deaktivieren der Sende- und Empfangseinheiten vorrangig die genannten unteren Schichten. Die oberhalb von IP liegenden Schichten des OSI-Referenzmodells sind von den Spezifika der Zugriffsverfahren unabhängig [61].

Die Betrachtung der speziellen Zugriffsverfahren ist nicht Bestandteil dieser Arbeit. Im Rahmen der Untersuchungen dieser Arbeit ist die qualitative Aussage ausreichend, dass eine größere Menge zu übermittelnder Daten einen höheren Energieverbrauch zur Folge hat. Auf quantitative Untersuchungen des konkreten Energieverbrauchs wird im Rahmen dieser Arbeit verzichtet. Die Protokolle auf höheren Schichten und deren Implementierungen müssen die Auswirkungen der

Zugriffsverfahren allerdings dahingehend beachten, als dass bei der Kommunikation hohe Latenzen und Paketverluste auftreten können und die Schlafzustände somit indirekt auf der Anwendungsschicht sichtbar werden. Die eigentlichen Zustände der Sende- und Empfangseinheiten bzw. die Zugriffsverfahren müssen aber von den oberen Schichten nicht weiter beachtet werden.

3.1.1 Standardisierungsgremien

Die vier wichtigsten Standardisierungsgremien im Kontext von Internettechnologien, sind das Institute of Electrical and Electronics Engineers (IEEE), die Internet Engineering Task Force (IETF), das World Wide Web Consortium (W3C) und die Organization for the Advancement of Structured Information Standards (OASIS). Die einzelnen Gremien haben bei der Standardisierung der einzelnen Protokolle jeweils voneinander getrennte Aufgaben- und Zuständigkeitsbereiche. Diese Einteilung ist in Abbildung 3.4 grafisch dargestellt.

Das IEEE befasst sich vornehmlich mit Lösungen auf der Zugriffsschicht. Die bekanntesten Protokolle bzw. Protokollfamilien des IEEE sind unter den Begriffen Ethernet und Wireless Local Area Network (WLAN) bekannt.

Die IETF ist von den genannten Standardisierungsgremien, nach der IEEE, die zweitälteste Institution. Aus der IETF gingen weitverbreitete Protokolle wie IPv4, IPv6, TCP, UDP, HTTP und SMTP hervor. Seit 2005 befasst sich die IETF auch mit dem Einsatz von IP-basierten Technologien in stark ressourcenlimitierten Umgebungen.

Trotz der Standardisierungsbemühungen der IETF waren im Verlauf der Entwicklung des Internets zunehmend Inkonsistenzen zu beobachten, die sich vor allen in der fehlenden Vereinheitlichung auf der Anwendungsschicht und bezüglich der Datenrepräsentationen äußerten. Aus diesem Grund bildete sich mit dem W3C ein Zusammenschluss der namhaftesten Browserhersteller, die auch in diesen Bereichen interoperable Lösungen schaffen wollten. Bis heute sind die bekanntesten Lösungen des W3C die XML-Protokollfamilie, die Auszeichnungssprache Hypertext Markup Language (HTML) und die SOAP-basierten Web Services.

OASIS wurde ursprünglich als Konsortium von Anbietern ins Leben gerufen, die sich mit der Standard Generalized Markup Language (SGML) befassten. Inzwischen standardisiert OASIS vornehmlich anwendungsspezifische Protokolle und Profile, wobei oft auf Lösungen der zuvor

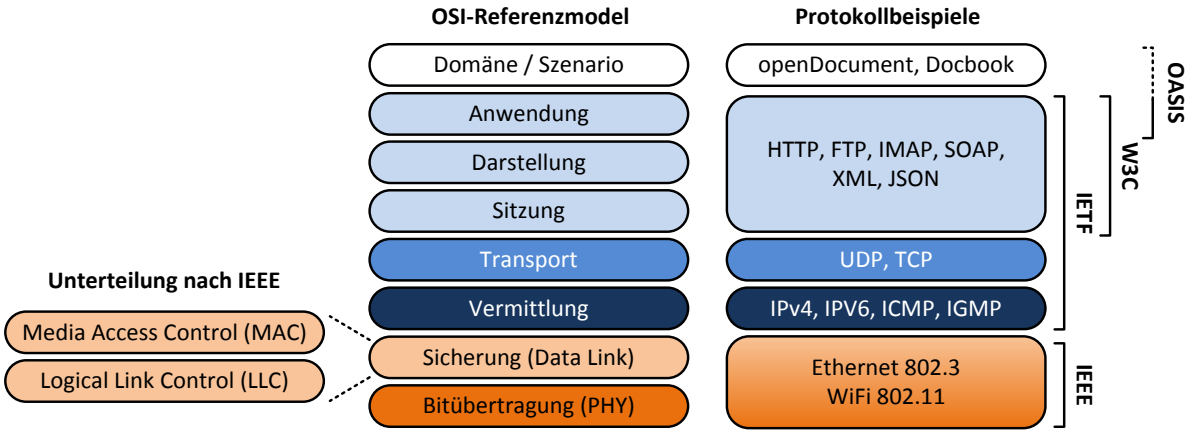

Abbildung 3.4 OSI-Referenzmodell und Aufteilung Standardisierungsgremien

genannten Standardisierungsgremien zurückgegriffen wird. Die im Rahmen dieser Arbeit wichtigste Lösung von OASIS ist das Devices Profile for Web Services (DPWS), welches ein auf SOAP Web Services basiertes Profil für gerätenahe Anwendungen darstellt.

Das Institut für Angewandte Mikroelektronik und Datentechnik der Universität Rostock, an dem diese Arbeit entstanden ist, ist Mitglied der Arbeitsgruppe der OASIS, in der DPWS spezifiziert wird. Weiterhin betätigt sich der Autor dieser Arbeit als aktives und beitragendes Mitglied dedizierter Arbeitsgruppen innerhalb der IETF, die sich mit dem Einsatz von IP-basierten Protokollen in stark ressourcenlimitierten Umgebungen befassen.

Die WS4D-Initiative (Web Services for Devices) ist eine Plattform zur Verbreitung von relevanten Forschungsergebnissen aus dem Bereich der eingebetteten Web Services und wird ebenfalls unter anderem vom Institut für Angewandte Mikroelektronik und Datentechnik der Universität Rostock vorangetrieben. Die bearbeiteten Themenschwerpunkte von WS4D sind somit stark an die Standardisierung innerhalb der IETF, des W3C und der OASIS gekoppelt. Weiterhin wurden und werden die in dieser Arbeit entstandenen Implementierungen im Rahmen von WS4D als Open Source publiziert.

3.1.2 Zugriffsschicht

Die Zugriffschicht bezeichnet die beiden untersten Schichten innerhalb des OSI-Referenzmodells und kann in die physikalische Bitübertragungsschicht (engl. Physical Layer, PHY) sowie die Sicherungsschicht (engl. Data Link Layer) unterteilt werden. Innerhalb von IEEE wird die

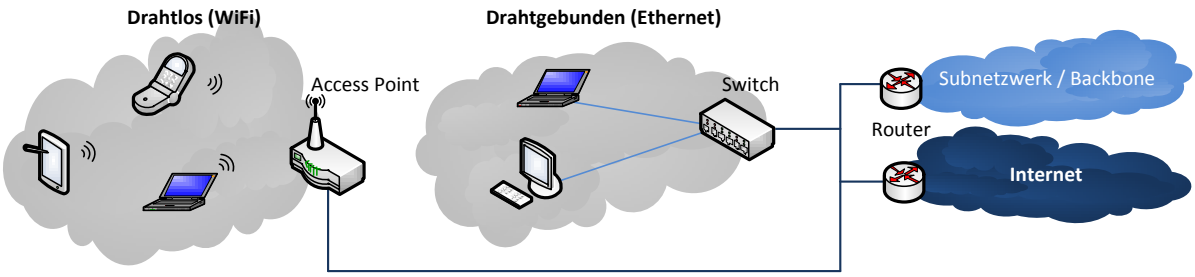

Abbildung 3.5 Topologie drahtgebundener und drahtloser Netzwerke

Sicherungsschicht weiterhin in die beiden Bereiche Logical Link Control (LLC) und Media Access Control (MAC) unterteilt.

Ethernet - IEEE 802.3

Die Ethernet-Protokollfamilie ist eine der bekanntesten Vertreter auf der Zugriffschicht. Ethernet wird innerhalb der IEEE Arbeitsgruppe 802.3 bearbeitet, die dieser Protokollfamilie auch die Bezeichnungen der einzelnen Standards vererbt. Da Ethernet eine kabelgebundene Übertragungstechnologie darstellt, können vergleichsweise hohe Datenraten erreicht werden. Zum Zeitpunkt des Entstehens dieser Arbeit sind für Ethernet Standards mit Übertragungsraten von 10 MBit/s bis zu 10 GBit/s spezifiziert.

Ethernet wird derzeit vornehmlich genutzt, um die Vernetzung innerhalb von Gebäuden mit ausreichender Bandbreite zu ermöglichen. Obgleich auch ressourcenschwache Hardwareplattformen für Ethernet existieren, kommt diese Technologie vorrangig in Bereichen zum Einsatz, die nicht durch Ressourcenlimitierungen geprägt sind.

Wireless LAN – WiFi - IEEE 802.11

Die unter dem Begriff WLAN bekannte Protokollfamilie bildet das drahtlose Pendant zu dem kabelgebundenen Ethernet und wird in der IEEE Arbeitsgruppe 802.11 standardisiert. Innerhalb eines WLAN-Netzwerkes können sich einzelne Endpunkte mit dedizierten Routern verbinden, die in der Regel die Umsetzung zwischen drahtgebundenen und drahtlosen Übertragungsmedien realisieren.

Durch die inzwischen allgegenwärtige Nutzung von WLAN in verschiedenen Bereichen haben sich in den vergangenen Jahren zunehmend spezialisierte Lösungen für einzelne Anwendungen entwickelt. Hierzu zählen IEEE 802.11s für Netzwerke mit vermaschten Topologien und

IEEE 802.11p zum Einsatz von WLAN in Personenkraftfahrzeugen bzw. intelligenten Verkehrssystemen. Zukünftig sind Lösungen zu erwarten, die besonders energiesparende Eigenschaften besitzen. Dennoch hat sich die WLAN-Protokollfamilie bislang kaum in stark ressourcenlimitierten Umgebungen durchgesetzt. Grund ist auch hier der vergleichsweise hohe Energiebedarf.

Die beiden Technologien WiFi und Ethernet werden in heutigen Infrastrukturen meist gemeinsam eingesetzt (siehe Abbildung 3.5). Jedes drahtlose bzw. drahtgebundene Netzwerk bildet dabei auf den unteren Schichten eine Sterntopologie und ist über einen Access Point bzw. Switch miteinander gekoppelt. Router verbinden die Netzwerke untereinander. Die Elemente Access Point, Switch und Router werden in einfachen Infrastrukturen oft in einem einzelnen Gerät vereint. Dadurch befinden sich alle Netzwerkteilnehmer innerhalb des gleichen Subnetzwerkes, unabhängig davon, ob eine drahtlose oder drahtgebundene Konnektivität vorherrscht.

Wireless Personal Area Network (WPAN) - IEEE 802.15.4

Da weder Ethernet noch WLAN durch energiesparende Eigenschaften geprägt sind, werden in der IEEE Arbeitsgruppe 802.15.4 Lösungen entwickelt, die speziell für den Einsatz in drahtlosen, energiearmen und vermaschten Netzwerkumgebungen optimiert sind. Basierend auf IEEE 802.15.4 werden von großen Industriekonsortien unter anderem Protokolle wie ISA SP100 und ZigBee spezifiziert [64]. Vor allem ZigBee wird in der Literatur oftmals fälschlicherweise synonym mit IEEE 802.15.4 verwendet.

Der Hauptmerkmale von IEEE 802.15.4 bestehen, neben den energiesparenden Eigenschaften, in geringen Herstellungskosten, Mechanismen für sich selbst heilende Netzwerke und einer enorm

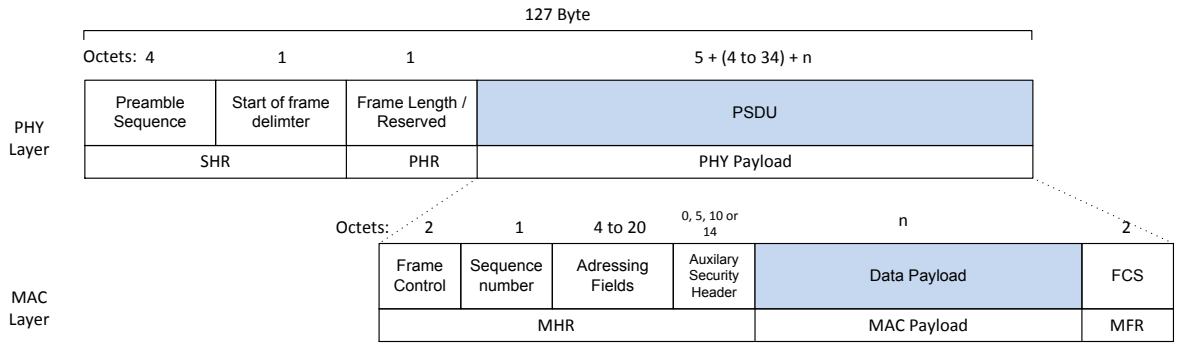

Abbildung 3.6 IEEE 802.15.4 Protokollkopf

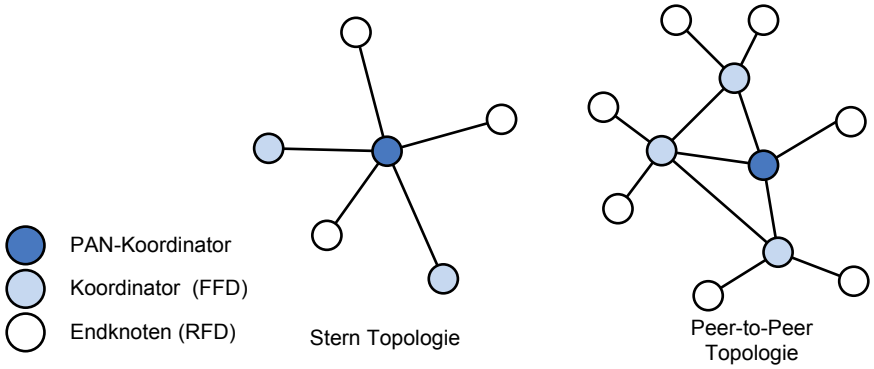

Abbildung 3.7 IEEE 802.15.4 Topologien

hohe Skalierbarkeit. Zur Sicherstellung der hohen Skalierbarkeit unterstützt 802.15.4 unterschiedliche Topologien (siehe Abbildung 3.7), bei denen ressourcenstärkere Full Functional Devices (FFD) Koordinatoren bzw. Aggregatoren darstellen und ressourcenschwächere Reduced Functional Devices (RFD) die Endpunkte.

IEEE 802.15.4 unterschiedet sich von den zuvor genannten Protokollfamilien Ethernet und WLAN weiterhin in der Größe der Datenpakete, die den oberen Schichten zur Übertragung zur Verfügung stehen (vgl. Protocol Data Unit, PDU). Bei den derzeit existierenden 802.15.4 Standards beträgt die Größe der Datenpakete auf der physikalischen Übertragungsschicht lediglich 127 Byte (siehe Abbildung 3.6). Die geringen Herstellungskosten und die energiesparenden Eigenschaften implizieren weiterhin geringe Datenraten von bis zu maximal 250 kBit/s.

3.1.3 Vermittlungsschicht

Die Vermittlungsschicht (engl. Network Layer) befindet sich auf Ebene 3 des OSI-Referenzmodells. Zu den Funktionen auf dieser Schicht zählen unter anderem das Bereitstellen netzwerkübergreifender Adressen, die Fragmentierung von Paketen und das Routing. Das Routing beinhaltet den Aufbau und die Aktualisierung von Routingtabellen zur Laufzeit. Das am weitesten verbreitete Protokoll auf dieser Schicht ist das Internet Protokoll in den Versionen 4 und 6, besser unter den Abkürzungen IPv4 und IPv6 bekannt sind.

Die genannten Protokolle IPv4 und IPv6 beschreiben in vernetzten Umgebungen eine komplette Gruppe von Standards und Hilfsprotokollen, die aufeinander aufbauen und sich teilweise gegenseitig bedingen. Beispielsweise kann das Internet Control Message Protocol (ICMP) von Routern und IP-Endpunkten eingesetzt werden, um Informationen bzw. Fehlermeldungen über den

Zustand des Netzwerkes austauschen zu können. Das Internet Group Management Protocol (IGMP) hingegen dient zur Verwaltung von Multicast-Gruppen.

Internet Protokoll Version 4 - IPv4

IPv4 wurde bereits 1981 bei der IETF in [31] spezifiziert. Dieser RFC791 wurde inzwischen durch verschiedene zusätzliche Dokumente erweitert, ist im Kern aber bis heute gültig.

Zum Zeitpunkt der Entstehung von IPv4 war es allerdings nicht absehbar, dass das Internet in kürzester Zeit zu einem weltweiten Kommunikationsnetz heranwächst. Somit wurde der für IPv4 definierte Adressraum von etwa 4,3 Milliarden Adressen als angemessen eingeschätzt. Mit dem Einzug von Computern in die Haushalte nahm die Anzahl der verfügbaren Adressen mit hoher Geschwindigkeit ab. Im Jahr 2011 wurden in der Folge vom Réseaux IP Européens Network Coordination Centre (RIPE NCC) die letzten zur Verfügung stehenden Blöcke des IPv4-Adressraumes vergeben. Das RIPE NCC verwaltet das europaweite zentrale Register für IP-Adressräume.

Die IETF hat die Unzulänglichkeit von IPv4 bereits sehr früh erkannt und die Entwicklung eines Nachfolgers für IPv4 bereits Ende der 90er Jahre ausgeführt.

Internet Protokoll Version 6 - IPv6

Der Kern von IPv6 wurde im Jahr 1998 in RFC2460 [32] veröffentlicht. Der bekannteste Unterschied zwischen IPv4 und IPv6 besteht im immens vergrößerten Adressraum, mit dem es möglich ist, $3{,}4 \times 10^{38}$ Endpunkte zu adressieren. Dies sind genügend Adressen, um jedem Quadratmillimeter der Erdoberfläche mindestens $6{,}65 \times 10^{17}$ IP-Adressen [65] bereitzustellen.

IPv6 beinhaltet aber auch auf konzeptioneller und funktionaler Ebene einige Weiterentwicklungen gegenüber IPv4. Die stetig wachsende Anzahl von Endpunkten im Internet impliziert eine zunehmende Last für die Router. Daher war die Lastreduzierung für Router bei der Entwicklung von IPv6 eines der entscheidenden Kriterien. Hierzu wurde beispielsweise die insgesamt 128 Bit breite IPv6 Adresse zu gleichen Teilen in einen Präfix und einen Interface Identifier aufgeteilt. Das Präfix definiert das Subnetz. Der Interface Identifier beschreibt das Interface des Endpunktes im Subnetz. Während die Präfixe von den Routern verwaltet und den Endpunkten mitgeteilt werden, können die Interface Identifier von den Endpunkten selbst generiert werden. Eine verbreitete Methode hierfür ist das Nutzen der vorhandenen Adressen auf der Zugriffschicht

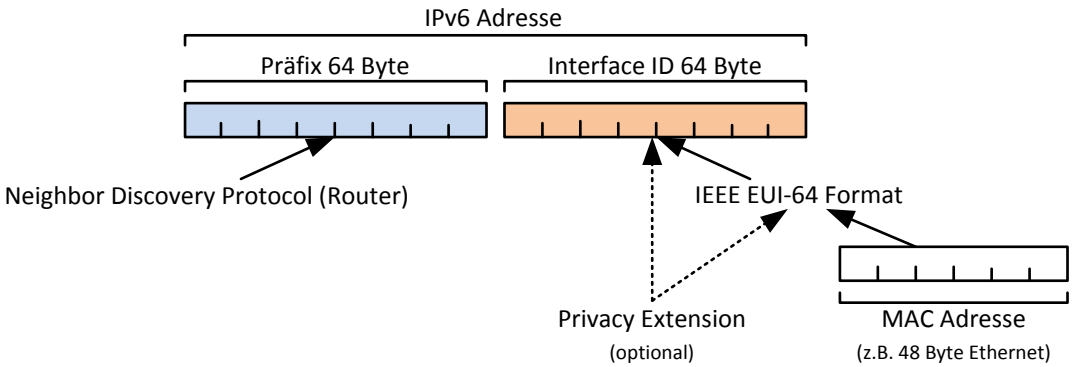

Abbildung 3.8 Aufbau einer IPv6-Adresse am Beispiel Ethernet

(z.B. MAC-Adresse), um daraus einen gültigen Interface Identifier zu generieren (siehe Abbildung 3.8). In IPv4 war diese Aufteilung zwischen Präfix und Suffix dynamisch und vom jeweiligen Subnetz abhängig.

Weiterhin wird bei der Übermittlung der IPv6-Pakete keine Fragmentierung in den Routern vorgenommen, wie es bei IPv4 der Fall ist. Der Absender ist dafür zuständig, IPv6-Pakete mit einer maximalen Größe zu versenden, die von allen Zwischenstationen; einschließlich des Empfängers; unterstützt wird. Die mindestens von allen IPv6-fähigen Geräten und Routern unterstützte Maximum Transmission Unit (MTU) beträgt 1280 Byte. Das bedeutet, dass Pakete von weniger als 1280 Byte nie fragmentiert werden müssen. Für alle Pakete, die größer als 1280 Byte sind, muss der Sender entweder ein Path MTU Discovery durchführen oder die Pakete in entsprechend kleinere Pakete aufteilen.

Im Gegensatz zu der Vergrößerung des Adressraumes von 32 Bit auf 128 Bit ist die Größe des gesamten Protokollkopfes von IPv4 zu IPv6 nicht im gleichen Verhältnis um das 4-fache angestiegen. Dies liegt darin begründet, dass die in IPv4 nur selten genutzten und somit nicht benötigen Datenfelder bei IPv6 entweder gar nicht vorgesehen oder in IPv6-Erweiterungs-Kopfdaten (engl. Extension Header) ausgelagert wurden. Die Erweiterungs-Kopfdaten sind optional und können je nach Bedarf an den IPv6-Protokollkopf angehangen werden.

Obgleich ein immenser Bedarf nach einem vergrößerten Adressraum auf der Vermittlungsschicht besteht, hat sich IPv6 in der breiten Masse bislang kaum gegenüber IPv4 durchgesetzt. Gründe hierfür sind die vergleichsweise hohen Investitionskosten für die Umrüstung und die Möglichkeit,

durch Technologien wie Network Address Translation (NAT) den Adressraum von IPv4 künstlich zu vergrößern. Durch die Vergabe des letzten freien IPv4-Adressblockes kann allerdings ein vermehrter Einsatz von IPv6 in den kommenden Jahren erwartet werden. Dennoch werden die beiden Protokolle sehr wahrscheinlich für mehrere Jahre parallel betrieben [66, 67].

IPv6 over Low power Wireless Personal Area Networks - 6LoWPAN

Die IETF hat bereits sehr früh die Notwendigkeit erkannt, dass IP-basierte Kommunikation auch in stark ressourcenlimitierten Umgebungen möglich ist. In einem *Internet der Dinge* sollen diese stark ressourcenlimitierten Kleinstgeräte einen essentiellen Anteil am weltweiten Datenfluss haben.

Das Akronym 6LoWPAN (IPv6 for Low Power Wireless Personal Area Networks) ist neben dem Protokoll zugleich die Bezeichnung der zuständigen IETF Arbeitsgruppe. Ziel der 6LoWPAN-Arbeitsgruppe war es, einen Standard zu definieren, mit dessen Hilfe IPv6-Pakete über IEEE 802.15.4 effizient übertragen werden können. Im Verlauf der Entwicklung wurden die 6LoWPAN-Protokolle zum großen Teil unabhängig von IEEE 802.15.4. Daher wurden vielmehr konkrete Anforderungen an die Zugriffsschicht definiert, die diese erfüllen muss, damit die Anpassungen gegenüber nativem IPv6 greifen. In der Realität exponiert sich IEEE 802.15.4 derzeit

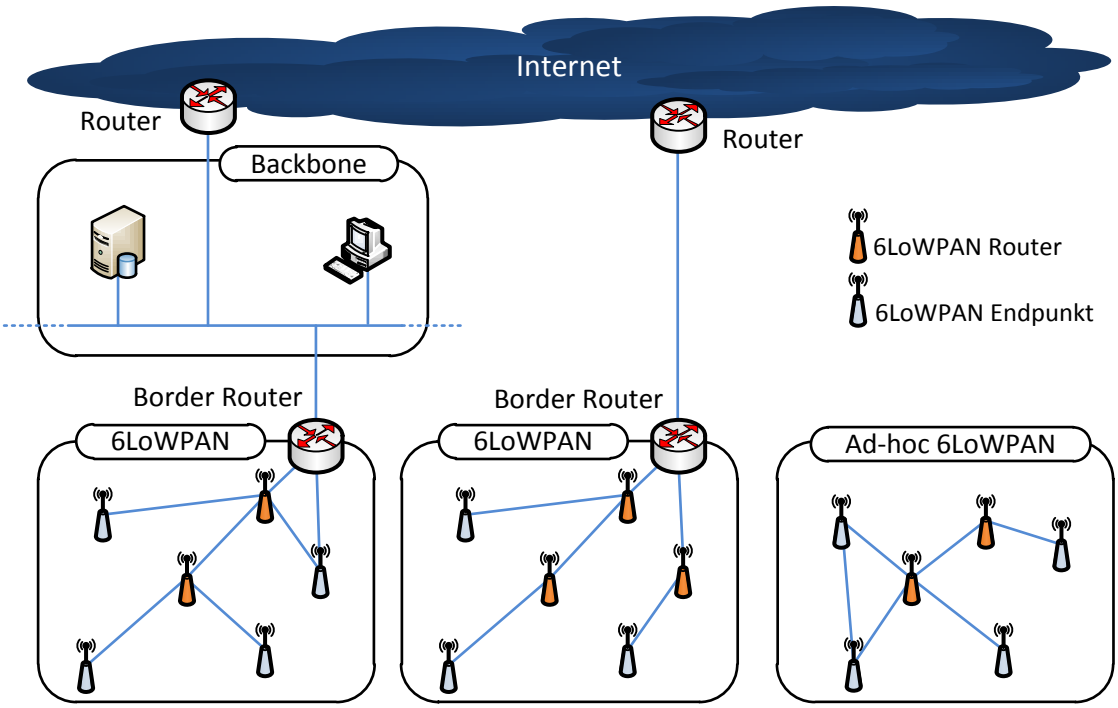

Abbildung 3.9 6LoWPAN Topologien

als das einzige Protokoll auf Zugriffschicht, welches diese Anforderungen erfüllt. Aufgrund des zu geringen Adressraumes von IPv4 und weiterer konzeptioneller Vorteile von IPv6 (z.B. Entlastung der Router, Generierung der Interface Identifier aus MAC-Adresse) wurde durch die IETF ausschließlich eine IPv6-basierte Lösung spezifiziert.

Es gilt hervorzuheben, dass die 6LoWPAN-Protokolle einzig die Vermittlungsschicht bzw. konkrete Anpassungen von nativem IPv6 darstellen und somit höhere Schichten nicht beeinflussen. Die 6LoWPAN-Protokolle stellen für die höheren Schichten die identischen Funktionalitäten wie natives IPv6 bereit und sind somit transparent. Die 6LoWPAN-Protokolle müssen allerdings beim Übergang zwischen dem 6LoWPAN und anderen IPv6-Netzwerken in die entsprechenden Pendants umgesetzt werden. Somit bilden 6LoWPANs eigenständige Sub-Netzwerke, die über definierte Verknüpfungspunkte zu anderen Netzwerken verfügen. Diese Verknüpfungspunkte werden als *Border Router* bezeichnet und unterscheiden sich von den *Routern* im drahtlosen und vermaschten 6LoWPAN dahingehend, als dass *Border Router* Konnektivität sowohl in das 6LoWPAN als auch in das externe Netzwerk besitzen und die Vermittlung zwischen diesen Netzwerken vornehmen. Dennoch sind die Funktionen der *Border Router* ausschließlich auf der Vermittlungsschicht angesiedelt und beeinflussen bei der Umsetzung der Protokolle höhere Schichten nicht. Die Abbildung 3.9 stellt typische Topologien für 6LoWPANs dar.

Die Anpassungen, die an IPv6 vorgenommen wurden, können in drei Bereiche unterteilt werden: (1) Protokollkopf-Kompression, (2) Fragmentierung und (3) Neighbor Discovery.

(1) *Protokollkopf-Kompression:* Zugriffschichten wie 802.15.4 verfügen meist nur über sehr kleine PDUs. Ein IPv6-Protokollkopf hat eine typische Größe von mindestens 40 Byte (exklusive Extension Header). Um IPv6 effizienter über

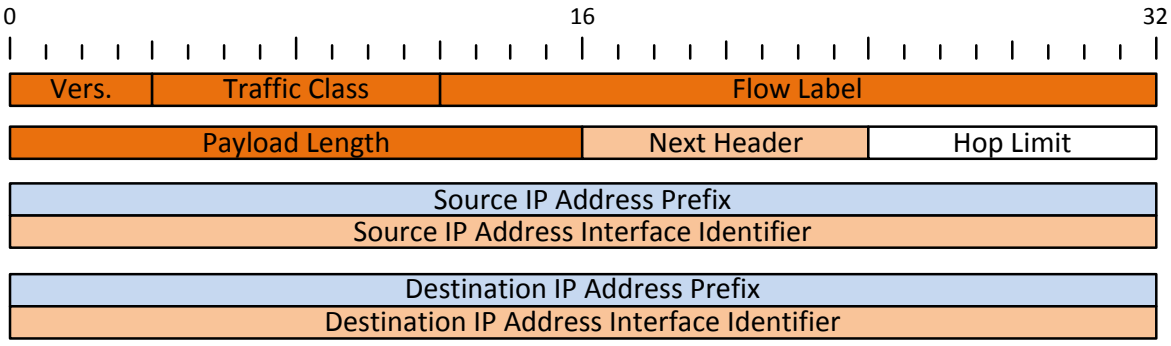

Abbildung 3.10 6LoWPAN Protokollkopf-Kompression

802.15.4 übertragen zu können, wurde eine Protokollkopf-Kompression definiert. Der Zweck dieser Kompression besteht darin, redundante Informationen zu vermeiden, die implizit bekannt oder bereits in der Zugriffschicht enthalten sind. Die Kompression der Protokollköpfe bildet die Grundlage für alle weiteren 6LoWPAN-Protokolle.

Abbildung 3.10 stellt die Funktionsweise der 6LoWPAN-Kompression eines IPv6-Protokollkopfes beispielhaft grafisch dar:

- Die *Version* ist immer IPv6. Das Feld wird nicht übermittelt.
- Die Felder *Traffic Class* und *Flow Label* sind auf 6LoWPANs aufgrund der vermaschten Topologie und somit andersartigen Routingstrategien nicht anwendbar und werden im Regelfall nicht übermittelt.
- Das Feld *Payload Length* ist redundant, da diese Information bereits im Protokollkopf der Zugriffsschicht IEEE 802.15.4 enthalten ist. Das Feld wird nicht übermittelt.
- Das *Next Header* Feld bleibt nur im Protokollkopf enthalten, wenn ein *IPv6 Extension Header* verwendet wird.
- Das Feld *Hop Limit* kann nicht komprimiert werden und bleibt vollständig enthalten.
- Die *Präfixe* der Quell- und Zieladresse können den Endpunkten bereits bekannt sein und können für diesen Fall weggelassen werden (z.B. beide Endpunkte befinden sich im gleichen 6LoWPAN-Netzwerk, für das ein gemeinsamer Präfix verwendet wird).
- Die *Interface Identifier* der Quell- und der Zieladresse können ebenfalls redundant sein, da diese teilweise aus den MAC-Adressen der Endpunkte generiert werden.

(2) *Fragmentierung:* IPv6 fordert eine MTU von mindestens 1280 Byte. Da IEEE 802.15.4 diese nicht erreicht, wurde ein Fragmentierungsmechanismus definiert, mit dem auf Layer 2.5 (zwischen Zugriffs- und Vermittlungsschicht) die geforderte MTU realisiert werden kann. Auf den unteren Schichten werden somit kleine und energieeffiziente Pakete verwendet. Für die höheren Schichten stehen dem IPv6-Standard entsprechende Paketgrößen zur Verfügung.

(3) *Neighbor Discovery:* Eine weitere grundlegende Änderung gegenüber dem ursprünglichen IPv6 ist das Neighbor Discovery, mit dem ein Knoten den IPv6-Präfix und weitere Netzwerkkonfigurationsdaten erhält. In herkömmlichen Netzwerken hat jeder Endpunkt eine direkte Verbindung zum Router, welcher für die Verwaltung und Vergabe der IP-Adressen zuständig ist. Vermaschte 6LoWPANs ermöglichen, dass der für das gesamte Subnetzwerk zuständige Border Router mehr als einen physikalischen Hop entfernt ist. Um der Unterscheidung zwischen den Routern und dem Border Router in der 6LoWPAN-Topologie gerecht zu werden, waren daher beim Neighbor Discovery für 6LoWPANs spezielle Anpassungen nötig, die in Abbildung 3.11 schematisch dargestellt sind.

Die aufgezeigten Anpassungen von IPv6 für ressourcenlimitierte vermaschte Netzwerke werden beispielsweise in der Dissertation von Hui [61] und in [8, 68] ausführlich beschrieben. Obgleich durch 6LoWPAN verschiedenste Anpassungen gegenüber nativem IPv6 notwendig sind, ist entscheidend, dass diese Anpassungen für die Anwendungen und Protokolle auf höheren Schichten transparent sind. Ein nativer IPv6-Endpunkt kann mit Endpunkten im 6LoWPAN kommunizieren,

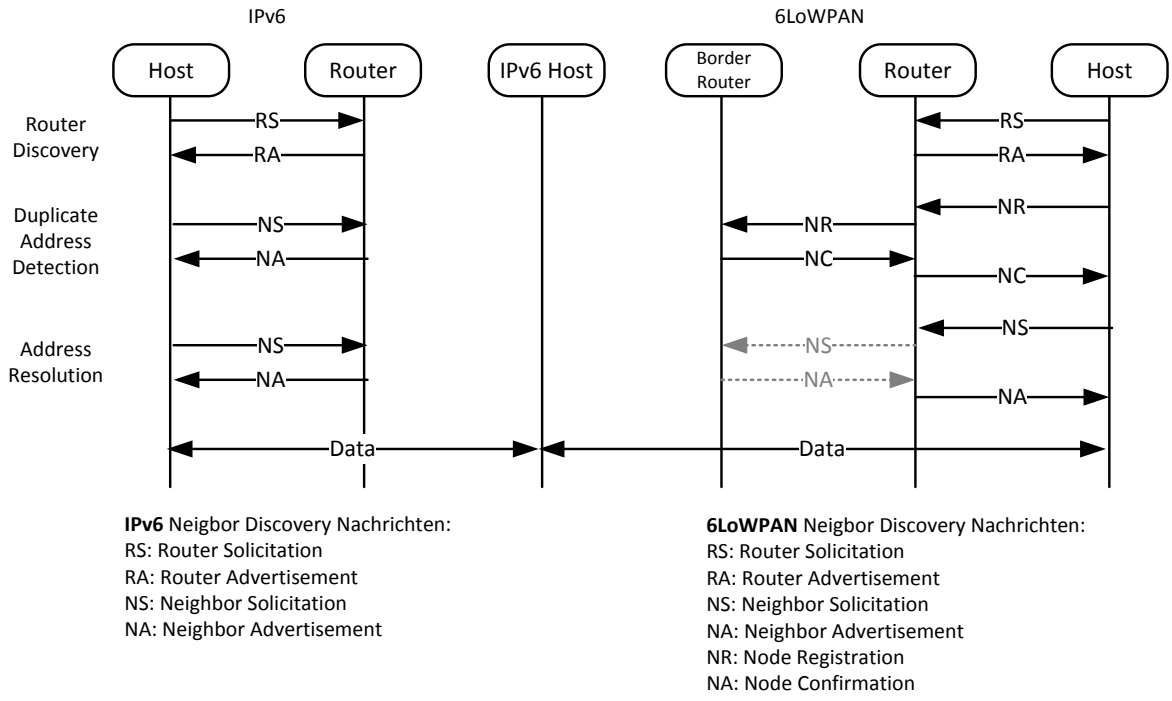

Abbildung 3.11 IPv6 vs. 6LoWPAN Neighbor Discovery

ohne die 6LoWPAN-Protokolle selbst unterstützen zu müssen. Den höheren Schichten des Protokollstapels auf den Endpunkten im 6LoWPAN stehen ebenfalls alle Funktionalitäten von IPv6 zur Verfügung, ohne dass diese Endpunkte die Anpassungen gegenüber nativem IPv6 beachten müssen.

Routing Over Low Power and Lossy Networks - ROLL

Durch die Entwicklung der 6LoWPAN-Protokolle entsteht ein Bedarf an einem standardisierten und ressourceneffizienten Routingprotokoll für 6LoWPAN-Netzwerke. Eine Lösung wird durch die IETF ROLL (Routing Over Low Power and Lossy Networks) Arbeitsgruppe hervorgebracht. Das entstehende Protokoll trägt den Namen RPL (IPv6 Routing Protocol for Low Power and Lossy Networks; gesprochen „Ripple“) [35, 36, 69]. RPL ist ein IPv6-Routingprotokoll und daher auf der Vermittlungsschicht angesiedelt. Dem Design von RPL liegen konkrete Anwendungsszenarien zugrunde, die einen spezifischen Kommunikationsfluss beschreiben. Dabei wird von einem Kommunikationsmuster ausgegangen, bei dem die Sensoren eines 6LoWPANs Daten aufnehmen und diese aus dem 6LoWPAN hinaustransportieren. Die Daten können während des Transports auf speziellen Datenaggregatoren zusammengefasst werden. Eine Kommunikation in das 6LoWPAN hinein bzw. eine direkte Interaktion von Knoten untereinander wird zwar unterstützt, RPL ist darauf aber nicht optimiert.

Innerhalb der Routingtopologie ist die lokale Senke, zu der die Daten im 6LoWPAN-Netzwerk transportiert werden, in den meisten Szenarien mit dem Border Router des 6LoWPANs gleichzusetzen, der das 6LoWPAN mit einem anderen Netzwerk verbindet. Die Senke kann aber auch jeder andere Knoten des 6LoWPANs sein. Zu der Senke hin wird durch RPL ein gerichteter azyklischer Graph aufgebaut. Die Wurzel des Graphen ist die Senke. Ein Netzwerk kann mehrere Senken haben, allerdings richtet sich ein einzelner RPL-Graph immer auf genau eine Senke aus. Daher können in einem 6LoWPAN auch mehrere solcher RPL-Graphen parallel existieren, um einem Knoten die Kommunikation mit mehreren Senken zu ermöglichen. Ein Datenpaket muss in einem solchen Fall beim Absenden einem Graphen zugeteilt werden und kann entlang des Pfades den Graphen nicht wechseln. Ein einzelner RPL-Graph kann auf eine oder mehrere Metriken optimiert werden. Mögliche Metriken sind beispielsweise die Anzahl von Hops oder der Batteriezustand der Zwischenknoten (Router). Die Routingtopologie von RPL stellt Abbildung 3.12 schematisch dar.

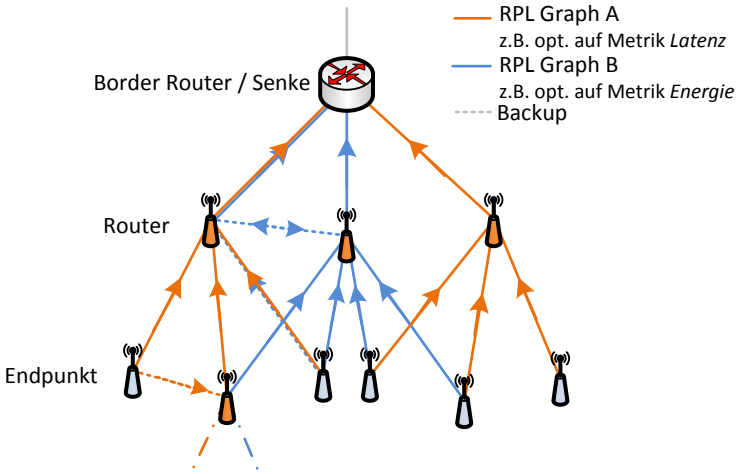

Abbildung 3.12 RPL Topologie der Graphen

Für eine Kommunikation in das Netzwerk hinein muss sich jeder Knoten seinem übergeordneten Vaterknoten (Default Router, näher an der Senke) mitteilen. Der Vater hat folglich Kenntnis über das gesamte über ihn erreichbare Subnetzwerk. Diese Informationen teilt jeder Router der Wurzel (Border Router) mit. Die Wurzel hat somit als einziger Knoten Kenntnis über das gesamte Netzwerk. Für die Kommunikation in das Netzwerk hinein wird daher von der Wurzel beginnend ein Source Routing Verfahren genutzt, bei dem alle zu traversierenden Zwischenstationen beim Absenden in das Paket eingefügt werden. Um zu vermeiden, dass bei der Kommunikation der 6LoWPAN-Knoten untereinander ein Datenpaket immer vollständig bis zur Wurzel traversieren muss und von dort wieder durch Source Routing heruntertraversiert, können die Vaterknoten bzw. Router die Routingtabellen über das ihnen zugehörige Subnetzwerk auch selbst verwalten. Dieser optionale Modus ermöglicht es, dass Datenpakete nur bis zum nächsten gemeinsamen Vater bzw. Router transportiert werden müssen.

3.1.4 Transportschicht

Oberhalb der Vermittlungsschicht des OSI-Referenzmodell befindet sich die Transportschicht. Zu den Aufgaben der Transportschicht gehören die Stauvermeidung und die Segmentierung des Datenstromes. Die im Internet am häufigsten eingesetzten Protokolle der Transportschicht sind das Transmission Control Protocol (TCP) [3] und das User Datagram Protocol (UDP) [70]. Der grundlegende Unterschied zwischen UDP und TCP liegt in der Verbindungslosigkeit bzw. Verbindungsorientierung der beiden Protokolle. Andere Protokolle als UDP und TCP haben sich im Internet bislang nicht durchgesetzt und bilden lediglich Insellösungen für

anwendungsspezifische Intranets [71, 72]. Die Protokolle sind unabhängig von der Anzahl der zu traversierenden physikalischen Zwischenstationen und ermöglichen eine direkte Ende-zu-Ende Konnektivität.

Transmission Control Protocol - TCP

TCP ist das ältere der beiden Protokolle und wurde erstmals 1974 in RFC 675 bei der IETF erwähnt. 1981 wurde mit RFC 793 das Protokoll so spezifiziert, wie es, abgesehen von einigen kleineren Aktualisierungen, bis heute eingesetzt wird. TCP ist ein zuverlässiges, verbindungsorientiertes, paketvermitteltes Übertragungsverfahren. Dadurch ist TCP in der Lage Verbindungs- und Übermittlungsfehler zu erkennen und diese durch geeignete Maßnahmen zu korrigieren. Weiterhin beinhaltet TCP Mechanismen zur Flusskontrolle und somit zur Stauvermeidung. Hierzu ist es notwendig, die Ende-zu-Ende Verbindung zwischen den beiden Endpunkten kontrolliert auf- und abzubauen. Dabei kommen spezielle Handshake-Verfahren zum Einsatz, welche in Abbildung 3.13 dargestellt sind. Einige der Kontrollnachrichten und die Nachrichten zur Datenübertragung können zur Übermittlung in einem Paket zusammengefasst werden. Dennoch sind für eine komplette TCP-Übertragung, inklusive Auf- und Abbau der Verbindung, mindestens 6 Pakete zu übertragen (6 Pakete sind bereits ein Sonderfall, bei dem die Verbindung simultan von beiden Endpunkten beendet wird).

Die Handshake-Verfahren verursachen einen nicht zu vernachlässigenden Overhead bezüglich der Anzahl der auszutauschenden Kontrollnachrichten, wenn nur sehr geringe Datenmengen übertragen werden sollen. Weiterhin ist TCP nur auf Unicast-Kommunikation zwischen zwei Endpunkten beschränkt.

User Datagram Protocol - UDP

Im Gegensatz zu TCP ist UDP verbindungslos und somit nicht zuverlässig. UDP kann als leichtgewichtige Alternative gegenüber TCP also immer nur dann eingesetzt werden, wenn beispielsweise die Übermittlung nicht garantiert sein muss, keine Flusskontrolle benötigt wird oder Mechanismen auf anderen Schichten diese Funktionen realisieren. Da bei UDP keine Kontrollnachrichten ausgetauscht werden müssen, sondern die zu übermittelnden Daten direkt in das Einzige zu übermittelnde Datenpaket eingebettet werden (siehe Abbildung 3.14), entsteht durch UDP nur ein sehr geringer Overhead. Weiterhin beschränkt sich UDP nicht auf eine

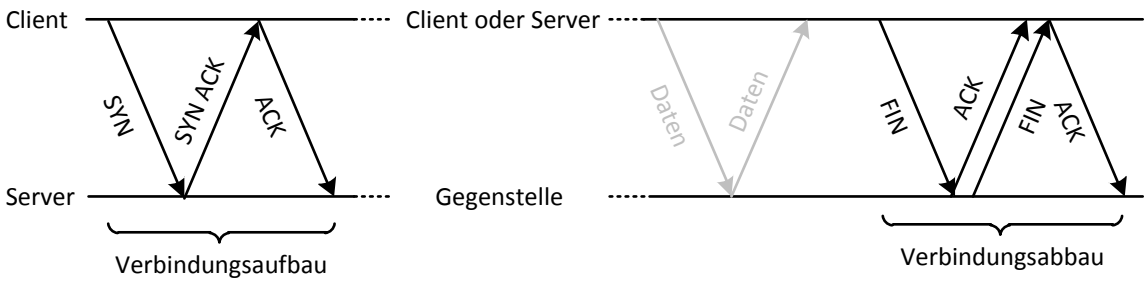

Abbildung 3.13 TCP Verbindungsaufbau und -abbau

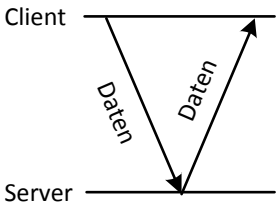

Abbildung 3.14 UDP Verbindungsablauf

Unicast-Übermittlung, sondern kann auch zur Multicast- und Broadcast-Übertragung eingesetzt werden.

Im Kontext der 6LoWPAN-Protokolle muss hervorgehoben werden, dass für UDP in RFC6282 [73] eine Protokollkompression beschrieben wird. Für TCP existiert eine solche Kompression bislang nicht.

3.1.5 Anwendungsschicht

Oberhalb der Transportschicht befinden sich im OSI-Referenzmodell die Sitzungs-, Darstellungs- und Anwendungsschicht. In heutigen Anwendungen kommen zur Realisierung dieser Schichten oft Protokolle zum Einsatz, die alle drei Schichten gleichzeitig umsetzen. Daher werden die drei genannten Schichten im Allgemeinen und somit ebenfalls im Rahmen dieser Arbeit unter dem Begriff Anwendungsschicht bzw. Anwendungsschichtprotokolle zusammengefasst. Viele Anwendungsschichtprotokolle unterstützen verschiedene Datenrepräsentationen, die im OSI-Referenzmodell auf der Darstellungsschicht angesiedelt sind. Da die optimierte Datenrepräsentation einen Schwerpunkt dieser Arbeit darstellt, wird im Anschluss an diesen Unterabschnitt die Datenrepräsentation als gesonderter Punkt erläutert.

In verteilten Anwendungen wird auf der Anwendungsschicht oft der Begriff *Dienst* genutzt. Der Begriff *Dienst* wird dabei in der Literatur fälschlicherweise oft mit der Basisarchitektur SOA bzw. konkreten Technologien in Verbindung gebracht (siehe Diskussion in Kapitel 1.2). Ein *Dienst* bezeichnet allerdings eher eine abstrakte Funktion, die als geschlossene Komponente genutzt werden kann. Der Aufruf eines *Dienstes* geschieht über konkrete Protokolle. Einige Anwendungsschichtprotokolle, die solche Dienstaufrufe abbilden können, werden im Folgenden vorgestellt.

Dienste können weiterhin in *Operationen* bzw. *Methoden* unterteilt werden. Diese *Operationen* bzw. *Methoden* definieren unter anderem die Semantik der durchzuführenden Aktion. Dies können einfache CRUD-basierte Manipulationen sein (Create, Read, Update, Delete). Die *Operationen* bzw. *Methoden* können aber auch weiterführende und komplexere Funktionalitäten abbilden.

Mit den hier aufgeführten Protokollen ist es möglich Client-Server-Modelle umzusetzen. Bei diesem Modell werden Dienste von einem Server angeboten und können von Clients genutzt werden. Ein Server, der einen oder mehrere Dienste bereitstellt, ist somit immer passiv und führt ohne Anfragen bzw. Aufforderungen von Clients keine Aktionen aus. Im Rahmen dieser Arbeit wird synonym für einen Server bzw. Dienstanbieter auch die Bezeichnung *Gerät* genutzt. Der Hintergrund hierfür wird im Kontext des Devices Profile for Web Services (DPWS) noch ausführlich erläutert.

Es ist möglich, dass Endpunkte zugleich Server und Client sind. Beispielsweise kann ein Server Dienste bereitstellen, die eine Kapselung und Zusammenführung von mehreren Diensten anderer Endpunkte darstellen. In einem solchen Fall wird der Endpunkt als Peer bezeichnet. In einem vollständig gleichberechtigten Netzwerk, bei dem alle Endpunkte Dienste anbieten und zugleich nutzen können, spricht man von einem Peer-to-Peer Modell. Obgleich die Realisierung eines Peer-to-Peer Netzwerkes mit den hier aufgezeigten Protokollen möglich ist, liegt darin nicht der Hauptfokus dieser Arbeit.

Hypertext Transfer Protocol - HTTP

Das Hypertext Transfer Protocol (HTTP) [74] bildet eines der Basisprotokolle des Internets wird ist sehr ausführlich in der Dissertation von Roy Fielding beschreiben, der in [39] ebenfalls die konzeptionellen und architektonischen Hintergründe zu HTTP erläutert. Am bekanntesten ist HTTP für das Abrufen von Internetseiten mit Hilfe von Webbrowsern. HTTP ermöglicht es aber

auch, andere Daten als Hypermedien (vgl. Webseiten) zu übertragen. Ein Beispiel für die erweiterte Nutzung von HTTP ist WebDAV [75].

Die bekanntesten Konzepte von HTTP sind Uniform Resource Locators (URL) zur Identifizierung der Ressourcen bzw. Daten, vereinheitlichte Methoden bzw. Operationen (GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS, CONNECT) und die Statuscodes, um Rückmeldungen über z.B. Erfolg bzw. Fehler des Aufrufes zu übermitteln. Weiterhin ist HTTP unabhängig von den zu übertragenden Datentypen und Repräsentationsformen (vgl. Media Type) und verfügt über Caching-Mechanismen, um die benötigte Skalierbarkeit im Internet bereitzustellen.

Da HTTP ohne Erweiterungen ein zustandsloses und somit unzuverlässiges Protokoll wäre, nutzt es auf unteren Schichten TCP für den Transport.

Constrained Application Protocol - CoAP

Das Constrained Application Protocol (CoAP) [8, 37] ist ein vergleichsweise neues Anwendungsschichtprotokoll, welches zum Zeitpunkt des Entstehens dieser Arbeit von der IETF für den Einsatz in ressourcenlimitierte Umgebungen entwickelt wird. Basierend auf CoAP wird in Kapitel 7 ein neuartiger Transportmechanismus für SOAP-basierte Web Services vorgestellt. Eine ausführlichere Erläuterung zu CoAP findet daher in Kapitel 7.3.1 statt.

Grundsätzlich ist CoAP für den Einsatz in 6LoWPANs vorgesehen. CoAP ist aber von den unteren Protokollen wie 6LoWPAN und RPL unabhängig und kann z.B. auch in existierenden IPv4-Netzwerken eingesetzt werden.

CoAP ist stark an HTTP angelehnt und verfügt über einen ähnlichen Aufbau. Dies beinhaltet z.B. Ressourcen als Informationsträger, Manipulation von Ressourcen mittels Methoden (GET/PUT/POST/DELETE), Response Codes, Media Types, URLs und Caching (siehe Abbildung 3.15). Dadurch ist es möglich, CoAP direkt in HTTP umzusetzen. Hierfür wurden in CoAP spezielle Proxymechanismen definiert. Der Grundgedanke ähnelt dabei dem des 6LoWPAN Border Routers. Innerhalb des 6LoWPANs können effiziente und optimierte Protokolle genutzt werden, die beim Übergang in ein anderes Netzwerk transparent und zustandslos in ihre nativen Pendants überführt werden können. Die nahtlose Umsetzung von CoAP zu HTTP in dedizierten Proxys unterscheidet CoAP von den meisten existierenden, teilweise proprietären, Protokollen für stark ressourcenlimitierte Umgebungen.

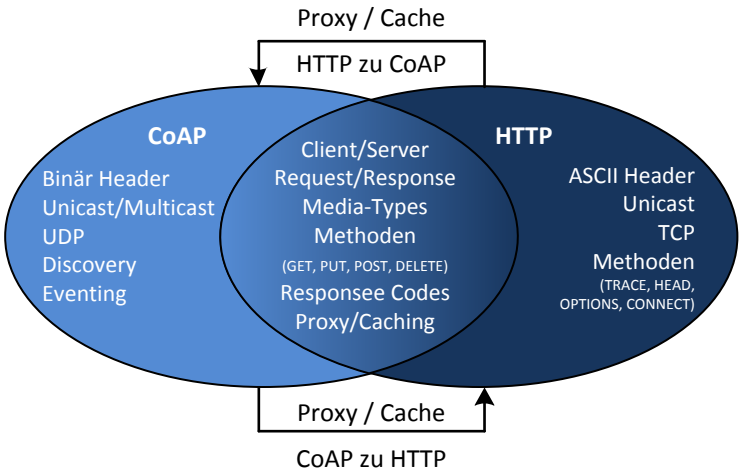

Abbildung 3.15 Gegenüberstellung CoAP und HTTP

CoAP kodiert alle Protokollköpfe im Gegensatz zu HTTP binär, wodurch die benötigte Bandbreite deutlich sinkt. Ein typischer CoAP-Protokollkopf umfasst ca. 4-20 Byte. Der gleiche Protokollkopf benötigt in HTTP durch die ASCII-Kodierung (American Standard Code for Information Interchange) mehr als 100 Byte. Da TCP für 6LoWPANs keine optimale Flusskontrolle und Fehlerbehandlung bietet [8], unterscheidet sich CoAP von HTTP auf Transportebene durch die Nutzung des leichtgewichtigeren UDP anstelle von TCP. Aus diesem Grund ist es notwendig, in CoAP Mechanismen zur Herstellung der Transportverlässlichkeit zusätzlich zu integriert. Durch die Nutzung von UDP bleibt CoAP nicht wie HTTP auf Unicast beschränkt, sondern kann auch über Multicast gesendet und empfangen werden. Ebenfalls sind rudimentäre Funktionen zum Suchen und Finden von Geräten (vgl. Discovery), eine Beschreibung der von den Geräten angebotenen Dienste und Daten (vgl. Metadaten) sowie einfache asynchrone Benachrichtigungen über Statusänderungen (vgl. Push bzw. Eventing) in CoAP integriert. Einige der Funktionen (z.B. asynchrone Benachrichtigungen) können nicht direkt in HTTP abgebildet werden und müssen mittels Protokollumsetzer auf andere Protokolle oder durch spezielle HTTP-Erweiterungen umgesetzt werden.

Web Services - WS-*

Die genannten Protokolle HTTP und CoAP sind besonders gut geeignet, um Daten und Hypermedien auszutauschen. Aufgrund der großen Verbreitung von HTTP ist CoAP stark daran angelehnt. HTTP ist allerdings für die Verwendung in H2M-Anwendungen mit vergleichsweise statischen Inhalten optimiert (vgl. Webseiten), wodurch beispielsweise Caching-Mechanismen zur

Skalierbarkeit von Systemen beitragen. Weiterführende Informationen zur Kontrolle bzw. Steuerinformationen, wie sie in dynamischen M2M-Anwendungen benötigt werden, können in den beiden Protokollen nur insoweit eingebettet werden, wie die vergleichsweise einfach strukturierten Protokollköpfe dies zulassen. In komplexeren Szenarien müssen viele domänen- oder anwendungsspezifische Steuerinformationen gemeinsam mit den Nutzdaten in der Nutzlast von HTTP bzw. CoAP in strukturierter Form eingebettet werden. Für diese Einbettung existieren für die beiden genannten Protokolle aber keine standardisierten Beschreibungsformen, Datenmodelle oder –formate.

Das World Wide Web Consortium (W3C) hat ein komplettes Web Services (WS) Protokoll-Framework spezifiziert, welches z.B. gegenüber HTTP bzw. CoAP Funktionen realisiert, die über einfache, statische Szenarien hinausgehen [25]. Das WS-Framework definiert dazu kein monolithisches Protokoll, sondern eine Ansammlung von grundlegenden Protokollbausteinen, die einer gemeinsamen Architektur unterliegen und unabhängig voneinander kombiniert werden können. Diese Architektur ist es, die das W3C WS-* Framework von anderen Protokollen unterscheidet. Während bei Protokollen wie HTTP die Interoperabilität auf Erfahrungen und Best Practices (dt. bewährte Verfahren) basieren, ist diese im Bereich der W3C Web Services durch die Prinzipien der zugrundlegenden Architektur sichergestellt. Die Prinzipien und die Architektur sind dazu verbindlich definiert und in einer Spezifikation formuliert. Alle Protokollbausteine müssen sich in diese Web Services Architecture (WSA) [25] einfügen, wodurch die jeweiligen Protokollkomponenten sowohl zu einer vertikalen, domänenspezifischen als auch zu einer horizontalen, domänenunabhängigen Lösung kombiniert werden können. Die Möglichkeit interoperable Erweiterungen in die Protokollmechanismen und –funktionen einzubringen, ist somit bereits grundlegender Bestandteil.

Das atomare Element der WSA ist die Nachricht. Im Vergleich dazu sind die atomaren Elemente einer allgemeingültigen SOA bzw. ROA die Dienste bzw. Ressourcen. Eine Ressource wiederum kann mittels eines Dienstaufrufes manipuliert werden. Eine Nachricht im Sinne der WSA kann beides abbilden, sowohl einen Dienstaufruf als auch eine Ressource. Die Abstraktion mittels Nachrichten stellt nicht nur die Unabhängigkeit von Betriebssystemen, Programmiersprachen und Softwaretechniken sicher, sondern ermöglicht es ebenfalls, dass Transportdetails von den oberen Schichten verborgen werden können. Gemeinsame Grundlage der WSA-konformen Spezifikationen ist somit ein universelles Nachrichtenformat und –austauschprotokoll. Der

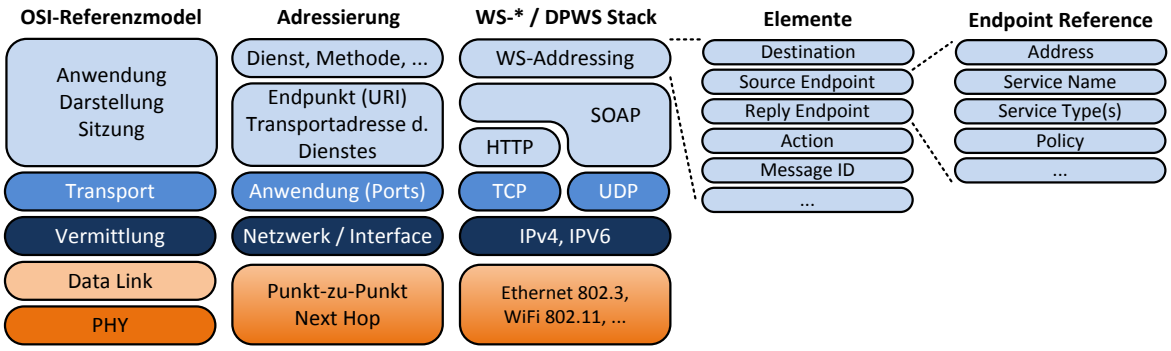

Abbildung 3.16 Adressierungsmechanismen aufgeteilt nach Schichten

Nachrichtencontainer ist die niedrigste Ebene, die innerhalb des WS-* Framework definiert wird. Eine weitverbreitete Möglichkeit die Nachrichtencontainer zu übertragen, liefert die Nutzung von HTTP als Übertragungsprotokoll. Grundsätzlich sind die Funktionen des WS-* Frameworks aber unabhängig von HTTP bzw. anderen darunter liegenden Schichten. Diese sind somit untereinander austauschbar. Eine Transportbindung an ein konkretes Protokoll wird als *Binding* bezeichnet.

Den zentralen Kern der WSA bildet SOAP [29] als universelles Protokoll und Container für den Austausch der Nachrichten, die meist in der Extensible Markup Language (XML) dargestellt sind. Daher werden Umsetzungen des WS-* Frameworks oft auch als SOAP Web Services oder XML Web Services bezeichnet. Der in einer Nachricht versendete Informationsträger wird als SOAP-Envelope bezeichnet. Ein SOAP-Envelope besteht aus einem Header und einem Body. Im Body werden die anwendungsspezifischen Daten und Strukturen eingebettet. Im Header sind die Kontrollstrukturen und -informationen der jeweils genutzten WS-* Spezifikationen enthalten. Einige der WS-* Spezifikationen werden im Folgenden beispielhaft dargestellt.

> **WS-Addressing.** WS-Addressing ist in den heutigen Umsetzungen fester Bestandteil der Untermenge des genutzten WS-Frameworks. WS-Addressing stellt Mechanismen bereit, die eine dauerhafte und eindeutige Ende-zu-Ende Adressierung von Endpunkten ermöglichen. Diese Mechanismen sind vollständig unabhängig von den darunterliegenden Schichten und Protokollen, die z.B. zum Transport genutzt werden (siehe Abbildung 3.16). Die entstehenden Konstrukte werden als *Endpoint Reference* bezeichnet und sind erweiter- sowie wiederverwendbar konzipiert, sodass andere Spezifikationen auf diese aufbauen können.

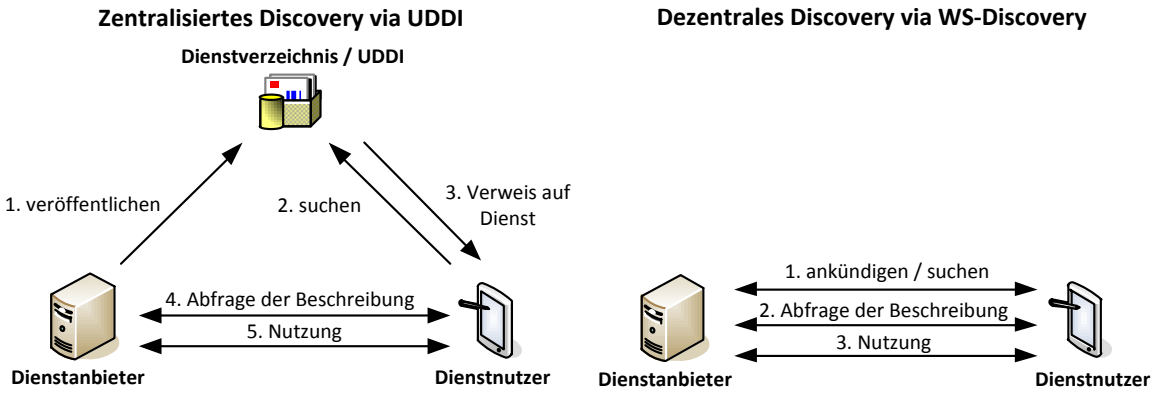

Abbildung 3.17 Gegenüberstellung zentrales und dezentrales Discovery

WS-Discovery. Der WS-Discovery Standard wird derzeit bei OASIS weiterentwickelt und befasst sich mit dem Auffinden von Endpunkten (Geräte, Dienste) in einem Netzwerk. In WS-Discovery wird dazu ein verteilter Ansatz genutzt, bei dem Gruppennachrichten (vgl. Multicast) von Servern bzw. Clients gesendet werden. Die Nachrichten kündigen die angebotenen Dienste an bzw. dienen dazu, nach diesen zu suchen. Damit bildet WS-Discovery das Gegenstück zu zentralisierten Verzeichnisdiensten wie Universal Description, Discovery and Integration (UDDI), bei denen jeder Server die angebotenen Dienste einer zentralen Entität mitteilt und Clients diese über die zentrale Entität suchen bzw. finden (siehe Abbildung 3.17).

WS-Eventing. Um wiederholtes Aufrufen und somit Polling zu verhindern, können Entwickler auf WS-Eventing zurückgreifen. Mittels WS-Eventing können asynchrone Benachrichtigungen über Zustandsänderungen durch den Server bzw. Dienst versendet werden. Eine solche Benachrichtigung wird von einer Event-Quelle zu einer oder mehreren Event-Senken gesendet und wird als *Event-Notification* bzw. Event-Benachrichtigung bezeichnet. Das Aussenden der *Event-Notification* an die jeweilige Event-Senke muss zuvor vom Client expliziert initiiert werden. Dieser Vorgang wird als *Event-Subscription* bezeichnet. Im Rahmen der *Subscription* werden zusätzlich dienst- bzw. anwendungsspezifische Filterparameter übertragen. Da WS-Eventing und SOAP unabhängig von den auf unteren Schichten genutzten Übertragungsprotokollen sind, wird im sogenannten *Eventing Delivery Mode* ebenfalls die Form der Übertragung im Rahmen der *Subscription* ausgehandelt.

WS-Transfer. Die WS-Transfer Spezifikation ist das WS-* Pendant zu HTTP bzw. CoAP und beschreibt grundlegende Funktionen, die mittels WS-Addressing adressierbare Ressourcen in Form von XML-Repräsentationen übertragen bzw. manipulieren zu können. Zur Manipulation werden einfache und wohldefinierte Operationen, wie CREATE, UPDATE und DELETE genutzt.

WS-MetadataExchange. Die Informationen zur Dienstbeschreibung, die zur Laufzeit zwischen zwei Endpunkten ausgetauscht werden müssen, werden als Metadaten bezeichnet. Diese generellen Beschreibungen der Dienste können als WS-Transfer Ressource dargestellt werden. Darauf aufbauend beschreibt WS-MetadataExchange unter anderem, wie Dienstbeschreibungen konkret als WS-Transfer Ressource abgebildet bzw. in diese Form überführt, die Metadaten mittels WS-Addressing adressiert und ebenso wie die Metadaten der Dienste mittels definierter Operationen abgerufen werden können. Die Metadaten können hierzu in einzelne Abschnitte unterteilt werden, um jeweils nur einzelne Aspekte aus der Gesamtheit der Metadaten eines Dienstes abzubilden (vgl. *Metadata Sections*).

Web Services Description Language (WSDL). Die WSDL ist das zur Entwicklungszeit am häufigsten genutzte Pendant zu dem zur Laufzeit verwendeten WS-MetadataExchange. Die Metasprache WSDL beschreibt dazu im XML-Format die angebotenen Dienste, deren Methoden, die Eingabe- bzw. Ausgabedaten, die Datentypen und -modelle sowie die unterstützen Transport-Bindings. WSDLs können beispielsweise dazu genutzt werden, um Quellcode zur Implementierung einer auf SOAP Web Services basierenden Anwendung automatisch zu generieren.

Um WSA-basierte Anwendungen untereinander interoperabel zu machen, muss die jeweilige konkrete Untermenge der genutzten WS-* Spezifikationen definiert sein. Hierfür werden von verschiedenen Standardisierungsgremien Profile spezifiziert. Diese Profile beinhalten (1) die Auswahl der genutzten bestehenden Standards, (2) wenn notwendig Einschränkungen der Freiheitsgrade der verwendeten WS-* Spezifikationen und (3) wenn erforderlich Erweiterungen gegenüber den genutzten Standards.

Devices Profile for Web Services – DPWS

Das Devices Profile for Web Services (DPWS) ist ein Profil, welches existierende WS-* Spezifikationen zusammenfasst und um weitere modulare Standards ergänzt. Derzeit wird DPWS in der Version 1.1 bei OASIS gepflegt [22].

DPWS ist speziell für den Einsatz in gerätenahen Umgebungen entwickelt und somit grundsätzlich ähnlich zu Universal Plug and Play (UPnP) [76]. DPWS ist aber nicht wie UPnP an einen spezifischen Anwendungsbereich gebunden. Ein weiterer wichtiger Unterschied zu UPnP besteht darin, dass DPWS auf den aktuellen Standards und Spezifikationen des W3C und von OASIS aufsetzt. Hierdurch lässt sich DPWS nahtlos in Systeme integrieren, die auf der Web Services Architecture (WSA) [25] basieren. Somit kann DPWS um Protokollbausteine aus dem W3C SOAP Web Services Protokoll-Framework und eigene Erweiterungen gleichermaßen ergänzt werden. Der grundlegende Protokollstapel von DPWS ist in Abbildung 3.18 dargestellt.

Da DPWS für gerätenahe Umgebungen entworfen wurde, ist in den Spezifikationen die Modellierung des abstrakten Konstrukts eines Gerätes mittels Diensten beschrieben. Die Dienste eines DPWS-konformen Gerätes werden als *Hosted Services* bezeichnet. Die *Hosted Services* sind zu nativen SOAP Web Services konform und können auch außerhalb des Kontextes von DPWS unabhängig von Clients genutzt werden. Ein zusätzlicher Dienst, den ein Gerät immer anbietet, um konform mit DPWS zu sein, ist der *Hosting Service*. Der *Hosting Service* repräsentiert das eigentliche Gerät und dessen Beschreibung. Die Elemente der Beschreibung sind durch DPWS spezifiziert und umfassen unter anderem den Hersteller, die Bezeichnung, die Seriennummer und

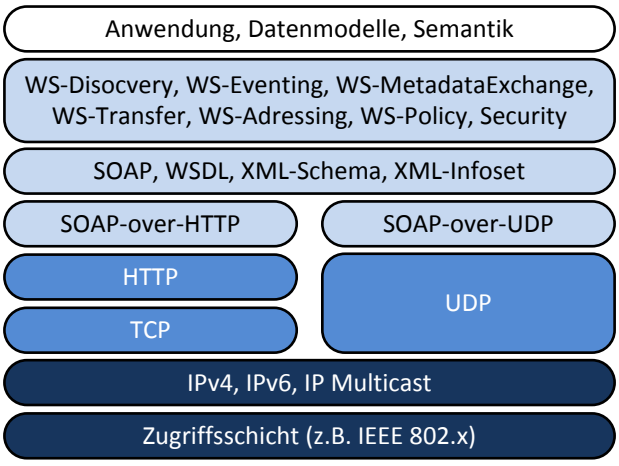

Abbildung 3.18 DPWS Protokollstapel

die Firmware-Version des Gerätes. Teil der Beschreibung des Gerätes ist ebenfalls die Referenz auf die vom Gerät angebotenen *Hosted Services*. Der *Hosting Service* und dessen Beschreibung können von Clients im Netzwerk mittels WS-Discovery gesucht bzw. gefunden werden.

Die Beschreibung des *Hosting Services* erfolgt entsprechend der WS-MetadataExchange Spezifikation und umfasst verschiedene *Metadata Sections*, die folglich mittels WS-Transfer Operationen abgerufen werden. Die Beschreibungen der *Hosted Services* hingegen können als generelle Dienste mittels WSDL erfolgen.

3.1.6 Datenrepräsentation

Die Auszeichnungssprache Extensible Markup Language (XML) hat sich vor allem in heterogenen Umgebungen durchgesetzt, da sie von Plattformspezifika (z.B. der Byte-Order) unabhängig ist. Daher lässt sich XML aus verteilten Anwendungen nicht mehr wegdenken. Das Geräteprofil DPWS stützt sich im Kern auf SOAP. SOAP nutzt zur Datenrepräsentation ebenfalls XML. Aus diesem Grund beschränkt sich dieser Unterabschnitt bei der Darstellung von Datenformaten auf die Grundlagen bezüglich XML und wie XML in SOAP-basierten Web Services Umgebungen eingesetzt wird.

Extensible Markup Language - XML

Der Begriff XML beschreibt eine ganze Familie von Spezifikationen, die vornehmlich vom W3C gepflegt wird. Bei XML wird zwischen (1) der *Serialisierung*, (2) der *Kodierung*, (3) der *Grammatik* und (4) dem *Vokabular* unterschieden.

Die *Serialisierung* beschreibt die Abbildung von abstrakten Objekten und Strukturen auf eine sequentielle und somit seriell übertragbare Form der Darstellung. Diese abstrakte Form des Datenmodells und die für die *Serialisierung* zur Verfügung stehenden sequentiellen Strukturelemente bzw. Informationsträger (Tags, Attribute, Namensräume, Kommentare usw.) werden durch die XML Information Set (XML-Infoset) Spezifikation [77] festgelegt.

Um XML einsetzen zu können, ist es erforderlich, das genutzte *Vokabular* und die *Grammatik* eines Dokumentes definieren zu können. Das *Vokabular* definiert unter anderem die zur Verfügung stehenden Bezeichner für Strukturelemente bzw. Informationsträger. Die *Grammatik* definiert die Reihenfolge und die Struktur der *Vokabeln*. Die Definition des *Vokabulars* und der *Grammatik* kann durch XML-Schema-Dokumente [78] realisiert werden. Auch die Definition von einfachen

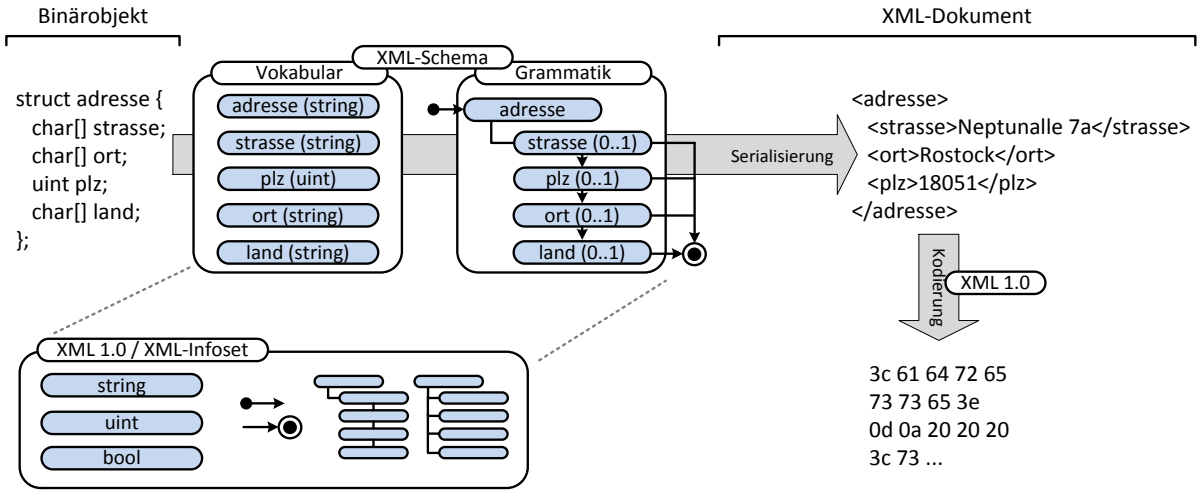

Abbildung 3.19 XML-Dokument: Vom Binärobjekt zum Datenstrom

und komplexen Datentypen ist ein Bestandteil der XML-Schema-Dokumente. Für das Nachrichtenprotokoll SOAP ist sowohl die Serialisierung aller Strukturen entsprechend XML-Infoset als auch die Nutzung von XML-Schema zur Beschreibung des Vokabulars und der Grammatik obligatorisch.

Um die einer *Grammatik* und einem *Vokabular* folgenden *serialisierten* Objekte übertragen zu können, müssen diese als Datenstrom *kodiert* werden. Das grundsätzliche, zur *Kodierung* genutzte Format wird in der XML 1.0 Spezifikation [79] definiert. Unicode ist der bekannteste Zeichensatz, der zur *Kodierung* von XML genutzt wird. Allerdings ist XML bei der *Kodierung* nicht auf Unicode festgelegt. Unterschiedlich *kodierte* Datenströme können daher identische logische Daten repräsentieren. Dies entkoppelt das Darstellungsformat der Nachrichten vom Datenmodell der *Serialisierung*. Die aufgeführten Begriffe und Spezifikationen sind in Abbildung 3.19 grafisch veranschaulicht.

Auf eine ausführlichere Diskussion der Teilbereiche von XML, deren Ausprägungen und Umsetzungen, wird im Rahmen dieser Arbeit aufgrund des Umfanges verzichtet. Weiterführende Informationen finden sich in der Literatur (siehe z.B. [80]).

3.2 Zwischenfazit

Ausgehend von der IETF 6LoWPAN Arbeitsgruppe existieren seit 2006 Standardisierungsbemühungen für den breiten Einsatz von IP-basierten Protokollen in stark

ressourcenlimitierten Umgebungen. Diese Umgebungen sind drahtlosen, vermaschten Sensornetzwerken bezüglich der Anforderungen sehr ähnlich, konnten aber vor der Spezifikation der 6LoWPAN-Protokolle nicht direkt dem weltweiten Kommunikationsnetz beitreten.

Aufbauend auf den 6LoWPAN-Protokollen können existierende Protokolle, wie beispielsweise UDP und TCP auf Transportschicht, eingesetzt werden. Vor allem auf der Anwendungsschicht genügen die existierenden Protokolle aber meist nicht den Anforderungen (z.B. geringe Bandbreite und wenig Speicher der Zielplattformen) und können ohne richtungsweisende Neukonzeptionierungen nicht in 6LoWPANs eingesetzt werden. Mit dem derzeit bei der IETF in der Entwicklung befindlichen CoAP-Protokoll steht Entwicklern in naher Zukunft ein optimiertes Protokoll zur Verfügung. Abstrakt kann CoAP, ähnlich wie die 6LoWPAN-Protokolle zu nativem IPv6, als binäres und ressourceneffizientes Pendant zu HTTP angesehen werden. CoAP unterscheidet sich von anderen binären Feldbusprotokollen darin, dass die zustandslose und transparente Umsetzung von CoAP in HTTP und umgekehrt in dedizierten Proxys integraler Bestandteil der Spezifikationen von CoAP ist.

Somit betreffen die Anpassungen der Protokolle in den entstehenden Umsetzungen nur das 6LoWPAN-Netztwerk. Existierende Implementierungen und Dienste in ressourcenstärkeren Umgebungen sind davon nicht betroffen und müssen nicht geändert werden. Dennoch können durch die gezielt zustandslose Umsetzungen der Protokolle ineinander Ende-zu-Ende Verbindungen zwischen den Endpunkten zur Kommunikation genutzt werden. Diese Ende-zu-Ende Konnektivität ist eines der grundlegendsten Prinzipien bei der Konzeption des Internets [57], welches sich als hochskalierbares Netzwerk herausgestellt hat. Dadurch wird die Infrastruktur entlastet und simplifiziert.

Protokolle wie HTTP und CoAP sind in datenlastigen Anwendungen die bevorzugte Wahl. Für komplexere, dynamische M2M-Anwendungen, bei denen von den Protokollen ein weiterführender Funktionsumfang benötigt wird, sind die einfachen Ausgestaltungen von HTTP und CoAP nicht ausreichend. Solche komplexeren Anwendungen in stark heterogenen Umgebungen benötigen beispielsweise umfassende Mechanismen zum Suchen und Finden von Diensten bzw. Geräten, effiziente asynchrone Benachrichtigungen über Zustandsänderungen sowie ganzheitliche Sicherheitskonzepte über die einfache Verschlüsselung bei der Übertragung innerhalb einer Sicherheitsdomäne hinaus. Nicht zuletzt bildet die flexible Erweiterbarkeit um Domänen-, Hersteller- oder Anwendungsspezifika eine nicht zu vernachlässigende, essenzielle Voraussetzung

für die Realisierung zukünftiger vertikaler wie auch horizontaler Szenarien. Trotz dieser Erweiterbarkeit muss die Interoperabilität der entstehenden Umsetzung durch die zugrundeliegenden Prinzipien sowie protokolleigene Mechanismen gewährleistet sein. Nur dadurch können heterogene gerätenahe und höherwertige Dienste unabhängig von Bandbreite, Ressourcen und Plattformen in einer homogenen und skalierbaren Netzwerkinfrastruktur und Dienstkomposition zusammengeführt werden.

In höherwertigen Diensteumgebungen haben sich hierfür die SOAP Web Services Protokolle des W3C durchgesetzt. DPWS bildet dabei ein spezielles Profil für gerätenahe Anwendungen. DPWS ist aber dennoch mit der WS-Architecture kompatibel und kann daher um existierende Lösungen aus dem WS-* Framework und individuelle Ergänzungen gleichermaßen erweitert werden.

4 Architektur, Infrastruktur, Designkriterien und Referenzszenario

Inhaltsübersicht

In verteilten Anwendungen kommen oft Vermittler oder Stellvertreter zum Einsatz. Eine der bekanntesten Arten von Vermittlern sind Caches, die sich vor allem in Internetanwendungen zur Entlastung von Servern und der Netzwerkinfrastruktur durchgesetzt haben. Dazu sind die Caches Teilnehmer der Kommunikation zwischen den Endpunkten. Hat ein vom Cache erkanntes Datum eine definierte Gültigkeitsdauer und trifft innerhalb der Gültigkeitsdauer eine identische Anfrage für das gleiche Datum ein, so kann der Cache diese aufgrund seines bestehenden Wissens stellvertretend beantworten. Obwohl der Zustand des Caches aufgrund vorheriger Aufrufe generiert wurde, ist der jeweilige Aufruf, der durch den Cache beantwortet wird, nicht vom vorherigen Aufruf abhängig. Ohne den Cache würde der Aufruf zu einer identischen Antwort führen. Das Vorhandensein eines Caches ist somit weder eine hinreichende noch eine notwendige Bedingung für die Kommunikation zwischen zwei Endpunkten.

Im Gegensatz zum Internet sind die Vernetzungslösungen gerade für stark ressourcenlimitierte Umgebungen meist nicht nach allgemeingültigen Architekturprinzipien entworfen, sondern wurden speziell für eine dedizierte Anwendung entwickelt und optimiert. Daher ist bei diesen Vernetzungslösungen eine direkte Ende-zu-Ende Kommunikation nicht möglich. Als Vermittler müssen aus diesem Grund komplexere Systeme als einfache Caches eingesetzt werden. In solchen, teils proprietären, Infrastrukturen realisieren Gateways den Übergang der optimierten und

anwendungsspezifischen Protokolle in Netzwerktechnologien, wie sie beispielsweise im Internet genutzt werden. Da sich die Vernetzungslösungen auf den beiden Seiten der Gateways gänzlich unterscheiden, bedarf es bei Änderungen der Anwendungslogik meist auch Änderungen des Gateways. Gateways sind somit implementierungsspezifische Stellvertreter und eine notwendige Vorrausetzung für die Kommunikation. Durch die Notwendigkeit der Gateways für das Zustandekommen einer Kommunikation sowie die erforderliche Umsetzung der Protokolle und Daten bis hoch zur Anwendungslogik werden solche Gateway-Infrastrukturen meist als unflexibel und schwergewichtig beschrieben (siehe [8]).

Neben der Inflexibilität von Gateways auf Protokollebene hat sich die Zustandsbehaftung als Problem hinsichtlich der Skalierbarkeit der Systeme herausgestellt [8, 39, 45]. Zustände können auf verschiedenen Protokollebenen und in verschiedenen zeitlichen Zusammenhängen existieren. Beispielsweise bedeutet eine aktive TCP-Verbindung auf Transportschicht, dass die Verbindung verwaltet werden muss. Dies ist dann besonders kritisch, wenn keine Daten über die aktive Verbindung übertragen werden. Die Denial of Service-Attacke SYN-Flood nutzt diesen Fakt gezielt aus. Dieser Zustand ist aber auf die Dauer der TCP-Verbindung beschränkt. Auf den darüber liegenden Schichten hingegen können, unabhängig von der Transportschicht, dauerhafte Zustände existieren, wodurch sich aufeinanderfolgende Aufrufe bedingen. In diesem Fall besteht bei der Nutzung von Gateways ein Nachteil der Zustandsbehaftung darin, dass für aufeinander folgende Aufrufe der gleiche Vermittler genutzt oder jeweils der Zustand und Kontext wiederhergestellt werden muss.

Ist ein Vermittler notwendige Voraussetzung zur Kommunikation und muss dieser zugleich viele Zustände verwalten, so skaliert diese Architektur mit zunehmender Anzahl von Endpunkten nicht. Im schlechtesten Fall muss die Zwischenstation für jede Kombination aus Endpunkten einen Zustand verwalten. Dadurch würden die benötigten Ressourcen exponentiell mit der Anzahl der Verbindungen steigen.

Eine leichtgewichtigere Variante zu Gateways sind Proxys, da diese zum einen zustandslos implementiert werden können und zum anderen lediglich die Protokolle, aber nicht die Anwendungslogik ineinander umsetzen.

In der Literatur finden sich für die Begriffe *zustandslos*, *zustandsbehaftet*, *Proxy* und *Gateway* verschiedene Definitionen, da diese von dem konkreten Bezug und der Betrachtungsweise

abhängen (siehe z.B. [81]). Im Kontext dieser Arbeit werden die wichtigsten Begriffe wie nachfolgend aufgeführt definiert.

Aufruf. Als Aufruf wird die Interaktion mit Daten und/oder das Nutzen von Methoden, Funktionen und Diensten von zwei vernetzten Endpunkten bezeichnet. Ein Aufruf besteht aus einer Anfrage und kann zusätzlich aus einer Antwort bestehen. Der Aufruf schließt den Auf- und Abbau einer Transportverbindung bzw. eines Transportkanals zwischen den Endpunkten ein, nicht aber das Suchen und Finden der Entitäten in der Netzwerkumgebung.

Zustandslos. Ein Aufruf bzw. die Umsetzung von Daten und Protokollen mittels Vermittlern ist dann zustandslos, wenn einzelne Aufrufe voneinander unabhängig sind und einander nicht bedingen. Ein einzelner Aufruf kann die Verwaltung eines Zustandes erfordern. Dies ist auf die Dauer des Aufrufs beschränkt.

Zustandsbehaftet. Ein Aufruf bzw. die Umsetzung von Daten und Protokollen mittels Vermittlern ist dann zustandsbehaftet, wenn getrennte Aufrufe voneinander abhängig sind und sich gegenseitig bedingen. Dies kann die Notwendigkeit der Einhaltung einer spezifischen Reihenfolge der Aufrufe beinhalten. Die Dienste, Aufrufe bzw. Nachrichten sind somit durch sich selbst und dem vorherrschenden Kontext definiert.

Gateway. Ein Gateway ist ein Vermittler, der Daten und Protokolle zwischen Endpunkten umsetzt und dafür einen Zustand verwalten muss. Die Zustandsverwaltung kann beinhalten, dass eine Pufferung der Daten beim Gateway erfolgt und für Aufrufe keine Ende-zu-Ende Verbindung zwischen den Endpunkten zustande kommt. Weiterhin muss ein Gateway die Semantik der Anwendung interpretieren können, um entsprechend darauf zu reagieren und die korrekten Umsetzungen durchführen zu können.

Proxy. Ein Proxy ist ein Vermittler, der Daten und Protokolle zwischen Endpunkten umsetzt und dafür keinen Zustand verwalten muss. Ein Proxy muss nicht zwingend die Semantik der durch ihn vermittelten Aufrufe und Daten kennen, um die Umsetzungen korrekt durchzuführen. Ein Proxy kann das Zwischenpuffern (vgl. Cache) von Daten beinhalten, um die Infrastruktur zu entlasten.

Die Umsetzung der Protokolle und Daten kann bei einigen Technologien *transparent* erfolgen. Der Begriff *transparent* wird im Kontext dieser Arbeit wir folgt definiert.

> **Transparent.** Eine transparente Umsetzung zwischen Protokollen und Daten bedeutet, dass die Umsetzung vor den Kommunikationspartnern verborgen werden kann.

Es gilt hervorzuheben, dass die Komplexität der Umsetzung kein Bestandteil der Definitionen ist. Es kann angenommen werden, dass der Vermittler immer über genügend Ressourcen verfügt, um die Umsetzungen prinzipiell durchführen zu können. Weiterhin ist hervorzuheben, dass ein einzelner Aufruf in sich zustandsbehaftet sein kann. Dieser Zustand kann unter Umständen sehr komplex sein. Dies ist aber nicht als kritisch zu bewerten, da der Zustand nur für die Dauer des Aufrufs verwaltet werden muss und sich bei jedem Aufruf durch den Aufruf selbst erzeugen lässt.

4.1 Entwicklung einer skalierbaren Protokoll- und Netzwerkarchitektur

Während bei Gateway-Infrastrukturen die Interoperabilität abhängig von der konkreten Implementierung der Anwendung ist, basieren skalierbarere Lösungen auf der direkten Interaktion von Geräten und Systemen, bei der die genutzten Protokolle selbst kompatibel sind. In diesem Unterkapitel wird daher eine grundlegende Architektur konzipiert, bei der Gateway-Infrastrukturen vermieden werden können.

Gemeinsame Grundlage der entwickelten Lösung bildet die durchgängige Adressierungsschicht, die durch IPv6 umgesetzt wird. IPv6 schafft die Voraussetzung für eine direkte Ende-zu-Ende Verbindung, bei der z.B. vom Zugriffsverfahren, dem Übertragungsmedium und der Topologie abstrahiert werden kann. Mit der Entwicklung der 6LoWPAN-Protokolle können IPv6, UDP und TCP auch in stark ressourcenlimitierten Umgebungen eingesetzt werden. Dadurch verfügt die gesamte Umgebung über eine horizontale, homogene Kommunikationsschicht, die unabhängig von den jeweiligen Ressourcenbeschränkungen und dem genutzten Medium (drahtgebunden oder drahtlos) ist.

Die 6LoWPAN-Protokolle, IPv6, UDP und TCP sind aber lediglich in der Lage, Endpunkte zu adressieren und Datenpakete auszutauschen. In M2M-Anwendungen müssen darüber hinaus weitere Schnittstellenfunktionalitäten bereitgestellt werden. Diese beinhalten unter anderem das Suchen und Finden von Endpunkten (siehe Discovery), asynchrone Benachrichtigungen über

Zustandsänderungen (siehe Eventing) und Sicherheitsmechanismen. Da nicht jede Anwendung alle diese Funktionalitäten benötigt, muss die entstehende Lösung modular aufgebaut sein und flexibel um notwendige Protokollkomponenten erweitert werden können. Eine domänenübergreifende Gesamtlösung, die dennoch für einzelne Szenarien angepasst werden kann, ist somit essentielle Voraussetzung für den Einsatz in einer heterogenen Anwendungsumgebung.

Die im Rahmen dieser Arbeit vorgeschlagene Lösung betrachtet vornehmlich die Anwendungsschicht. Schichtenübergreifende Ansätze, im Sinne des ISO/OSI-Referenzmodells [30], können Lösungen für einzelne Anwendungen optimieren. Dies schränkt aber die Flexibilität und Wiederverwendbarkeit stark ein. Weiterhin führen schichtenübergreifende Lösungen meist zur Nutzung von Gateways, da Mechanismen und Protokolle auf verschiedenen Schichten nicht mehr nahtlos ineinander überführt werden können.

Nicht alle Anwendungsschicht-Protokolle eignen sich für M2M-Szenarien. So ist natives HTTP für H2M-Szenrien optimiert und erfährt den breitesten Einsatz beim Abrufen von Internetseiten. Für andere Anwendungen muss HTTP erweitert werden. CoAP ist stark an HTTP angelehnt, verfügt aber zusätzlich über erweiterte Mechanismen wie beispielsweise Discovery und Eventing. Allerdings sind sowohl CoAP als auch HTTP in ihrem Funktionsumfang und hinsichtlich der Erweiterbarkeit eingeschränkt. Daher definieren ganzheitliche Lösungen ein modulares und flexibel erweiterbares Protokoll-Framework. SOAP-basierte Web Services stellen eine derartige Lösung dar. Nur solche Ansätze sind in der Lage, komplexe Normen und Standards umzusetzen. Health Level 7 (HL7) ist ein Beispiel für eine solche komplexe Norm und dient zum Austausch von Daten im Gesundheitswesen [82, 83].

Im Bereich der SOAP Web Services hat sich das DPWS-Protokoll für den Einsatz in gerätenahen Anwendungen durchgesetzt. Durch DPWS können Geräte nahtlos in höherwertige Diensteumgebungen integriert werden. Der Vorteil von DPWS besteht in der Konformität mit der Web Services Architecture (WSA, WS-Architecture) [25] und somit der Erweiterbarkeit um Lösungen aus dem W3C SOAP Web Services Protokoll-Framework sowie eigenen Ergänzungen gleichermaßen.

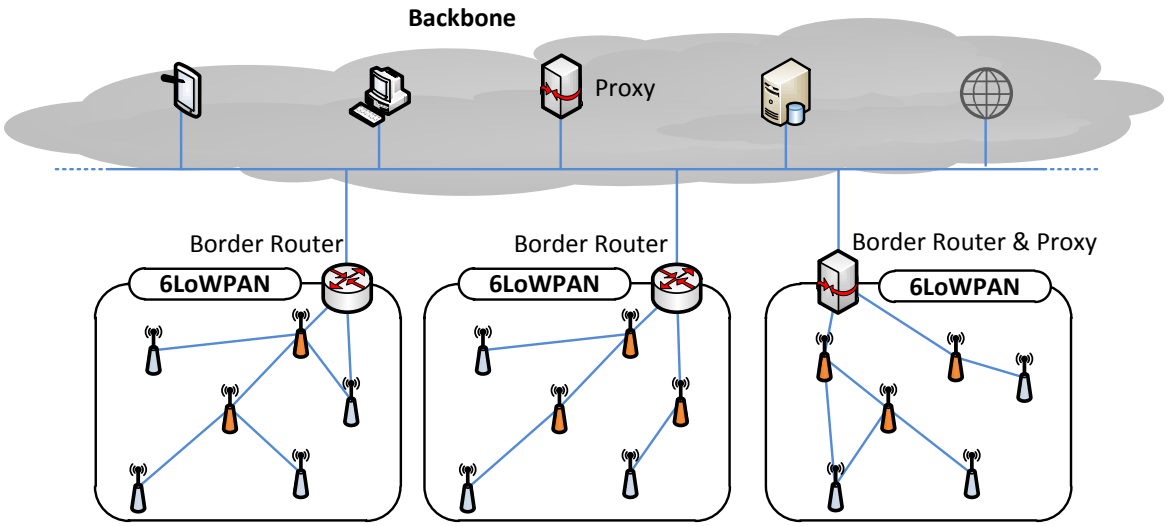

Abbildung 4.1 Netzwerkarchitektur

In früheren Forschungsarbeiten wurde bislang nur rudimentär untersucht, ob SOAP Web Services im Allgemeinen und konkret das Geräteprofil DPWS auch in 6LoWPANs angewendet werden können. Allgemeiner formuliert gilt es zu untersuchen, wie weit DPWS nach unten skaliert und wo die untere Grenze bezüglich der Ressourcenanforderungen liegt. Die existierenden Ansätze zur Untersuchung dieser Fragestellung stützen sich im Kern auf grundlegende Änderungen von DPWS oder SOAP. Beispiele dafür sind in [84–89] beschrieben. Die dort vorgestellten Lösungen sind zwar bezüglich der Ressourcenanforderungen optimiert, sind aber nicht mehr mit der WS-Architecture und dem WS-* Framework kompatibel. Dadurch können diese Ansätze nicht ohne weiteres in eine Dienstkomposition eingebunden werden. Sie benötigen Netzwerkinfrastrukturen, bei denen die Leichtgewichtigkeit innerhalb der limitierten Umgebung durch eine Verschiebung der Komplexität auf schwergewichtige, anwendungsspezifische Gateways als Stellvertreter erreicht wird. Der Vorteil der Nutzung von IPv6 bzw. 6LoWPAN ist somit durch die fehlende Ende-zu-Ende Konnektivität marginal.

Eine ganzheitliche Lösung, die

(1) den Ressourcenanforderungen,
(2) der Konformität zur WS-Architecture,
(3) als auch der Skalierbarkeit von Netzwerkinfrastrukturen genügt,

ist nicht vorzufinden und wird in dieser Arbeit erstmals beschrieben.

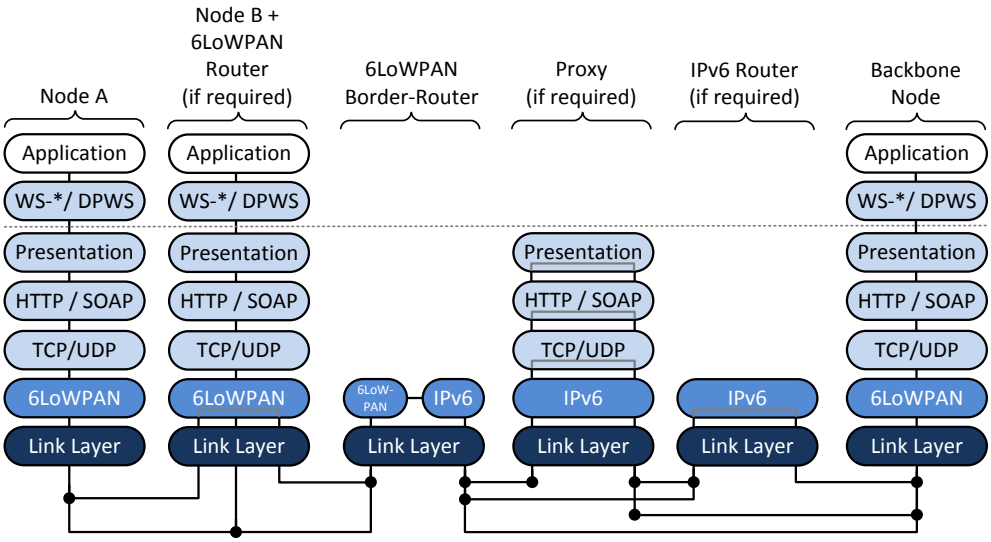

Abbildung 4.2 Funktionsweise der Netzwerkelemente

Die in dieser Arbeit vorgestellten Ansätze erhalten die grundlegende Konformität von DPWS zum WS-* Framework und bedienen sich des generellen Aufbaus der WSA und von SOAP. Betrachtet wird somit nicht nur DPWS selbst, sondern auch gezielte Aspekte der zugrundeliegenden WSA. Durch die Konformität zu existierenden Standards müssen bestehende Implementierungen von DPWS oder anderen WSA-basierten Systemen nicht geändert werden und es kann direkt auf die Daten und Funktionen der Sensoren bzw. Aktoren zugegriffen werden, wodurch sich für die Anwendungen potenziell vollständig neue Datenquellen erschließen.

Die Netzwerkinfrastruktur für den Einsatz von DPWS in stark ressourcenlimitierten Umgebungen ist in Abbildung 4.1 dargestellt. Die Netzwerkumgebung besteht aus dem Backbone und einem oder mehreren 6LoWPAN-Netzwerken. Die Ansätze dieser Arbeit beschränken sich aber nicht auf die 6LoWPAN-Protokolle, sondern können ebenfalls in anderen stark ressourcenlimitierten Umgebungen eingesetzt werden. Der Begriff Backbone steht stellvertretend für ein ressourcenstärkeres Netzwerk, das dem aktuellen Stand der Technik entspricht und welches keiner weiteren Anpassung bedarf. Dies schließt das Internet selbst mit ein.

An der Schnittstelle zwischen 6LoWPANs und dem Backbone befinden sich Border Router, die zwischen 6LoWPAN-Protokollen und nativem IPv6 vermitteln. Auf der Anwendungsschicht können zustandslose Proxys als Vermittler zwischen den existierenden und neu entwickelten Protokollen eingesetzt werden. Die Proxys sind in ihrer Positionierung innerhalb der Infrastruktur nicht wie Gateways an den Übergang der Netzwerke (vgl. Border Router) gebunden. Die

Funktionsweisen der einzelnen Netzwerkelemente bezüglich der Protokollschichten sind gesondert in Abbildung 4.2 dargestellt.

4.2 Anpassungen von DPWS für ressourcenlimitierte Umgebungen

Um DPWS in 6LoWPANs und somit auf stark ressourcenlimitierten Zielplattformen einsetzen zu können, sind drei voneinander getrennte Kernaspekte zu betrachten (siehe Abbildung 4.3). Es wird unterschieden zwischen (1) Mechanismen und Funktionen der Schnittstellen, (2) der Darstellungsform und Datentypen sowie (3) dem Transports- bzw. Austauschprotokoll. Der Aspekt (1) bezieht sich auf konforme Erweiterungen für DPWS und einzelne WS-* Spezifikationen. Die Aspekte (2) und (3) hingegen beziehen sich direkt auf die Nachricht als atomares Element der WSA. Daher begünstigen die Aspekte (2) und (3) wiederum den Einsatz von zustandslosen Proxys, da die Umsetzung des Nachrichtenformates und des Transportprotokolls in andere Technologien kein Wissen über die Semantik der Anwendung, der Softwaretechnik o.ä. erfordert.

(1) *Mechanismen und Funktionen:* Die Mechanismen und Funktionen sind abstrakte und allgemeingültige Beschreibungen der Verhaltensweise der Schnittstelle und bilden daher die Logik. Beispiele hierfür sind Discovery und Eventing, für die in Kapitel 5 entsprechende Erweiterungen konzipiert werden.

(2) *Darstellungsform und Datentypen:* Die Darstellungsformen und Datentypen dienen zur Serialisierung und Kodierung der Mechanismen und Funktionen, damit

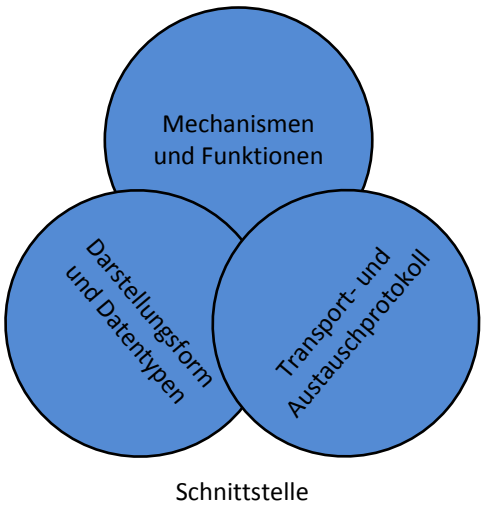

Abbildung 4.3 Teilbereiche Schnittstellen

diese in paketorientierten Netzwerken in Form von Nachrichten übertragen werden können. Durch die Nutzung von plattform- und programmiersprachenunabhängigen Formaten (z.B. XML) wird ein höherer Grad der Abstraktion von konkreten Implementierungen und Plattformen erzielt. In Kapitel 6 wird eine ressourceneffiziente Darstellungsform entwickelt, deren Grundlage das EXI-Format (Efficient XML Interchange) ist. Das neue Datenformat EXI ist kompatibel mit XML-Infoset und stellt somit eine konforme Ergänzung zur WSA, SOAP und den DPWS-Spezifikationen dar. Dennoch würde die Kodierung in dem neuen Format eine Änderung der bestehenden Implementierungen erfordern. Um dies zu vermeiden, braucht es zustandslose, transparente Umsetzungen von nativem XML in EXI und umgekehrt. Dedizierte Proxys, die zwischen zwei Endpunkten vermitteln, setzen die Kodierungsformen ineinander um. In der existierenden IT-Infrastruktur kann somit weiterhin natives Unicode XML genutzt werden.

(3) *Transports- bzw. Austauschprotokoll:* Die Nachrichten werden zwischen den Endpunkten durch geeignete Transport- und Austauschprotokolle übermittelt. Diese Transportmechanismen haben keinen Einfluss auf die Schnittstellenlogik der Dienste und können, wie auch das Datenformat, ausgetauscht werden. Ausgehend von dem existierenden Binding SOAP-over-HTTP wird in Kapitel 7 ein neuartiges SOAP-Binding, basierend auf dem neuen Protokoll CoAP, konzipiert und spezifiziert. CoAP lehnt sich stark an HTTP und verfügt über einen ähnlichen grundlegenden Aufbau. Dadurch ist es möglich, CoAP direkt und zustandslos in HTTP umzusetzen. Da CoAP und auch HTTP nicht an eine spezielle Serialisierung bzw. Kodierung der Nutzdaten gebunden sind, können beide Protokolle für den Transport von SOAP-Dokumenten genutzt werden.

Die drei genannten Bereiche sind zu großen Teilen voneinander entkoppelt und können getrennt voneinander betrachtet werden. Einige Aspekte überlagern sich dennoch und können sich gegenseitig ergänzen. Vor allem hinsichtlich der Effizienz der Gesamtlösung muss die Realisierung der abstrakten Mechanismen einer Schnittstelle durch konkrete Protokolle immer mitbetrachtet werden. Beispielsweise benötigt das Nutzen einer Schnittstelle beim Client die eindeutige Zuordnung einer eingehenden Antwort zu der zuvor ausgesendeten Anfrage. Dies ist ein notwendiger und allgemeingültiger Mechanismus. Wird als Austauschprotokoll ein zustandsloser

Transportmechanismus wie UDP verwendet, müssen die Mechanismen durch die Schnittstellenlogik und deren Mechanismen selbst realisiert werden. Wird hingegen ein verbindungsorientierter Transportmechanismus genutzt, so wird der allgemeingültige Mechanismus durch das Austauschprotokoll realisiert. In diesem Fall bedarf es keiner weiteren Umsetzung durch die Funktionslogik der Schnittstelle.

4.3 Angepasste DPWS-Architektur

Mittels der genutzten Protokolle lässt sich der nun konzipierte Protokollstapel wie in Abbildung 4.4 darstellen. Als Binding (vgl. Transportmechanismus) für SOAP kann auf drei Protokolle zurückgegriffen werden. In Abbildung 4.4 sind die Bindings SOAP-over-HTTP, SOAP-over-UDP und SOAP-over-CoAP ebenfalls dargestellt. Das SOAP-over-CoAP Binding, ein neuartiges Binding, wurde im Rahmen dieser Arbeit entwickelt und spezifiziert.

Zur Serialisierung und Kodierung der SOAP-Dokumente kommen XML-Infoset und damit konforme Formate wie natives Unicode XML, EXI oder - wie in Abbildung 4.4 ebenfalls beispielhaft dargestellt - Fast Infoset zum Einsatz.

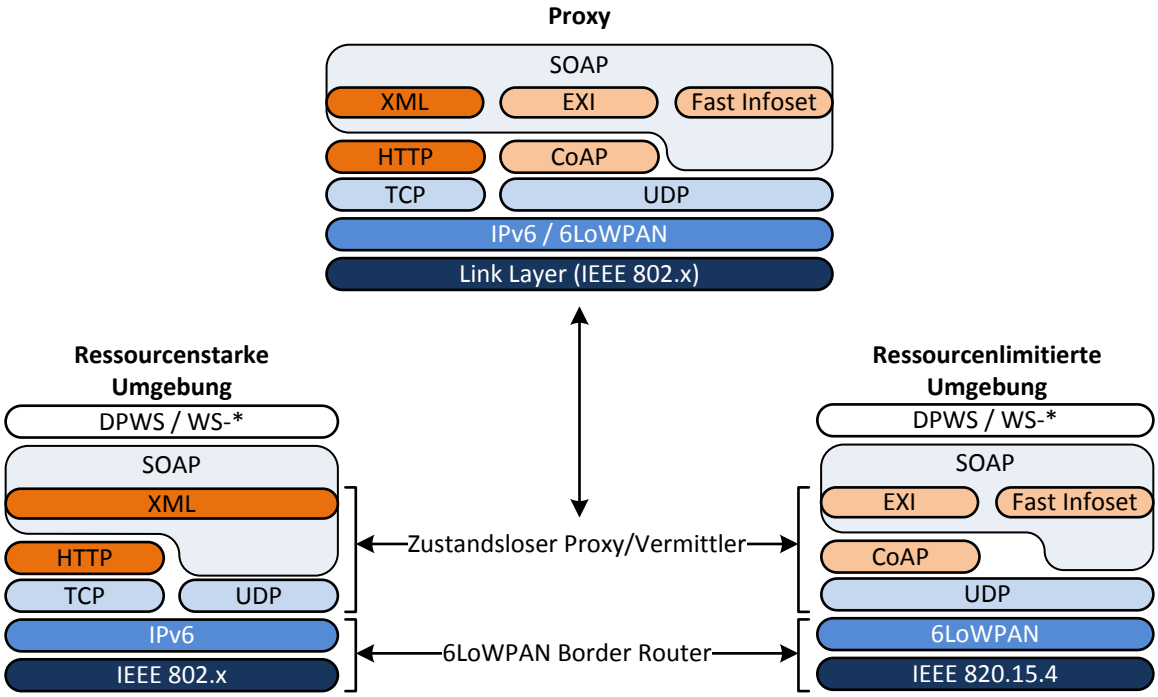

Abbildung 4.4 Angepasster DPWS-Protokollstapel

Binding	**HTTP**	**HTTP**	**CoAP**	**CoAP**	**UDP**	**UDP**
Kodierung	**XML**[1]	**EXI**	**XML**[1]	**EXI**	**XML**[1]	**EXI**
Zuverlässiger Transport	ja	ja	optional	optional	nein	nein
Flusskontrolle Stauvermeidung	ja	ja	nein	nein	nein	nein
Unicast	ja	ja	ja	ja	ja	ja
Multicast	nein	nein	ja	ja	ja	ja
Zustandslos Umsetzbar in Binding	CoAP	CoAP	HTTP	HTTP	-	-
Zustandslos Umsetzbar in Kodierung	EXI	XML[1]	EXI	XML[1]	EXI	XML[1]

[1]Unicode kodiertes XML

Tabelle 4.1 Eigenschaften und Protokollumsetzungen bei angepasstem DPWS-Protokollstapel

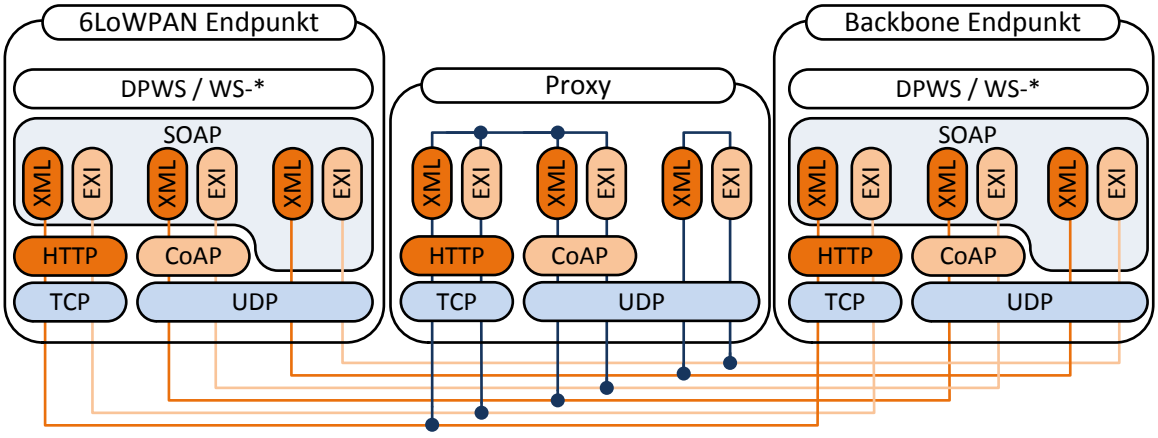

Abbildung 4.5 Kommunikationsszenarien bei angepasstem DPWS-Protokollstapel

EXI wurde im Rahmen dieser Arbeit erstmals im Kontext von SOAP-basierten Web Services in 6LoWPANs evaluiert und aus dessen komplexem Funktionsumfang die optimale Konfiguration extrahiert. Dies schließt die weltweit erste frei zugängliche Implementierung von EXI für Zielplattformen von 6LoWPANs mit ein.

DPWS selbst, die WS-* Protokolle und deren grundlegende Kriterien werden nur ergänzt und nicht grundlegend verändert, um die Konformität zu anderen Spezifikationen und Erweiterungen zu erhalten.

Die jeweiligen Anpassungen können frei miteinander kombiniert werden. Durch die Nutzung von Proxys können die einzelnen Anpassungen zustandslos in ihre nativen Pendants umgesetzt werden. Die Anbindung von 6LoWPANs erfordert somit keine Änderungen der bestehenden Systeme.

Die Tabelle 4.1 fasst mögliche Protokollumsetzungen und Eigenschaften der Protokolle zusammen. In Abbildung 4.5 sind mögliche Umsetzungen der Transportmechanismen und der Kodierungsformen für SOAP-basierte Web Services durch einen zustandslosen und anwendungsunabhängigen Proxy grafisch dargestellt.

4.4 Referenzszenario und Referenzplattform

Neben der Vermeidung von Gateways konnte im Rahmen dieser Arbeit eine Lösung hervorgebracht werden, die ausschließlich auf existierenden, domänenübergreifenden, offenen Standards basiert. Die Herausforderung dabei besteht darin, dass die genutzten Protokolle entweder nicht für den Einsatz in stark ressourcenlimitierten Umgebungen bzw. nicht im Kontext von SOAP-basierten Web Services entworfen wurden. Neben den sehr umfassenden Untersuchungen zur Konzeption einer geeigneten Lösung ist somit die neuartige Kombination der entwickelten Komponenten hervorzugheben. Integraler Bestandteil der vorgestellten Lösungen sind ebenfalls Implementierungen, welche die Umsetzbarkeit demonstrieren und weiterhin Bewertungen hinsichtlich der Leistungsfähigkeit der Lösung ermöglichen.

4.4.1 Referenzszenario

Beispielhaft wird durch die Implementierungen ein spezifisches Referenzszenario umgesetzt. Das Szenario bildet die Realisierung einer Temperaturregelung (engl. Air Conditioner). Die Temperaturregelung enthält Schnittstellen, die deren Überwachung und Steuerung ermöglichen.

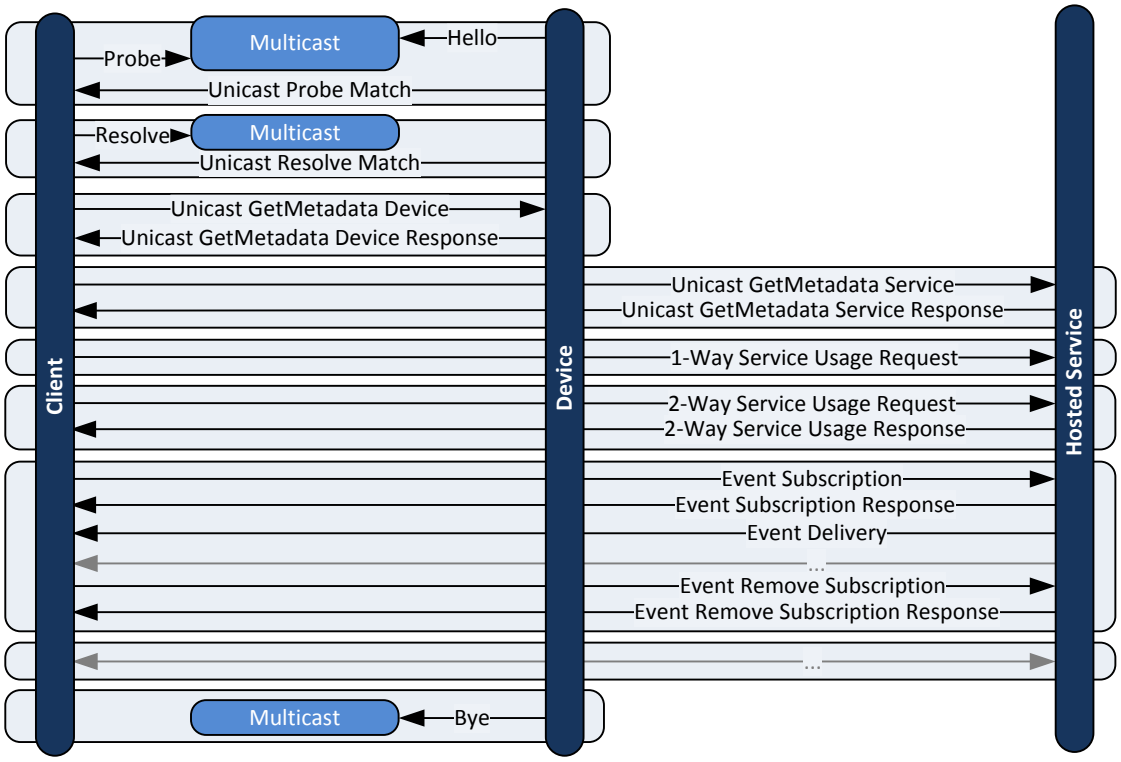

Abbildung 4.6 Mögliche Nachrichtenabfolge DPWS

Beispielsweise realisiert eine Schnittstelle das Abrufen der aktuell gemessenen Temperatur. Eine weitere Schnittstelle ermöglicht das Setzen einer Temperatur, die von der Temperaturregelung aufrechterhalten werden soll. Diese einfachen Schnittstellen weisen ein sehr schlechtes Verhältnis von Nutzdaten zu Protokolldaten auf, da die Daten sich zum Teil auf einzelne Zahlenwerte beschränken. Darüber hinaus existieren komplexere Schnittstellen, welche die zuvor genannten Funktionen zusammenfassen und um Statusinformationen ergänzen. Um zyklische Abfragen zu vermeiden, unterstützt die Temperaturregelung ebenfalls asynchrone Benachrichtigungen über Statusänderungen (vgl. Eventing). Gemeinsam mit den konformen Mechanismen zum Suchen und Finden von Geräten im Netzwerk (vgl. Discovery) setzt das gewählte Referenzszenario alle relevanten DPWS Protokollfunktionen und -mechanismen um.

Eine Temperaturregelanlage als Zielplattform muss nicht zwingend durch Ressourcenbeschränkungen charakterisiert sein. Stark ressourcenlimitierte Zielplattformen sind in den angebotenen Diensten und Daten unter Umständen deutlich eingeschränkter. Das Referenzszenario wurde dennoch aus zwei Gründen gewählt. Zum einen werden alle relevanten Mechanismen, die in DPWS beschrieben werden, genutzt und lassen sich auf das Referenzszenario

anwenden. Daher stellt das Referenzszenario das Maximum bezüglich der Anforderungen und der Protokollkomplexität dar. Zum anderen wurden die im Rahmen dieser Arbeit entstandenen Implementierungen auf deren DPWS-Konformität getestet. Grundlage für diese Tests waren Implementierungen der Web Services for Devices (WS4D) [90] Initiative, welche die Interoperabilitätsszenarien [91] im Rahmen der Standardisierung von DPWS bei der OASIS durchlaufen haben. Die Temperaturregelanlage ist das Referenzszenario der genutzten DPWS-Implementierungen von WS4D und kann somit mit allen für die Tests genutzten Implementierungen umgesetzt werden.

In dem umgesetzten Referenzszenario können 18 typische Nachrichten identifiziert werden, die im Rahmen dieser Arbeit wie nachfolgend aufgeführt bezeichnet werden. Eine mögliche Abfolge der Nachrichten ist in Abbildung 4.6 grafisch dargestellt.

Hello und Bye. *Hello*- und *Bye*-Nachrichten werden von einem Gerät gesendet, wenn dieses dem Netzwerk beitritt bzw. das Netzwerk verlässt. Die beiden Nachrichten werden über UDP an eine definierte Multicast-Gruppe gesendet.

Probe. Mit einer *Probe*-Nachricht können Clients nach Geräten in einem Netzwerk suchen. *Probe*-Nachrichten werden, ebenfalls über UDP, an die gleiche Multicast-Gruppe wie die *Hello*- und *Bye*-Nachrichten gesendet.

Probe Match. Ein *Probe Match* wird vom Gerät per Unicast als Antwort auf eine *Probe*-Nachricht gesendet.

Resolve. Ein *Resolve* wird von Clients genutzt, um für die eineindeutigen Geräteadressen entsprechend der WS-Addressing Spezifikation aus dem *Hello* bzw. *Probe Match* eine gültige Transportadresse zu ermitteln. Auch das Resolve wird mittels Multicast via UDP übertragen.

Resolve Match. Das *Resolve Match* ist die Unicast-Antwort auf eine *Resolve*-Anfrage und enthält eine gültige Transportadresse.

GetMetadata Device Request und Response. Mit diesen Nachrichten werden die als WS-Transfer Ressource modellierten Metadaten des Gerätes übermittelt.

GetMetadata Service Request und Response. Mit diesen Nachrichten werden die Metadaten einzelner Dienste übertragen. Diese Metadaten sind identisch zur WSDL.

1-Way Request. Diese Nachricht enthält eine Anfrage an einen spezifischen Dienst. Auf die Anfrage wird von dem Dienst keine Antwort gesendet.

2-Way Request und Response. Bei diesen Nachrichten wird an einen spezifischen Dienst eine Anfrage gesendet, auf die von diesem eine Antwort folgt.

Event Subscription und Response. Entsprechend der WS-Eventing-Spezifikation kann sich ein Client für asynchrone Benachrichtigungen mit einer *Event-Subscription* registrieren, deren Verarbeitung von der Event-Quelle entsprechend bestätigt wird.

Event Delivery. Eine *Event-Delivery*-Nachricht wird für die Zustellung eines Events genutzt.

Event Remove Subscription und Response. Ist ein Client an den asynchronen Benachrichtigungen nicht mehr interessiert, wird die Registrierung bei der Event-Quelle aufgehoben. Dieser Vorgang muss von der Event-Quelle ebenfalls bestätigt werden.

Die entstandenen Implementierungen des Referenzszenarios mit den aufgezeigten 18 Nachrichten stellen eine mögliche Umsetzung dar. Andere Implementierungen des gleichen Referenzszenarios können unterschiedlich ausfallen. Dennoch ist das Referenzszenario realistisch und ermöglicht eine Bewertung der entstandenen Lösung.

4.4.2 Geräteklassifizierung

Um ein besseres Verständnis für die Begrifflichkeit *stark ressourcenlimitierter Kleinstgeräte* zu ermöglichen, kann eine Diskussion aus der IETF Arbeitsgruppe Light-Weight Implementation Guidance (LWIG) herangezogen werden (siehe [92]). In der Diskussion der Arbeitsgruppe wurden mögliche Gerätekategorien identifiziert. Basierend auf der LWIG-Klassifizierung wird im Rahmen dieser Arbeit die in Tabelle 4.2 aufgezeigte Einteilung vorgenommen. Die Angaben in der Tabelle sind jeweils im Kontext der Anwendung in 6LoWPAN-basierten Netzwerken zu verstehen, denn nicht alle ressourcenoptimierten Plattformen und Betriebssysteme unterstützen die 6LoWPAN-Protokolle. Gründe für die unterschiedliche Leistungsfähigkeit der Geräteklassen

Klasse	ROM	RAM	Energie-verbrauch	Routing	typ. Program-miersprachen	typ. OS
0	<20kB	<4kB	-	-	Assembler	-
1	20-100kB	4-20kB	+	+	C, nesC	Contiki, TinyOS
2	100-250kB	20-50kB	++++	+++	C, Java	Contiki, TinyOS, FreeRTOS, TakaTuka
3	>250kB	>50kB	+++++ +++++ +++++	+++++ +++++ +++++	C, C++, Java	FreeRTOS, Linux, Windows CE

Tabelle 4.2 Plattformklassifizierung im Kontext der Anwendung in 6LoWPANs

können nichttechnische Anforderungen wie Bauform und Kosten oder technische Einschränkungen wie Kommunikationsbandbreite und Batteriekapazität sein. Grundsätzlich ist zu beobachten, dass die Plattformen in den aufsteigenden Klassen jeweils durch einen höheren Energieverbrauch gekennzeichnet sind.

Klasse 0. Die Plattformen der Klasse 0 sind die am stärksten ressourcenlimitierten Plattformen, die in vernetzten Umgebungen eingesetzt werden können. Diese Plattformen verfügen nach dem aktuellen Stand der Technik nicht über genügend Speicher, um darauf Internetprotokolle einsetzen zu können und sind somit zu stark ressourcenlimitiert. Sie verfügen nur über wenige kB RAM und ROM Speicher.

Klasse 1. Die nächstgrößere Klasse von Zielplattformen verfügt über bis zu etwa 100 kB ROM und bis zu etwa 20 kB RAM. Diese Plattformen sind die kleinsten bekannten Plattformen, auf denen Internettechnologien realisiert werden können. Dennoch benötigen diese Plattformen angepasste Lösungen bezüglich der Protokolle und der Implementierungen. Weiterhin sind diese Plattformen durch den begrenzten dynamischen Speicher dahingehend eingeschränkt, als dass nur wenige parallele Verbindungen zugleich bedient werden können. In einem vermaschten drahtlosen Netzwerk beinhaltet dies unter anderem die Anzahl der möglichen benachbarten Knoten, die für ein effizientes Routing innerhalb der vermaschten Topologie bekannt sein müssen.

Klasse 2. Auch die Plattformen der Klasse 2 benötigen optimierte Lösungen bezüglich der Protokolle und der Implementierungen. Allerdings verfügen diese Plattformen

bereits über genügend Speicher und Ressourcen, um auch mehrere und komplexere Anfragen zugleich bearbeiten zu können bzw. eine hohe Anzahl von Endpunkten in den Routingtabellen zu verwalten. Diese Zielplattformen können somit beispielsweise als Datenaggregator eingesetzt werden. Der statische ROM-Speicher dieser Plattform beträgt bis zu etwa 250 kB und der dynamische RAM-Speicher bis zu etwa 50 kB.

Klasse 3. Alle unter der Klasse 3 zusammengefassten Zielplattformen sind kaum noch als stark ressourcenlimitiert einzuschätzen und verfügen teilweise über mehrere MB ROM und RAM. Auf diesen Plattformen können existierende Internetprotokolle und Implementierungen ohne weitere Optimierungen eingesetzt werden.

Diese Arbeit ist vornehmlich in der Geräteklasse 1 und zu Teilen in der Geräteklasse 2 angesiedelt. Für den Einsatz von Internettechnologien in diesen Klassen müssen, durch die Ressourcenlimitierungen, sowohl für die Protokolle als auch für die Implementierungen grundlegend neuartige Ansätze entwickelt werden. Diese Arbeit befasst sich mit beiden Aspekten.

4.4.3 Referenzplattform

Als Zielplattformen für die Validierung der Ansätze dieser Arbeit wurden TelosB-Knoten genutzt. Der TelosB wurde von der University of California in Berkeley speziell für den Forschungseinsatz entwickelt [93] und wird derzeit von verschiedenen Herstellern gefertigt und vertrieben. Als CPU (Central Processing Unit) kommt beim TelosB ein MSP430 Mikrokontroller der Firma Texas Instruments (TI) zum Einsatz, der aufgrund seiner stromsparenden Eigenschaften oft für Untersuchungen bei drahtlosen Sensornetzwerken genutzt wird. Der Mikrocontroller verfügt über eine 16 Bit RISC Architektur und wird mit 8 MHz getaktet. Das verwendete Modell besitzt einen 48 kB großen Flash ROM und 10 kB RAM. Weiterhin ist der TelosB mit dem IEEE 802.15.4 konformen Funkmodul CC2420 von TI ausgestattet. Der TelosB kann somit der Klasse 1 zugeteilt werden.

Für Plattformen der Klasse 1 werden vorzugsweise Programmiersprachen wie beispielsweise C eingesetzt. Durch Projekte wie TakaTuka [94] wurden Ansätze entwickelt, durch die auch höherwertige Sprachen wie Java auf den stark ressourcenlimitierten Systemen eingesetzt werden können. Nach Ko et al. [95] benötigt eine einfache Laufzeitumgebung (engl. Java Virtual Machine; JVM) von TakaTuka für einen TelosB-Knoten rund 24 kB ROM-Speicher, was 50% des Gesamtspeichers entspricht. Eine triviale Anwendung, die Daten empfängt, benötigt nach Ko et al.

etwa 36 kB (75%) ROM-Speicher eines TelosB. Die Implementierung der 6LoWPAN-Protokolle auf einem TelosB ist nach Ko et al. aufgrund der Speicherlimitierungen nicht möglich. Aslam et al. identifizieren in [96] für eine CLDC[1]-konforme JVM (exklusive der 6LoWPAN-Protokolle) einen minimalen ROM-Bedarf von rund 34,5 kB. Höherwertigere Sprachen wie Java konnten sich somit aufgrund des vergleichsweise hohen Speicherbedarfs alleine für die Laufzeitumgebung im Bereich der stark ressourcenlimitierten Umgebungen nicht durchsetzen.

Dennoch sind in den vergangenen Jahren verschiedene Lösungen entstanden, die als Mikrokontroller-Betriebssysteme bezeichnet werden und auf effizienteren Programmiersprachen als Java basieren. Diese Betriebssysteme stellen eine der Grundlagen für eine immer größere Verbreitung von Kleinstgeräten dar, da die Programmierung auf einem höheren Abstraktionsniveau erfolgen kann. Beispiele für solche Lösungen sind TinyOS [97], Contiki [98] und FreeRTOS [99].

Für die Umsetzung in dieser Arbeit wird auf das existierende Mikrokontroller-Betriebssystem Contiki [98] des Swedish Institute of Computer Science (SICS) zurückgegriffen. Die verwendete Versionsbezeichnung von Contiki ist 2.5. Der TelosB und Contiki stehen lediglich beispielhaft für mögliche Zielplattformen. Die Entscheidung für Contiki wurde aus zwei Gründen getroffen. Zum einen verfügt Contiki bereits über Implementierungen der Protokolle, wie beispielsweise 6LoWPAN, RPL, UDP und TCP [68]. Diese Anforderung würde auch TinyOS [97] erfüllen. Zum anderen ist Contiki aber in der Programmiersprache C implementiert. TinyOS hingegen ist in der speziell entwickelten Programmiersprache nesC [100] implementiert. Da in C implementierte Lösungen unabhängiger und nicht auf ein Betriebssystem beschränkt sind, wurde Contiki bevorzugt. Der TelosB ist aufgrund seiner großen Verbreitung die native Standardplattform von Contiki und TinyOS. Die im Rahmen dieser Arbeit entstandenen Implementierungen beschränken sich aber nicht auf den TelosB.

4.5 Zwischenfazit

Existierende Architekturen zur Einbindung drahtloser Sensornetzwerke in IP-basierte Infrastrukturen basieren auf der Verschiebung der Komplexität der Anwendung auf ressourcenstarke Stellvertreter. Diese anwendungsspezifischen, zustandsbehafteten Protokollumsetzer werden als Gateways bezeichnet. Eine direkte Ende-zu-Ende Kommunikation

[1] Connected Limited Device Configuration - Kleinstmögliche Konfiguration einer Java Platform Micro Edition Laufzeitumgebung

zwischen den datenerzeugenden und den datenverarbeitenden Endpunkten ist nicht möglich. Eine anwendungsabhängige und implementierungsspezifische Kopplung einer Vielzahl *von Intranets der Dinge* ist die Folge. Solche Lösungen sind allgemein als schwergewichtig und unflexibel zu bewerten.

Ganzheitlichere Architekturen zur Realisierung eines *Internets der Dinge* erreichen einen deutlich höheren Grad der Skalierbarkeit, sind dabei gleichzeitig flexibler und leichter zu verwalten. Eine solche Architektur basiert unter anderem auf den grundlegenden Prinzipien, wie Ende-zu-Ende Konnektivität und dem Einsatz zustandsloser Proxys. In diesem Kapitel wurde eine entsprechende Architektur konzipiert. Im weiteren Verlauf dieser Arbeit wird diese Architektur durch die nachfolgend entwickelten und spezifizierten Protokolle umgesetzt sowie mittels vollständig funktionsfähiger Implementierungen validiert und bewertet.

5 Weiterentwicklung der Mechanismen und Funktionen von DPWS

Inhaltsübersicht

Die DPWS-Spezifikationen sowie alle damit verbundenen Protokolle des W3C Web Services Framework weisen eine hohe Flexibilität und Modularität auf. Die Protokollbausteine beschreiben meist abstrakte und generische Mechanismen. Diese Mechanismen sind in der Regel erweiterbar, um in einem möglichst breiten Spektrum von Anwendungen eingesetzt werden zu können.

Diese Flexibilität führt oft zu einer hohen Komplexität bei der Implementierung. Die Flexibilität kann aber auch ausgenutzt werden, um DPWS zu vereinfachen bzw. zur Erhöhung der Effizienz zu erweitern. Dies geschieht beispielsweise, indem Variabilität beseitigt und optionale Komponenten weggelassen bzw. als obligatorisch definiert werden. Im Rahmen dieser Vereinfachung können vornehmlich die Anzahl und die Komplexität der Nachrichten verringert werden. Dazu werden in den folgenden Unterkapiteln verschiedene Ansätze vorgestellt.

Im Kontext des nachfolgenden Kapitels ist zu beachten, dass die DPWS-Spezifikationen meist das Verhalten des Gerätes und der Dienste beschreiben, die spezifische Schnittstellen bereitstellen. Clients und deren konkretes Verhalten sind meist nicht spezifiziert. Die Clients müssen sich dem Verhalten der Geräte und der Dienste anpassen. Die Definitionen der umzusetzenden Funktionalitäten der Clients ergeben sich implizit aus den Geräte- und Dienstspezifikationen.

5.1 Effizientes Discovery

Die Anzahl und Größe der Nachrichten eines 6LoWPANs hat im Vergleich zu existierenden Netzwerken einen größeren Einfluss auf die Leistungsfähigkeit und den Energiebedarf des gesamten Netzwerkes, da die Knoten unter Umständen auch als Router fungieren. Folglich beeinträchtigt die aufkommende Kommunikation nicht nur die Endpunkte selbst, sondern zusätzlich andere Endpunkte, die als Zwischenstationen dienen.

In Kapitel 4.4 sind 18 typische Nachrichten eines DPWS-Gerätes aufgeführt. Zehn dieser Nachrichten beinhalten Mechanismen, die dem Discovery zuzuordnen sind. In diesem Unterkapitel werden daher verschiedene Erweiterungsansätze konzipiert, wie das Discovery für den Einsatz von DPWS in 6LoWPANs ressourceneffizienter gestaltet werden kann.

5.1.1 Vermeidung von Redundanzen beim Discovery

Beim Eintritt bzw. Verlassen eines Netzwerkes teilt ein Gerät dies mit den Nachrichten *Hello* bzw. *Bye* mit. Clients können nach Geräten ebenfalls aktiv suchen. Hierzu sendet ein Client eine *Probe* Nachricht, die vom Gerät mit einem *Probe Match* beantwortet wird. Für das Suchen können Filter genutzt werden, um zu vermeiden, dass alle Geräte eines Netzwerkes antworten (vgl. Response Storm). Die Nachrichten *Hello*, *Probe* und *Bye* werden mittels einer Punkt-zu-Multipunkt Übertragung (Multicast) gesendet. Das *Probe Match* wird mittels einer direkten Punkt-zu-Punkt Kommunikation (Unicast) übermittelt.

Da die WSA unabhängig von darunterliegenden Schichten ist, müssen in den Nachrichten *Hello* und *Probe Match* keine Transportadressen des Gerätes bzw. der Dienste (z.B. IP-Adresse und Port Nummer) enthalten sein. Diese können vom Client durch das Senden einer Multicast *Resolve*-Nachricht angefordert werden, die mit einem Unicast *Resolve Match* vom entsprechenden Gerät beantwortet wird. Die Multicast *Resolve*-Nachricht wird somit an alle Empfänger der Multicastgruppe gesendet und von diesen empfangen. Das gesuchte Gerät wird mittels einer eindeutigen ID angesprochen, die vom vorangegangenen Schritt durch ein *Hello* oder *Probe Match* bekannt ist. Als Resultat sind für das Ermitteln einer gültigen Transportadresse im ungünstigsten Fall bis zu drei Multicast-Nachrichten (*Hello*, *Probe*, *Resolve*) und zwei Unicast-Nachrichten (*Probe Match, Resolve Match*) nötig.

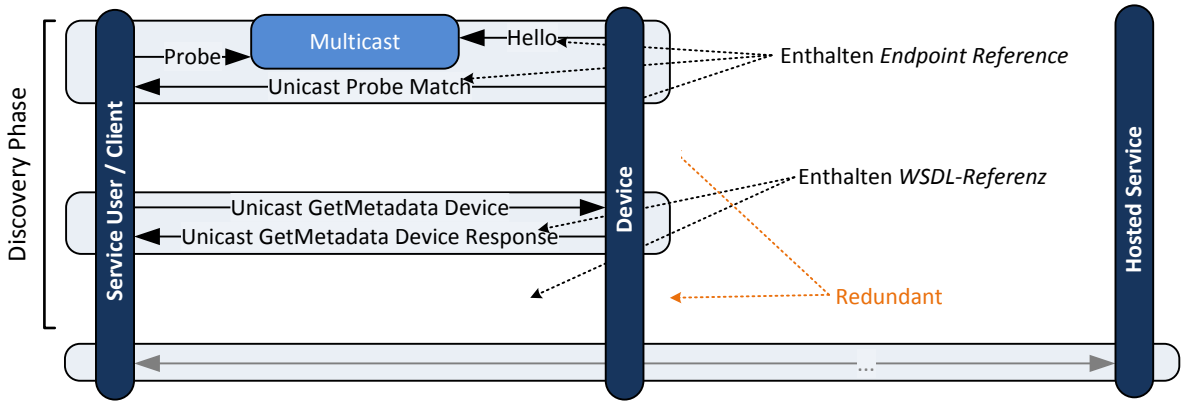

Abbildung 5.1 Angepasstes Discovery durch Vermeidung von Redundanz

Die erste offensichtliche Anpassung hinsichtlich des Discovery-Vorganges beabsichtigt somit, dass die beiden getrennten Schritte, nämlich das Finden des Gerätes und das Ermitteln der Transportadresse, funktional zusammengefasst werden. Hierdurch wird der *Resolve*-Mechanismus unnötig. Die Transportadresse wird entsprechend der DPWS und der WS-Addressing Spezifikationen in ein *Endpoint Reference*-Element in den Nachrichten *Hello* und *Probe* eingefügt. Zur Vermeidung der Namensauflösung von DNS-Hostnamen (Domain Name System) muss als Transportadresse die IP-Adresse und der Port direkt eingebettet werden. Die zusätzliche Angabe des DNS-Hostnamen ist optional möglich.

Ist die Transportadresse des Gerätes bekannt, können weitere Informationen über das Gerät abgerufen werden. Diese Informationen beinhalten Gerätedetails (z.B. Hersteller, Gerätemodell) und Aussagen über die angebotenen Dienste. Für das Abrufen dieser gerätespezifischen Informationen wird der Mechanismus genutzt, der durch WS-Transfer *Get* definiert ist.

Für weitere Informationen über einen Dienst kann dieser direkt angesprochen werden. Dies geschieht durch das Nutzen von WS-MetadataExchange *GetMetadata*. Die umfassendsten Informationen eines Dienstes sind in der WSDL enthalten. Eine WSDL wird meist nur zur Entwicklungszeit benötigt. Viele Geräte sind bezüglich der Anwendung festgelegt und eher selten generisch gestaltet. Daher muss ein Dienst die WSDL nicht selbst anbieten, sondern kann lediglich eine Referenz übermitteln, die beschreibt, wo die vollständige WSDL abgerufen werden kann. Diese Anforderung ist bereits in den DPWS-Spezifikationen enthalten. Im Kontext von stark ressourcenlimitierten Umgebungen muss die Anforderung verstärkt werden. Ein Dienst darf seine

WSDL nicht selbst anbieten und kann optional in den Metadaten des Gerätes eine externe Referenz bereitstellen.

Durch die aufgeführten Anpassungen entstehen Redundanzen bezüglich der Nachrichtenabfolge beim Suchen und Finden eines DPWS-Gerätes, wodurch einzelne Teilschritte entfallen können. Diese Redundanzen sind in Abbildung 5.1 grafisch dargestellt.

5.1.2 Zentralisiertes Discovery

Der existierende Discovery-Mechanismus von DPWS nutzt zum Senden der Nachrichten IP-Multicast. Es existieren Umgebungen und Netzwerkinfrastrukturen, in denen Gruppenkommunikation (z.B. IP-Multicast) nur bedingt eingesetzt werden kann. Gründe hierfür können administrative Restriktionen sein. Für DPWS existiert daher eine Lösung, bei der Gruppenkommunikation vermieden werden kann. Dieser Modus wird als *Managed Mode* bezeichnet. Im *Managed Mode* wird ein zentraler Verwaltungsdienst genutzt, der als Discovery Proxy (DP) bezeichnet wird. Geräte teilen sich dem DP mit, indem sie die Nachrichten zur Ankündigung (*Hello* und *Bye*) Unicast an den DP senden. Clients können Suchanfragen ebenfalls Unicast an den DP senden, die von diesem beantwortet werden. Trotz des Vorhandenseins eines DPs können Clients und Geräte weiterhin Gruppenkommunikation verwenden, wodurch ein Ausfall, Fehler oder eine Nicht-Erreichbarkeit des DP nicht den Ausfall des gesamten DPWS-Netzwerkes bedeutet. Weiterhin können mehrere DPs für ein einzelnes Subnetzwerk vorhanden sein. Durch die entstehende Redundanz wird die Ausfallsicherheit ebenfalls erhöht.

DPs können, neben dem Vermeiden der Nutzung von Multicast, ebenfalls genutzt werden, um Subnetzwerke miteinander zu verbinden. Dazu müssen die DPs direkte Konnektivität mit beiden oder mehreren Subnetzwerken besitzen. Dabei ist es erforderlich, dass sich die DPs beim jeweiligen Router zwischen den Subnetzwerken befinden.

Ein einzelnes 6LoWPAN bildet innerhalb einer Infrastruktur jeweils ein einzelnes Subnetzwerk. Aufgrund der Ressourcenbeschränkungen der Zielplattformen ist es nicht möglich, einen DP auf jedem Router eines 6LoWPANs zu realisieren. Um das Suchen und Finden von Geräten und Clients über die Grenzen des 6LoWPANs hinaus zu ermöglichen, können DPs aber beim Border Router eingesetzt werden. Die DPs bilden dann Schnittstellen mit beiden Netzwerkumgebungen. Vom Border Router kann angenommen werden, dass dieser über genügend Ressourcen verfügt, um die Funktionen eines DPs zu realisieren. Eine mögliche Netzwerkinfrastruktur ist in Abbildung 5.2

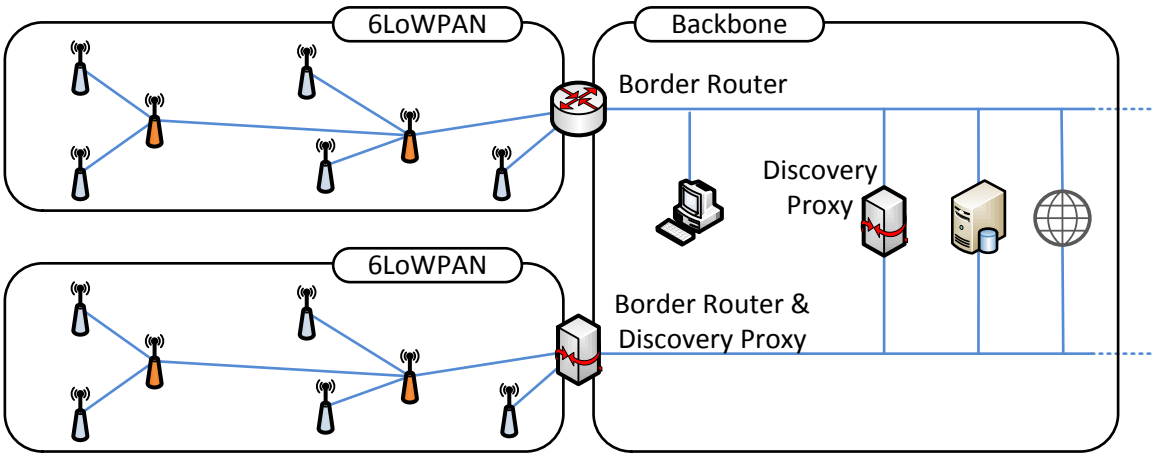

Abbildung 5.2 DPWS Discovery Proxys für 6LoWPANs

dargestellt. Durch die Entkoppelung mittels IP und der Möglichkeit DPs auch ausschließlich Unicast anzusprechen sind DPs weder bezüglich der Positionierung bei Routern oder Border Routern noch hinsichtlich einer spezifischen Anzahl gebunden (siehe Abbildung 5.2).

Pöhlsen stellt in [101] und [102] Konzepte für DNS-basiertes (Domain Name System) und DHCP-basiertes (Dynamic Host Configuration Protocol) Discovery für DPWS vor. Auch in diesen Lösungen ist das subnetzwerkübergreifende Discovery das Ziel.

Ungeklärt ist, wie solche zentralen Instanzen von Clients und Geräten zur Laufzeit gefunden werden können. In statischen Szenarien, bei denen die Adresse der zentralen Instanz zur Entwicklungszeit bereits bekannt ist, sind keine Mechanismen zur Laufzeit nötig. In dynamischen Szenarien und zur Erhöhung der Ausfallsicherheit in statischen Szenarien müssen Clients und Geräte in der Lage sein, auf sich ändernde Topologien und Netzwerkinfrastrukturen zu reagieren. Mögliche Umsetzungen zum Suchen und Finden der zentralen Entitäten sind Kommunikation mittels Anycast, DHCP oder Gruppenkommunikation. Welche der Umsetzungen geeignet ist, hängt von der Netzwerkinfrastruktur, der Topologie und der Anwendung ab.

5.1.3 Templates für Geräte

Anders als die Schnittstellen zum Suchen bzw. Finden eines Gerätes und für das Abrufen der Dienstinformationen kann die Vermeidung des Abrufens der gerätespezifischen Informationen zur Laufzeit nur durch Erweiterungen von DPWS erreicht werden. Ein Gerät kann in den Nachrichten *Hello* und *Probe Match* bereits Gerätetypen ankündigen und sich somit teilweise beschreiben.

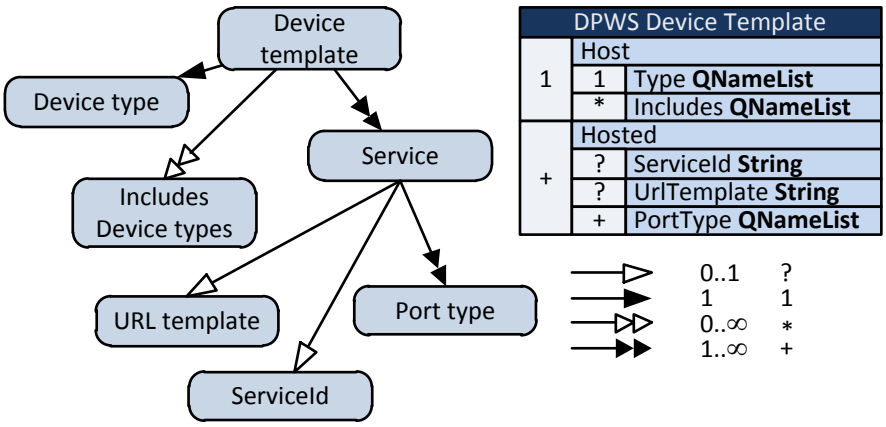

Abbildung 5.3 Gerätetemplates für DPWS

Diese Informationen beschränken sich aber auf eine Liste von *XML Qualified Names*. Diese Informationen genügen aber nicht, um die damit verbundenen Dienste hinreichend zu beschreiben und zu nutzen.

Die entsprechenden Details zu den Diensten sind unter anderem in der WSDL des Dienstes definiert. Für DPWS existiert keine definierte Zuordnung eines Gerätetyps zu den angebotenen Diensten. Dies ist unter anderem für die Implementierung von Clients hinderlich, da die Clients mit der entstehenden Variabilität umgehen können müssen. Für jedes Gerät muss durch den Client zur Laufzeit eine eigene Zuordnung vorgenommen werden, welche Dienste konkret von welchen Gerät angeboten werden.

Vor dem Hintergrund der Vereinfachung der Implementierung wurde in [103] durch Bobek et al. eine Sprache definiert, die es ermöglicht, Templates für Geräte zu beschreiben. Nach Bobek et al. repräsentiert ein DPWS-Gerätetyp mittels des Template-Konzeptes eine definierte Ansammlung von Diensten, die vom Gerät angeboten werden. Ähnliche Ansätze finden sich beispielweise bei UPnP.

Das Template-Konzept von Bobek et al. verknüpft neben den Geräten ebenfalls Dienste hierarchisch untereinander. Dazu werden DPWS-Diensttypen den WSDL Port Types zugeordnet. Da DPWS-Diensttypen sich entsprechend der Spezifikationen bereits direkt auf WSDL Port Types beziehen, ist diese Definition aber redundant und somit unwirksam.

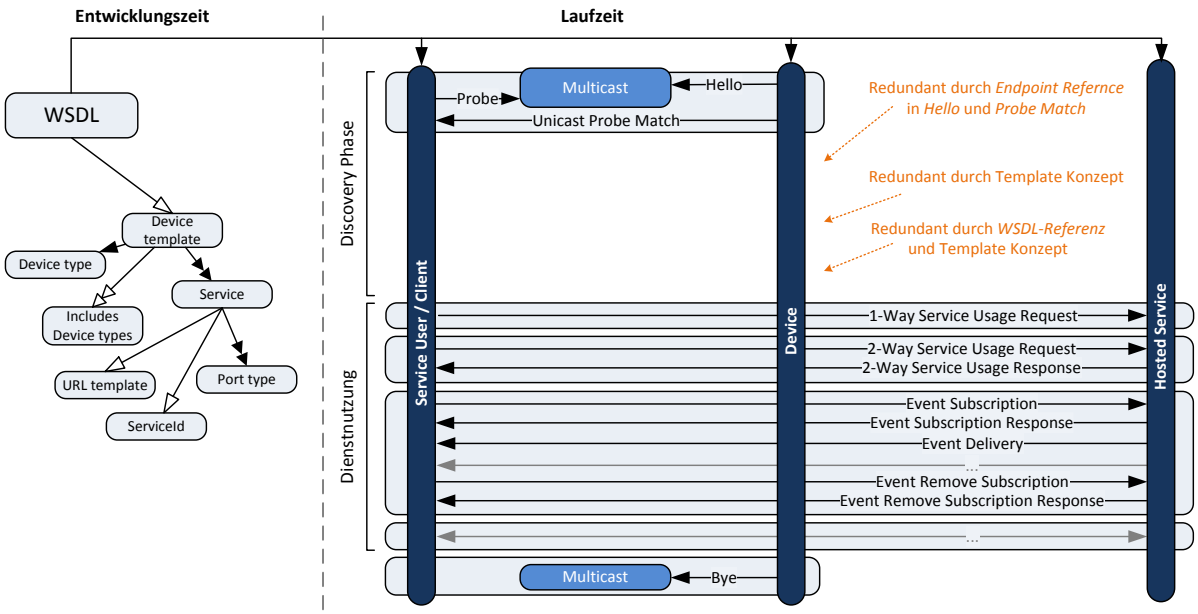

Abbildung 5.4 Device Template Konzept zur Laufzeit

Der von Bobek et al. initiierte Ansatz kann dennoch als Grundlage dienen und in wie nachfolgend konzipierter neugestalteter Form genutzt werden. Das redigierte Gerätetemplate stellt Abbildung 5.3 grafisch dar. Jedes Gerätetemplate beschreibt genau einen Gerätetyp. Weitere Gerätetypen können eingebunden werden, wodurch einfache Gerätetypen zu komplexeren kombiniert werden. Für die angebotenen Dienste enthält ein Gerätetemplate eine oder mehrere Angaben zu Dienste-IDs, WSDL Port Types und WS-Addressing Endpoint References des Dienstes. Die formale Beschreibung kann direkt in der WSDL erfolgen. Mit diesen Informationen können Clients die Dienste zur Laufzeitdirekt nutzen, sobald der Gerätetyp ermittelt wurde.

Als Ergebnis aus der Neukonzeption des Template-Ansatzes ergeben sich zwei Vorteile. Zum einen wird die Implementierung von Clients merklich vereinfacht, da die inhärente Variabilität vermieden wird, die durch die in DPWS unzureichende Zuordnung von Gerätetypen zu angebotenen Diensten entsteht. Ein Client kann genau für einen Gerätetyp implementiert werden, anstatt für spezifische Dienste entwickelt zu werden, die zur Laufzeit auf unbekannten Geräten zu finden sind. Zum anderen entfallen beim Discovery unter Umständen die Vorgänge des Abfragens der Metadaten des Gerätes und der Dienste, da diese Informationen nach dem Empfang der Nachrichten *Hello* bzw. *Probe Match* und den darin enthaltenen Angaben zum Gerätetyp bereits bekannt sind. In der Folge reduziert sich der Nachrichtenaustausch auf die Suche durch die Clients oder die Ankündigung des Gerätes beim Eintritt bzw. Verlassen des Netzwerkes und die

eigentliche Dienstnutzung (siehe Abbildung 5.4). Der Austausch von Metadaten zur Laufzeit ist nicht mehr zwingend erforderlich, um eine Dienstnutzung zur ermöglichen.

Ob diese zusätzlichen Abfragen für ein konkretes Szenario vermieden werden können, ist anwendungsabhängig. Durch das neu konzipierte Template-Konzept steht aber ein formaler Mechanismus zur Verfügung, welcher dies grundsätzlich möglich macht.

5.1.4 Realisierung von Gruppenkommunikation

Mit der Ausnutzung der bislang beschriebenen Erweiterungen verbleibt zur konkreten Umsetzung des Discovery-Vorganges die Forderung nach einer Form der Gruppenkommunikation. Gruppenkommunikation wird meist durch IP-Multicast realisiert, da Umsetzungen auf höheren Schichten als IP mittels sequentieller Nachrichten auf unteren Schichten umgesetzt werden müssen. Die Gruppenkommunikation mittels Abbildung auf sequentielle Nachrichten hat lediglich den Vorteil der besseren Abstraktion gegenüber höheren Schichten, erfordert aber mehr Bandbreite und Ressourcen.

Die beiden Gruppennachrichten *Hello* und *Bye* werden durch Geräte ausgesendet, um das Betreten oder Verlassen eines Netzwerkes mitzuteilen. Die Nachrichten *Probe* und *Resolve* werden durch Clients gesendet, um Geräte in einem Netzwerk anzusprechen. Für alle vier Nachrichten wird entsprechend der Spezifikationen die gleiche Multicastgruppe genutzt. Daher müssen die Nachrichten unter Umständen von Endpunkten empfangen und verarbeitet werden, ohne dass diese verwertet werden können. So können Clients *Probe* und *Resolve* Nachrichten nicht verwerten und verwerfen diese nach dem Empfang. Geräte können *Hello* und *Bye* Nachrichten nicht verwerten und verwerfen diese ebenfalls nach dem Empfang. Empfang und Verarbeiten der Nachrichten verursachen einen unzweckmäßigen Ressourcenverbrauch in 6LoWPANs, da die Sende- und Empfangseinheiten jedes Mal aktiviert werden müssen. Daher müssen in Abänderung zu den existierenden Spezifikationen getrennte Multicastgruppen für geräte- und clientgerichtete Nachrichten verwendet werden. Somit müssen nur Endpunkte den stromsparenden Modus verlassen und die Nachrichten verarbeiten, die diese auch verwerten können.

Diese Anpassung betrifft ausschließlich das jeweilige 6LoWPAN. Ein 6LoWPAN ist durch einen Border Router mit einem anderen Netzwerk verbunden. Da DPWS für Multicast einen Link Lokalen Geltungsbereich definiert, werden die Multicastnachrichten durch den Border Router nicht weitergeleitet. Es kann vorkommen, dass für spezifische Anwendungen ein erweiterter

Geltungsbereich für Multicast genutzt wird, der von den Border Routern weitergeleitet wird. Die entsprechenden Multicastadressen mit erweitertem Geltungsbereich sind aber auf die Adressierungsschicht und die Multicast-Verwaltungsprotokolle begrenzt.

5.1.5 Einsatz von Multicast in 6LoWPANs

Um zu betrachten, ob IP-Multicast in 6LoWPANs effizient eingesetzt werden kann, sind für die Bereiche (1) der Übertragungsschicht, (2) der 6LoWPAN-Protokolle und (3) des Routings getrennte Betrachtungen notwendig.

(1) *Übertragungsschicht:* Die für 6LoWPANs avisierte Übertragungsschicht IEEE 802.15.4 verfügt nativ nicht über die Möglichkeit, Multicast zu adressieren. Die Gruppenkommunikation von IEEE 802.15.4 wird daher auf Broadcast abgebildet und führt in der Folge u.U. zur Flutung des gesamten Netzwerkes. Wenn andere Übertragungsschichten für die 6LoWPAN-Protokolle über eine entsprechende Multicastadressierung verfügen, können diese Mechanismen entsprechend genutzt werden. Für eine allgemeingültige Lösung, die unabhängig von der Übertragungsschicht ist, müssen allerdings die höheren Schichten Multicast entsprechend abbilden und umsetzen.

(2) *6LoWPAN-Protokolle:* Die 6LoWPAN-Protokolle stellen im Kern lediglich eine Optimierung von IPv6 dar. Die Nutzung von Multicast ist nur rudimentärer Bestandteil der 6LoWPAN-Spezifikationen. So ist es in 6LoWPAN-basierten Netzwerken möglich, IPv6-Multicast immer auf Broadcast der Übertragungsschicht umzusetzen.

    Die Übertragungsschicht und die 6LoWPAN-Protokolle stellen somit grundsätzlich IP-Multicast zur Verfügung, können dies aber nur in einer ineffizienten Form mittels Broadcast abbilden.

(3) *Routing:* Eine mögliche Verbesserung entsteht durch den Einsatz eines dedizierten Routing-Protokolls, welches oberhalb von IP angesiedelt ist und Multicast effizient abbilden kann. Vor diesem Hintergrund wird von der IETF zum Zeitpunkt des Entstehens dieser Arbeit das Protokoll RPL (IPv6 Routing Protocol for Low Power and Lossy Networks) definiert.

Bei RPL wird zur Laufzeit proaktiv ein gerichteter, azyklischer Routinggraph entwickelt, der eine möglichst optimale Übertragung von der Quelle einer Nachricht zur Senke ermöglicht. RPL ist darauf optimiert, Daten möglichst effektiv aus einem 6LoWPAN herauszutransportieren. Die Senke des RPL-Graphen ist in einem solchen Fall der Border Router des 6LoWPANs.

Für die in das 6LoWPAN eingehenden Unicast Nachrichten und für Multicast ist es bei RPL nötig, dass dem Border Router das gesamte 6LoWPAN und die Topologie bekannt ist. Hierzu kündigt sich jeder Endpunkt beim Border Router an und teilt mit, welche weiteren Endpunkte über ihn erreichbar sind. Die Endpunkte, über die der Zugang zu weiteren Endpunkten möglich ist, werden als Router bezeichnet.

Der Border Router kann mittels Source Routing[2] alle Endpunkte im 6LoWPAN erreichen. Das genutzte Source Routing des Border Routers ist ähnlich zu anderen *Link State*[3] Routingverfahren. Um zu vermeiden, dass alle Router innerhalb eines 6LoWPANs Routingtabellen mit Informationen über die gesamte Topologie des Netzwerkes verwalten, können bei RPL innerhalb des 6LoWPANs *Distance Vector*[4] Routingverfahren genutzt werden. Bei *Distance Vector* Verfahren müssen Router nur die direkten Nachbarn kennen. Da RPL einen gerichteten Graph nutzt, genügt für *Distance Vector* Verfahren die Verwaltung der Nachbar-Router, die in der Topologie näher an der Senke liegen.

Der Einsatz von *Distance Vector* Verfahren Vermeidet die Verwaltung von Routingtabellen auf den Routern, impliziert aber zur Kommunikation innerhalb des 6LoWPANs, dass die Nachrichten von den Endpunkten immer vollständig bis zum Border Router traversieren und dann durch Source Routing wieder in das 6LoWPAN hineinlaufen. Dieser Modus wird als *Non-Storing Mode* bezeichnet. Um dies zu vermeiden, kann bei RPL für die Router ein hybrider Modus aus *Distance Vector* und *Link State* Routingverfahren genutzt werden. Dieser Modus wird als RPL *Storing Mode* bezeichnet. Bei diesem Modus müssen die Router im 6LoWPAN ebenfalls eigene Routingtabellen verwalten, in denen die angeschlossenen Subnetzwerke enthalten sind. Eine Unicast-Nachricht muss in diesem Modus nur bis zum nächsten gemeinsamen Router traversieren und werden von dort aus mittels Source Routing weitergeleitet. Die RPL Modi *Storing* und *Non-Storing* und deren Auswirkung auf die Kommunikation sind in Abbildung 5.5 grafisch veranschaulicht.

[2] Vollständige Wegsequenz wird vor Absenden festgelegt und in Datenpaket integriert.
[3] Topologie des gesamten Netzwerkes ist den Routern weitestgehend bekannt.
[4] Topologie der unmittelbaren Nachbarn ist bekannt.

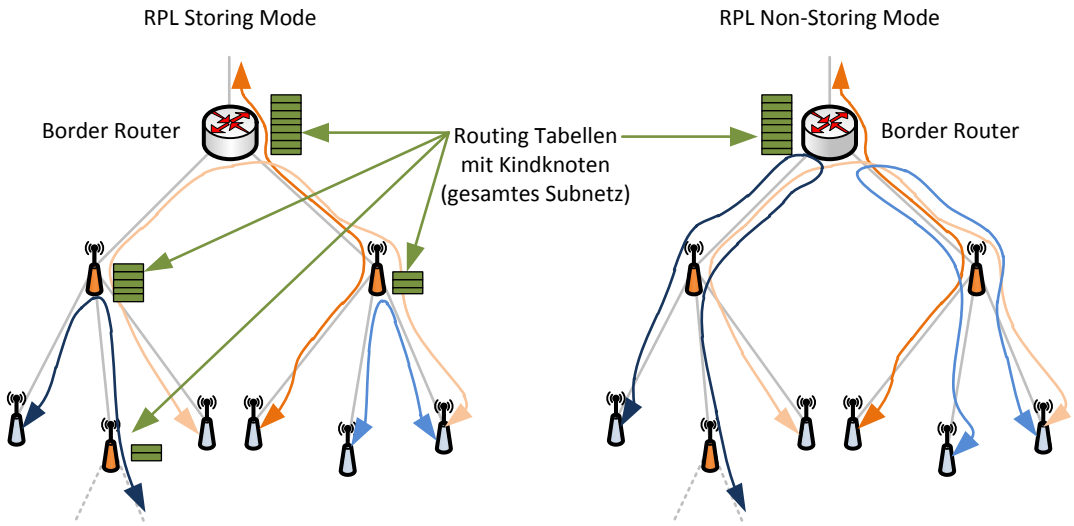

Abbildung 5.5 RPL Storing und Non-Storing Mode

Der Einsatz von Multicast erfordert entsprechend der RPL-Spezifikationen immer den RPL *Storing Mode*. Multicast Nachrichten müssen aber dennoch vollständig bis zum Border Router traversieren, da die Zwischenrouter kein Abbild des gesamten Netzwerkes und der Multicast-Gruppenmitglieder enthalten, sondern nur das der unterhalb von ihnen liegender Subnetzwerke.

Es ist ersichtlich, dass die derzeit für RPL definierten Mechanismen zur Realisierung von Multicast für eine große Anzahl von Knoten nicht skalieren. Speziell in der Nähe der Senke tritt ein hohes Nachrichtenaufkommen auf, da sich jeder Endpunkt des 6LoWPANs dem Border Router mitteilen und jede Multicast-Nachricht bis zur Senke traversieren muss.

Weiterhin können im RPL *Storing Mode* in großen Netzwerken sehr große Routingtabellen auftreten. In solchen großen Netzwerken müssen Router, die innerhalb der Topologie nah am Border Router liegen, das gesamte über ihn erreichbare Subnetzwerk verwalten. Ko et al. haben in [36] evaluiert, dass im *Storing Mode* von RPL auf einem TelosB-Knoten [93] mit TinyOS [97] etwa bis zu 30 Einträge in der Routingtabelle verwaltet werden können. Um für die Nutzung von Multicast-Routen zu allen Endpunkten zu verwalten ist somit die Größe des Netzwerks auf etwa 30 Knoten beschränkt. Nach Ko et al. kann die Verwaltung der Routingtabellen durch geeignete Optimierungen der Speicherstrukturen verbessert werden. Dennoch geben diese Untersuchungen einen guten Eindruck von den Größen der Routingtabellen im RPL *Storing Mode* für Plattformen der Klasse 1 (siehe Kapitel 4.4).

Ferner gilt es zu beachten, dass für das Source Routing alle zu traversierenden Zwischenstationen in das jeweilige Datenpaket integriert werden müssen. Eine vollständige IPv6-Adresse ist 16 Byte lang. Um eine effiziente Adressierung der Routen zu ermöglichen, müssen somit auch für das Source Routing neuartige Konzepte zur Beschreibung der Wegpunkte entlang des Routingpfades entwickelt werden. Um die Übertragungsschicht IEEE 802.15.4, die Adressierung mittels 6LoWPAN und das Routingprotokoll RPL nicht zu stark aneinander zu koppeln und somit direkte oder indirekte Abhängigkeiten zu schaffen gilt es hierfür zu untersuchen, inwieweit die für die 6LoWPAN-Protokolle existierende Strategie der Vermeidung von Redundanzen auch auf das Source Routing angewandt werden kann.

Über die entstehende Topologie, bestehend aus Endpunkten, Routern und dem Border Router, ist in der Folge ebenfalls der Geltungsbereich für die IP-Adressierung definiert. Eine Nachricht, welche an eine Link Lokale Adresse gesendet wird, wird von IP-Routern nicht weitergeleitet. In einem 6LoWPAN ist durch die 6LoWPAN-Protokolle und RPL jeder Router ein IP-Router. Folglich sind Link Lokale Nachrichten auf die Reichweichte des Senders beschränkt (siehe Abbildung 5.6). Es können nur Ziele erreicht werden, die genau einen physikalischen Hop entfernt sind. Da DPWS für das Discovery Link Lokale Multicastadressen nutzt, ist das existierende Discovery ohne Anpassungen in einem Multihop-Netzwerk nicht anwendbar.

Eine mögliche Lösung, um Discovery Nachrichten in das gesamte 6LoWPAN senden bzw. vom gesamten 6LoWPAN empfangen zu können, liefert die Nutzung einer Multicastadresse mit einem anderen Geltungsbereich als Link Lokal. Die Nutzung eines globalen Geltungsbereiches erweist sich als nicht zweckmäßig, da dies auch externe Netzwerke beeinflusst und diese unter Umständen bei jedem Auftreten von Multicast flutet. Die Nutzung eines Geltungsbereiches, der sich ausschließlich auf ein einzelnes 6LoWPAN beschränkt, muss im Zusammenhang mit einem geeigneten Routingverfahren und einem geeigneten Gruppenverwaltungsprotokoll betrachtet werden.

Bezüglich des Einsatzes von Multicast in 6LoWPANs muss daher festgestellt werden, dass zum Zeitpunkt des Entstehens dieser Arbeit mit den existierenden Protokollen und Mechanismen keine effiziente Gruppenkommunikation in großen 6LoWPANs möglich ist. Innerhalb der IETF gibt es Bemühungen, die existierenden Probleme zu lösen. Ob und wann sich spezifische Lösungen durchsetzen und diese standardisiert werden, ist bislang nicht abzusehen.

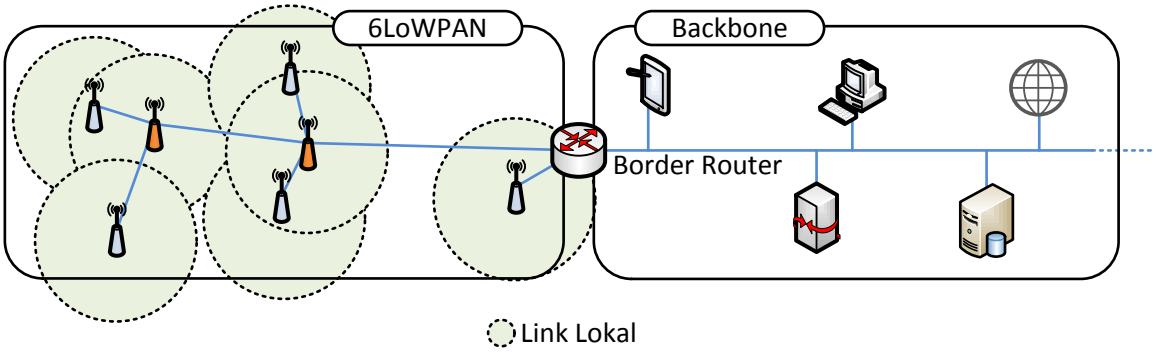

Abbildung 5.6 Link Lokaler Gültigkeitsbereich in 6LoWPANs

Die genannte Problemstellung bezüglich der Gruppenkommunikation positioniert sich unabhängig von DPWS und bedarf ganzheitlicher Konzepte. Es gilt zu klären, welche Minimalanforderungen an Router in Abhängigkeit von der Größe und dem Aufbau des 6LoWPANs gestellt werden, um effizientes IP-Multicast zu ermöglichen. Weiterhin gilt es zu untersuchen, wie sich die verschiedenen Umsetzungsverfahren für Multicast auf den Energiehaushalt des gesamten 6LoWPANs auswirken. Mögliche Kandidaten für die Umsetzung von Multicast sind sequentielles Unicast, Broadcast oder kontrolliertes Fluten wie beim Trickle-Algorithmus [104]. Weil sich die Problemstellung als äußerst komplex erweist und weil sich dedizierte Routingverfahren innerhalb der IETF in der Entstehung befinden, kann die Entwicklung eines geeigneten Ansatzes für Multicastkommunikation innerhalb eines gesamten 6LoWPANs nicht Bestandteil dieser Arbeit sein.

Der effiziente Einsatz von Multicast für das Discovery in DPWS beschränkt sich folglich auf kleine Netzwerke mit einem physikalischen Hop. Bei diesen kleineren Netzwerken hat die jeweilige Flutung des gesamten 6LoWPANs, hervorgerufen durch die Umsetzung von IP-Multicast auf Übertragungsschicht-Broadcast, einen geringeren Einfluss auf die Leistungsfähigkeit und den Energiehaushalt des 6LoWPANs. Für Multihop 6LoWPANs muss auf zentralisierte Verfahren, wie beispielsweise den Einsatz von Discovery Proxys, zurückgegriffen werden. Durch das Vorhandensein der einheitlichen Adressierungsschicht IP bleibt die Position der zentralen Entität für das Discovery nicht auf das 6LoWPAN oder ein anderes Subnetzwerk beschränkt. Die zentrale Entität kann auch außerhalb des jeweiligen 6LoWPANs oder Subnetzwerkes positioniert sein, was die Lage im Internet selbst nicht ausschließt.

5.2 Kommunikationsmuster

Sind den potenziellen Clients die Dienste eines Gerätes bekannt, können die Daten bzw. Dienste durch Abfragen abgerufen bzw. genutzt werden. Diese Form der Kommunikation wird als Pull bezeichnet. Nutzdaten können sowohl in der Anfrage also auch in der Antwort enthalten sein. Zyklisches Abfragen von Daten und Zuständen wird auch als Polling bezeichnet. Um dies zu vermeiden, können Push-Benachrichtigungen genutzt werden, bei denen die Daten durch das Gerät bzw. den konkreten Dienst bei Änderungen an den Client übermittelt werden. Welche Push-Benachrichtigungen bei welchen Zustandsänderungen durch den Server bzw. Dienst an den Client gesendet werden sollen, muss vom Client zuvor entsprechend initiiert werden.

Welche der beiden Kommunikationsformen effizienter ist, hängt von der konkreten Anwendung und dem damit verbundenen Verhältnis zwischen Abfrageintervallen der Clients und den Änderungsintervallen der zur Verfügung gestellten Daten ab. Sind die Abfrageintervalle gegenüber der Änderungsintervalle vergleichsweise groß, kann Pull genutzt werden. Sind die Abfrageintervalle gegenüber der Änderungsintervalle vergleichsweise klein, sollte Push bevorzugt werden, um die zyklische Übermittlung identischer und somit redundanter Daten zu vermeiden.

5.2.1 Feingranulares Push durch parametrierbare Event-Filter

Die Push-Kommunikation wird bei DPWS mittels WS-Eventing ermöglicht. WS-Eventing definiert Mechanismen, wie die asynchronen Benachrichtigungen (Events) von einer Quelle (Gerät bzw. Dienst) zu einer oder mehreren Senken (Clients) übertragen werden. Filter für das Eventing ermöglichen die Anpassung, welche Informationen bei welchen Änderungen übertragen werden. Der einzige existierende Filter bezieht sich auf die Operationen bzw. Methoden der Dienste selbst.

Am Beispiel eines Temperatursensors bedeutet dies, dass alle Änderungen der Temperatur immer übermittelt werden. Schwellwertüberwachungen, wie beispielsweise Benachrichtigungen beim Übersteigen einer bestimmten Temperatur, sind mit den existierenden Mechanismen nicht möglich.

Zusätzliche Parameter für Event-Filter erweisen sich als essenziell für den Einsatz von DPWS in 6LoWPANs, da Schwellwert- und sonstige generelle Werteüberwachungen zu den häufigsten Anwendungsszenarien zählen. Der existierende einfache Filtermechanismus kann dahingehend erweitert werden, dass weitere parametrierbare Filter zur Eingrenzung auf spezifische Parameter

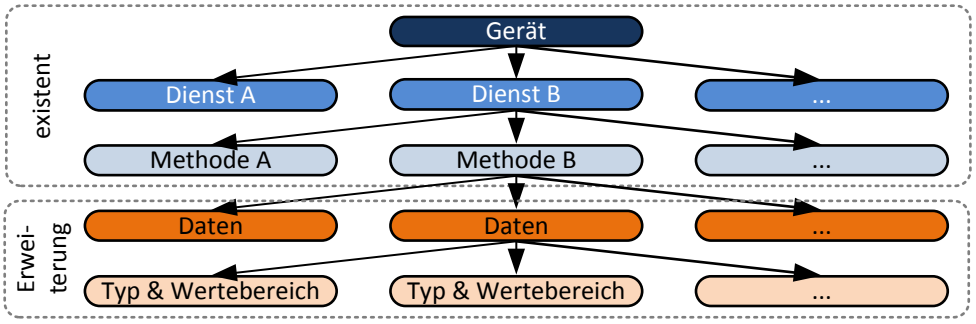

Abbildung 5.7 Erweiterung Filtermechanismus Eventing

und den relevanten Wertebereich definiert werden können (siehe Abbildung 5.7). Die Definition dieses parametrierbaren Event-Filters stellt die Abbildung 5.8 schematisch dar.

Jeder Parameter bezieht sich auf eine spezifische Methode des Dienstes bzw. der *WS-Addressing Action*, wie dies bereits beim existierenden WS-Eventing Event-Filter definiert ist. Dadurch ist der erweiterte Event-Filter kompatibel mit WS-Eventing. Implementierungen, die den erweiterten Filter nicht unterstützen, können die zusätzlichen Parameter ignorieren. In diesem Fall gestaltet sich die Kommunikation aber weniger effizient.

Jeder Parameter muss von einem bestimmten Typ sein. Definiert werden die Typen *Range*, *Step* und *Periodic*. Der Typ *Range* bezeichnet einen bestimmten Wertebereich. Solange sich der Wert innerhalb dieses Bereiches bewegt, wird bei jeder Änderung ein Event ausgelöst. Der Typ *Step* ist dem Typ *Range* grundsätzlich ähnlich. Beim Typ *Step* wird allerdings immer nur dann ein Event einmalig ausgelöst, wenn der spezifizierte Schwellwert überschritten wird. Der Typ *Periodic* kann eingesetzt werden, wenn unabhängig von der tatsächlichen Änderung des Wertes zyklische Benachrichtigungen erwünscht sind.

Beim Typ *Periodic* wird der gewünschte Zeitraum zwischen zwei Benachrichtigungen als *Value* im Parameter eingebettet. Ein *Key* und die Angabe des *Bound Types* sind für den Typ *Periodic* nicht nötig. Clients, die sich für Events *subscriben*, müssen aber damit umgehen können, wenn die gewünschten Perioden nicht eingehalten werden können. Dies kann unter anderem der Fall sein, wenn der Event-Quelle bereits bekannt ist, dass Events nur in bestimmten Abständen versendet werden können (z.B. durch stromsparende Schlafzustände). In der *Subscription Response* kann die Event-Quelle den Client darauf hinweisen, welcher Wert als Periode akzeptiert wurde.

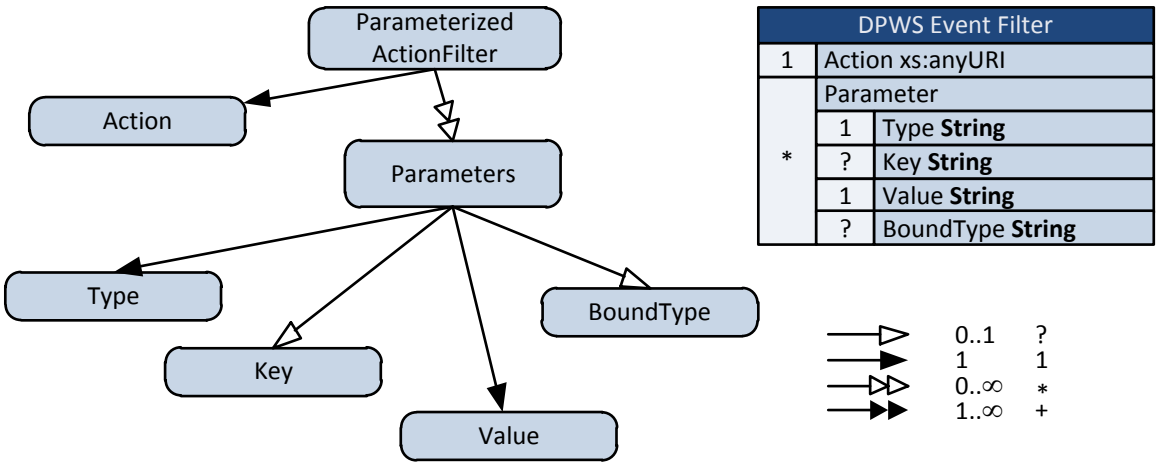

Abbildung 5.8 Parametrierbarer Event-Filter

Für die Typen *Range* und *Step* wird der jeweilige Wert mithilfe von *Key-Value* Paaren festgelegt. Der *Key* definiert die Daten und der *Value* den eigentlichen Wert. Weiterhin enthalten die Parameter vom Typ *Range* und *Step* durch den *Bound Type* Angaben darüber, ob der Wert eine obere oder untere Grenze definiert bzw. ob der angegebene *Value* einen diskreten Wert und keine Grenze definiert (gültige Angaben sind <, >, <=, >= und =).

Mehrere Parameter können miteinander in beliebiger Anzahl kombiniert werden. Dazu werden diese mittels eines logischen ODER miteinander verknüpft. So benötigt man für die Definition eines spezifischen Wertebereiches mindestens zwei Parameter. Der eine definiert die obere und der andere die untere Grenze. Mittels des *Bound Types* der beiden Parameter kann angegeben werden, ob Events für Werte innerhalb oder außerhalb des Bereiches ausgelöst werden. Mittels der Parameter Typen *Range* oder *Step* kann definiert werden, ob Benachrichtigungen nur einmal beim Überschreiten der Schwellwerte oder auch bei jeder weiteren Änderung innerhalb des definierten Bereiches ausgesendet werden sollen. Zusätzlich können periodische Benachrichtigungen angefordert werden die unabhängig davon sind, ob der Wert sich geändert hat oder nicht.

Mit welcher Semantik der Wert einer oberen bzw. unteren Grenze belegt ist, kann nicht pauschal definiert werden. Im Beispiel eines einfachen Temperatursensors kann eine Grenze den jeweils aktuellen Wert der gemessenen Temperatur beschreiben. Bei komplexeren Diensten, die weiterführende Daten bereitstellen und komplexere Datenmodelle als einfache Zahlenwerte nutzen, ist die Bedeutung anwendungsabhängig. Durch das Vermeiden der Festlegung auf eine bestimmte

Semantik bleibt der beschriebene Filtermechanismus allgemeingültig und beschränkt sich nicht auf einzelne Szenarien und Datentypen.

Die Parameter werden bei der *Event-Subscription* bzw. Änderungen der *Subscription* durch die Senke mit übermittelt. Ist die Quelle nicht in der Lage, die gewünschten Parameter zu beachten, so teilt sie dies der Senke in der *Subscription-Response* mit. In diesem Fall entscheidet die Senke, ob die *Subscription* in der Folge ohne die Parameter bestehen bleibt, aufgehoben oder verändert werden soll.

Wie die auftretenden Events von der Quelle an die Senke übermittelt werden, bestimmt der *Delivery Mode*. Der in WS-Eventing bestehende *Delivery Mode* erfordert es, dass bei jedem auftretenden Event zu jeder Senke eine Verbindung aufgebaut wird und die Nachrichten übertragen werden. Bei mehreren Senken bedeutet dies, dass mehrere aufeinander folgende sequentielle Übermittlungen nötig sind, die identische Nutzdaten enthalten.

Eine mögliche bessere Lösung bietet die Nutzung von Punkt-zu-Mehrpunkt (Multicast) Kommunikation für die Zustellung von Events. Ist eine solche Kommunikation effizient möglich (siehe Diskussion in Kapitel 5.1.5), kann die Senke in der *Subscription* einen entsprechenden *Delivery Mode* definieren. Kann die Quelle diese Anforderung erfüllen, teilt diese in der *Subscription-Response* die entsprechenden Details zur Kommunikation mit, die für die Zustellung der Events genutzt werden. Eine mögliche Information ist die genutzte Multicastgruppe. Kann die Quelle die Anforderung nicht erfüllen, so ist die *Subscription* nicht möglich und wird mit einer entsprechenden Fehlermeldung beantwortet. Ist die Senke nicht in Lage, die gewünschten Anforderungen bezüglich der Kommunikationsdetails zu erfüllen, wie beispielsweise der konkreten Multicastgruppe beizutreten, so muss diese die *Subscription* erneuern bzw. ändern und einen anderen *Delivery Mode* wählen oder die *Subscription* beenden.

WS-Eventing definiert, dass ein zugestelltes Event zur Identifizierung und Zuordnung eine spezielle Referenz enthalten kann. Die Referenz wird durch die Senke erzeugt, die bei der *Subscription* der Quelle mitgeteilt und bei allen folgenden Zustellungen des Events von der Quelle wiedergegeben wird. Für die Quelle bleiben der Inhalt und die Semantik der Referenz ohne weitere Bedeutung. Wird als *Delivery Mode* eine Form der Gruppenkommunikation verwendet, kann der Mechanismus der Referenzierung nicht unterstützt werden. Es würde bedeuten, dass alle Referenzen aller Senken bei der Zustellung des Events enthalten sein müssen. Dies kann zum einen

zu Doppelungen und Fehlern bei der Verarbeitung durch die Senken und zum anderen bei vielen Senken und einer großen Zahl von Referenzen zu sehr großen Nachrichten führen.

Bei der Nutzung von IP-Multicast muss durch geeignete Maßnahmen ebenfalls sichergestellt werden, dass keine Überlappungen der genutzten Multicastgruppen für die Zustellung der Events mit anderen Quellen oder gänzlichen anderen Protokollen als DPWS auftreten. Mögliche Maßnahmen sind administrative Vorgaben oder die Nutzung geeigneter Multicast-Verwaltungsprotokolle, wie beispielsweise das Internet Group Management Protocol (IGMP) [105]. Welche Maßnahme für die jeweilige Anwendung und Umgebung am geeignetsten ist, kann nicht pauschal spezifiziert werden. Für die ganzheitliche Spezifizierung im Rahmen einer umfassenderen Festlegung, beispielsweise in Form eines Geräte- oder Anwendungsprofils einer dedizierten Domäne, kann unter Umständen ein entsprechender Adressraum bei Standardisierungsverzeichnissen wie der Internet Assigned Numbers Authority (IANA) reserviert werden.

5.3 Implementierung von DPWS für stark ressourcenlimitierte Geräte

Um den Nachweis zu erbringen, dass DPWS mit den Schnittstellenmechanismen, Darstellungsformaten und Übertragungsprotokollen auch auf stark ressourcenlimitierten Zielplattformen realisiert werden kann, wurde ein entsprechender DPWS-Stack implementiert. Die entstandene Lösung heißt uDPWS. Die Bezeichnung lehnt sich an uIP [68], einen IP-Stack für 8-Bit Mikrokontroller, an.

uDPWS stellt die erste spezifikationskonforme Implementierung von DPWS für stark ressourcenlimitierte Zielplattformen dar und liefert somit erstmals den Nachweis der Machbarkeit. Existierende Lösungen, wie beispielsweise WS4D-gSOAP [106] oder SOA4D [107], die beide auf dem gSOAP-Toolkit [108] basieren, sind aufgrund ihres hohen Ressourcenverbrauchs nur für eingebettete Systeme nutzbar. Solche eingebetteten Systeme verfügen über mehr dynamischen und statischen Speicher. Dies schließt Zielplattform für 6LoWPANs aus.

Die zugrundliegende ressourcensparende Konzeption von uDPWS und der Nachweis der Umsetzbarkeit wurden 2008 in [109] und [110] veröffentlicht. Die prototypische Implementierung erfolgte 2010 im Rahmen der Masterarbeit von Lerche [111]. Im Rahmen dieser prototypischen Implementierung wird uDPWS seit 2010 unter der Bezeichnung WS4D-uDPWS im

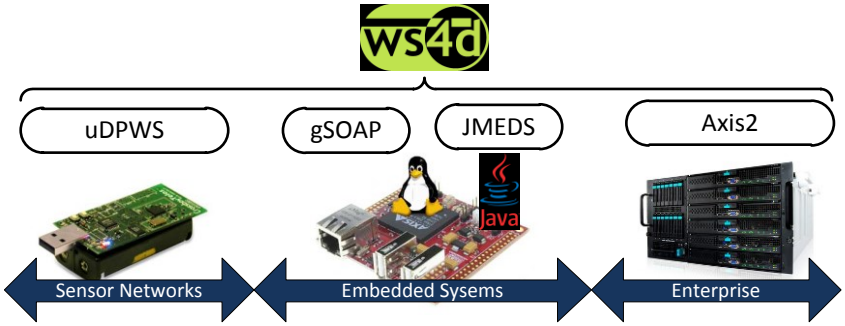

Abbildung 5.9 Eingliederung uDPWS in WS4D-Tools

Zusammenhang mit der WS4D-Initiative [90] als Open Source Projekt publiziert und aktiv gepflegt. WS4D-uDPWS ergänzt die existierenden Implementierungen der WS4D-Initiative somit um eine Lösung für 6LoWPAN-Netzwerke (siehe Abbildung 5.9). Konkrete Details bezüglich der Implementierung sind über die Webseite des WS4D-uDPWS Projektes erhältlich. Dieses Unterkapitel beschränkt sich auf eine konzeptionelle Beschreibung von uDPWS sowie die Darstellung resultierender Ergebnisse hinsichtlich der Leistungsfähigkeit und des Ressourcenverbrauches.

DPWS-Clients erfordern zum einen meist das Verwalten eines Zustandes und sind damit ressourcenaufwändiger als Dienste bzw. Geräte. Zum anderen sind für DPWS und den damit verbundenen WS-* Spezifikationen die Schnittstellen des Gerätes und der Dienste beschrieben. Das Verhalten der Clients wird meist nicht beschrieben, sondern ergibt sich implizit. Clients müssen sich dem Verhalten der Geräte und der Dienste somit anpassen. Die durch die Freiheitsgrade der Spezifikationen entstehende Variabilität verursacht eine nicht vermeidbare Komplexität bei der Implementierung von Clients. Durch die erhöhte Komplexität bei der Implementierung von Clients würde somit durch uDPWS nicht die untere Grenze für DPWS aufgezeigt werden können. Weiterhin wurden in Kapitel 2 Anwendungsszenarien identifiziert, die keine direkte Kommunikation zwischen den Knoten im 6LoWPAN erfordern, sondern eine Einbindung in höherwertige Dienstkompositionen. Folglich stellt die Implementierung uDPWS die Realisierung eines DPWS-Gerätes und der damit verbundenen Dienste dar. uDPWS ist auf konformes Verhalten geprüft. Als Clients stehen hierfür die unabhängigen Implementierungen für DPWS von der WS4D-Initiative zur Verfügung, welche die WS-DD Interoperabilitätskriterien [91] erfüllen.

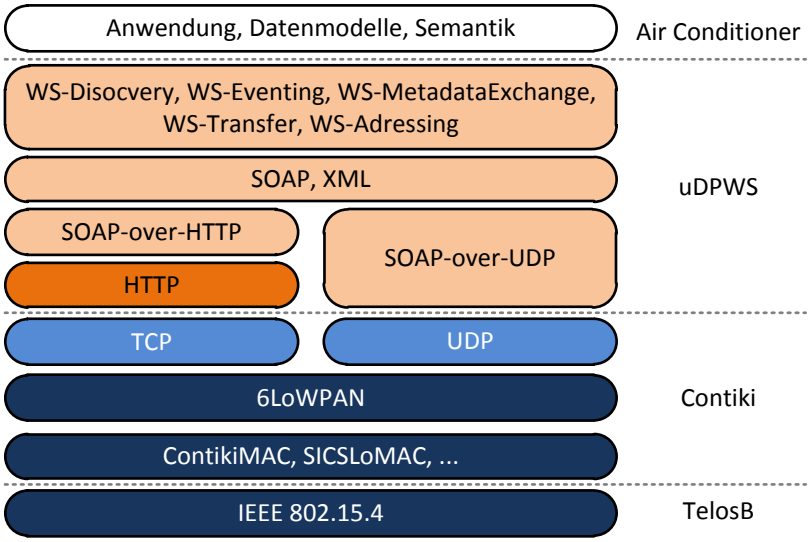

Abbildung 5.10 Implementierte Protokolle von uDPWS

Der uDPWS-Stack ist mit den DPWS 1.1 Spezifikationen in den meisten Punkten konform. Dies beinhaltet die Unterstützung von HTTP und UDP als Transportprotokoll sowie die Verarbeitung von Unicode kodierten XML-Dokumenten. Die implementierten Protokolle sind in Abbildung 5.10 als Protokollstapel dargestellt.

Die nicht von uDPWS unterstützten Bestandteile der DPWS-Spezifikationen beziehen sich vornehmlich auf Aspekte hinsichtlich der Daten- und Transportsicherheit. Die Unterstützung der Daten- und Transportsicherheit würde unter anderem die Nutzung von Transport Layer Security (TLS) bzw. Secure Sockets Layer (SSL) verschlüsselte Verbindung erfordern. Die für die Kryptographie zu verarbeitenden Zertifikate haben eine typische Größe von etwa 1024 Byte. Für die Integritätssicherung müssen weiterhin komplexe Zertifizierungsalgorithmen unterstützt werden, die das sequentielle Abfragen verschiedener Zertifizierungsstellen inkl. Validierungsmaßnahmen erfordern. Eine leichtgewichtigere Alternative zur TSL bzw. SSL stellt das UDP-basierte Datagram Transport Layer Security (DTLS) [112] dar. Bis zum Zeitpunkt der Entstehung dieser Arbeit existiert aber auch für DTSL kein Mechanismus, der trotz der Ressourcenlimitierungen zur Laufzeit eine effiziente Integritätssicherung ermöglicht. Diese Thematik ist unabhängig von DPWS und stellt auch für andere Anwendungsschichtprotokolle eine immense Herausforderung dar. Details zur genannten Problemstellung werden z.B. von Unger et al. in [113] diskutiert.

5.3.1 Implementierung

Für die Realisierung von uDPWS wurde auf das existierende Open Source Betriebssystem Contiki [98] zurückgegriffen. Die aktuell von uDPWS unterstützte Version von Contiki trägt die Bezeichnung 2.5. Als Hardwareplattform unterstütz uDPWS den TelosB, der bereits in Kapitel 4.4.3 vorgestellt wurde sowie den Atmel AVR Raven. Da die Nutzung des TelosB im Vergleich mit dem Atmel AVR Raven in ressourcenoptimaleren Ergebnissen resultiert und eine bessere Leistungsfähigkeit aufweist, wird im Folgenden nur auf den TelosB eingegangen.

In Abbildung 5.11 sind die implementierten Module von uDPWS dargestellt. Die Verarbeitung von Anfragen stellt Abbildung 5.12 dar. Zielplattformen, die starken Ressourcenbeschränkungen unterliegen, bieten keine komplexen Dienste an. Daher ist es ausreichend, eine sehr kleine Untermenge an SOAP-Headerelementen zu unterstützen und zur Laufzeit auszuwerten. Anwendungs- oder herstellerspezifische Erweiterungen können dennoch in uDPWS integriert werden. Basierend auf dem Dienstaufruf und der jeweiligen Operationen werden der SOAP-Body durch das dazugehörige Dienst-Modul ausgewertet und die entsprechenden Nutzdaten für die Antwort generiert. Die Aktionen, die sich durch die Dienstaufrufe ergeben und auf der Zielplattform ausgeführt werden müssen, werden ebenfalls von den Dienst-Modulen implementiert. Die Nutzdaten werden vor dem Absenden der Antwort von uDPWS durch die entsprechenden SOAP-Headerelementen ergänzt und zu einem vollständigen SOAP-Dokument zusammengefügt. uDPWS realisiert ebenfalls die gesamte Verwaltung der Verbindung während des Aufrufs, wodurch dies vollständig vor der Anwendung verborgen werden kann.

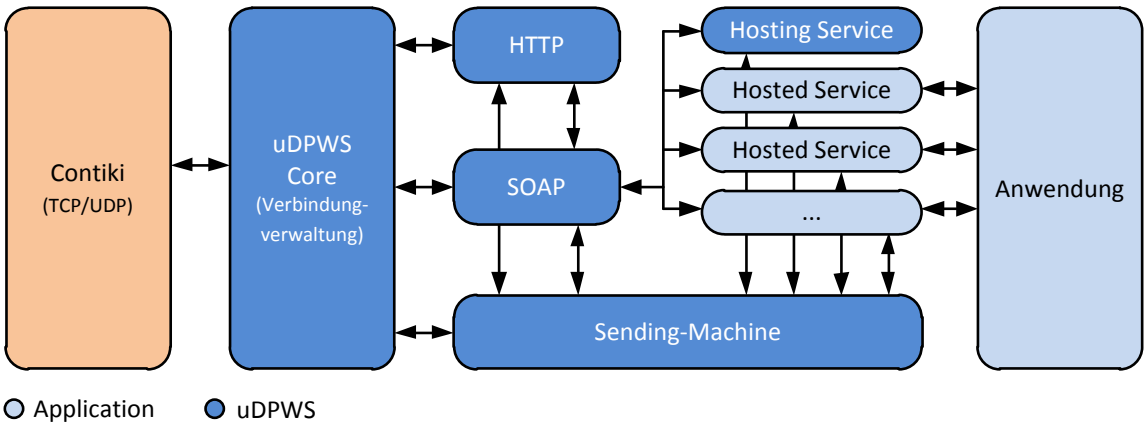

Abbildung 5.11 uDPWS Module

Das Aussenden von *Event-Notifications* durch uDPWS findet, gegenüber der Abarbeitung von externen Anfragen, in einer abgeänderten Form statt. Für *Event-Notifications* muss uDPWS selbst die Verbindung initiieren. Im Falle des Auftretens eines Events werden die Nutzdaten durch das entsprechende Dienst-Modul generiert. Die Nutzdaten werden an die *Sending-Machine* von uDPWS übergeben. Der *Sending-Machine* wird beim Übergeben entsprechend angezeigt, dass es sich um eine *Event-Notification* handelt. Dadurch wird durch die *Sending-Machine* eine Verbindung zur Event-Senke initiiert und ggfs. eine Synchronisierung mit anderen gleichzeitigen parallelen Dienstoperationen oder *Event-Notifications* vorgenommen.

Die ausgehenden Nachrichten, wie beispielsweise Antworten auf Anfragen oder Events, sind direkt von den Fähigkeiten der physikalischen Zielplattform und den damit verbundenen Sensoren bzw. Aktoren abhängig. Aus diesem Grund sind die meisten Bestandteile und Strukturen der ausgehenden Nachrichten, im Gegensatz zu den eingehenden Nachrichten, bereits zur Entwicklungszeit bekannt. Mögliche Clients müssen die entstehende Variabilität bei der

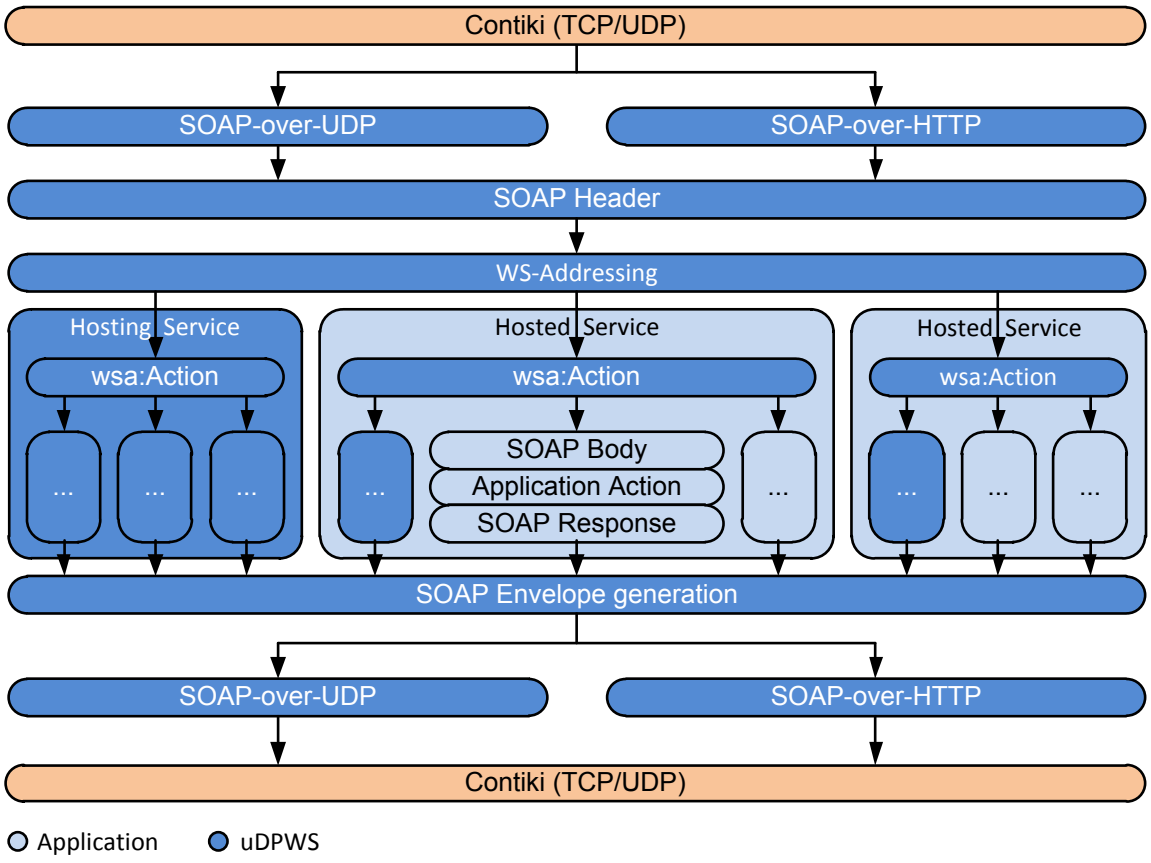

Abbildung 5.12 uDPWS interne Verarbeitung

Kommunikation beachten, da die Geräte bzw. Dienste durch die Ausprägung ihrer Schnittstellen die Kommunikation definieren. Nur wenige Bestandteile der ausgehenden Nachrichten ändern sich zur Laufzeit.

Abbildung 5.13 zeigt beispielhaft für eine spezielle Nachricht aus dem Referenzszenario die statischen und dynamischen Anteile. Die dynamischen Anteile ergeben sich zum einen aus der Anwendung selbst (Nutzdaten) und zum anderen aus der Anfrage (z.B. Empfänger der Antwort). Beim uDPWS-Stack werden daher alle potenziellen ausgehenden Nachrichten bereits zur Entwicklungszeit generiert. Die statischen Anteile der Nachrichten werden in die Implementierung eingebunden. Zur Entfernung von Redundanzen werden diese zuvor in Fragmente zerteilt und identische Elemente somit vermieden. Die kleinsten verwendeten Fragmente sind dabei XML-Infoset Elemente wie Tags, Attribute oder auch statische Werte. Die Nachrichten und die sich ergebenden statischen Fragmente werden dazu mittels eines eigens implementierten Codegenerators erzeugt. Zur Laufzeit werden die Antworten, abhängig von dem Dienstaufruf und dem Kontext, nach definierten Vorgaben aus statischen und sich dynamisch ändernden Fragmenten zusammengefügt (siehe Abbildung 5.14). Das Zusammenfügen der Fragmente wird in uDPWS durch die *Sending-Machine* implementiert. Das eigentliche Senden der entstehenden, vollständigen

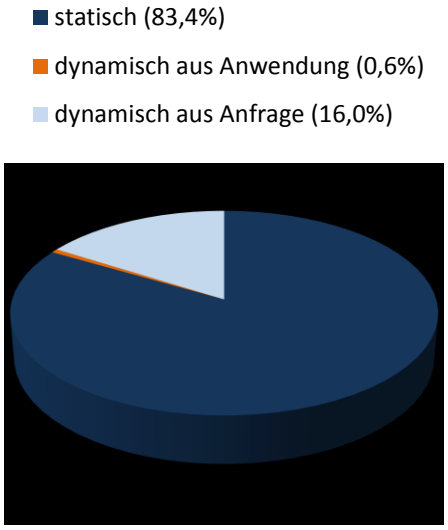

Abbildung 5.13 statischer und dynamischer Anteil von Nachrichten

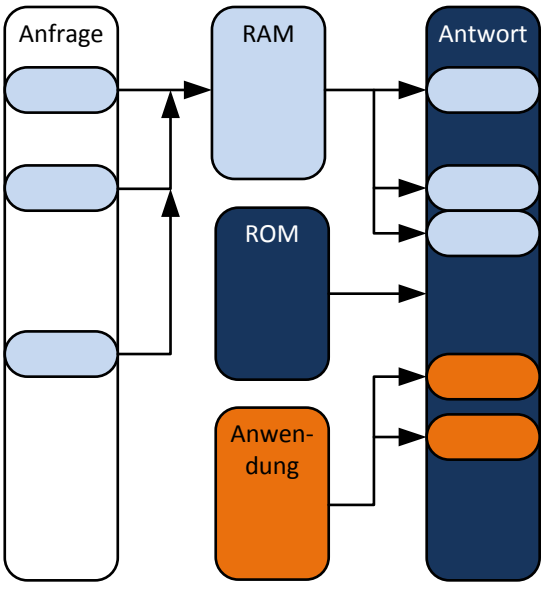

Abbildung 5.14 Generierung von Antworten

Nachricht wird von dem Modul uDPWS-Core implementiert.

Um die Größe der Nachrichtenpuffer und somit des dynamischen Speicherbedarfes zu minimieren, unterstützt uDPWS immer nur eine Anfrage zur gleichen Zeit. Jede Anfrage beginnt mit dem Empfang einer TCP-Verbindungsanfrage (vgl. TCP SYN). Wird die TCP-Verbindungsanfrage von uDPWS abgelehnt oder gar nicht beantwortet, kann die anfragende Gegenstelle dies unter Umständen nicht als temporäre, sondern als vollständige Nicht-Erreichbarkeit interpretieren. Daher wird in uDPWS das *Closed Window* Verfahren eingesetzt. Bei diesem Verfahren wird eine TCP-Verbindungsanfrage (vgl. SYN) für den Fall, dass bereits eine Anfrage bearbeitet wird, mit einer positiven Rückmeldung beantwortet (vgl. SYN ACK). In der Rückmeldung wird dem Anfragenden aber ebenfalls mitgeteilt, dass derzeit keine Daten empfangen werden können. Hierzu wird das *Sliding Window* der TCP-Flusssteuerung in der TCP-Antwort auf den Wert Null genutzt. Dadurch werden vom Anfragenden im Folgenden keine Daten gesendet. Hat uDPWS die vorhergehende Anfrage bearbeitet, so wird an den Client der bereits offenen Verbindung ein *Sliding Window* mit einer Zahl größer als Null gesendet, woraufhin dieser mit der Übermittelung der Daten beginnt.

5.3.2 Speicherbedarf

Da die größten Teile der ausgehenden Nachrichten statisch sind, können die Nachrichtenfragmente im ROM vorgehalten werden und müssen zur Laufzeit nicht in den RAM geladen werden. Dies senkt den erforderlichen Speicherbedarf des RAMs auf ein Minimum, impliziert aber einen erhöhten ROM-Bedarf. Im Hinblick auf den Energiebedarf ist diese Strategie der Minimierung des RAM-Bedarfes vorteilhafter, da der RAM permanent mit Energie versorgt werden muss, um eine Speicherung zu gewährleisten. Der ROM hingegen benötigt nur Energie, wenn Zugriffe darauf erfolgen.

Der größte Teil des dynamischen RAM-Speichers für uDPWS wird durch den Nachrichtenpuffer der eingehenden Anfragen benötigt. Für die dynamischen Anteile der ausgehenden Nachrichten wird der gleiche Puffer verwendet.

Der ROM-Speicherbedarf von uDPWS und dessen Module ist im oberen Teil der Tabelle 5.1 getrennt für die GNU GCC [114] Linker Sections .text[5] und .data[6] dargestellt. Die Werte beziehen sich auf eine anwendungsunabhängige Grundkonfiguration von uDPWS. Die *Common*-Funktionen in Tabelle 5.1 sind allgemeine Funktionen, beispielsweise zum Verarbeiten von Zeichenketten und XML, die keinem Modul exklusiv zugeordnet werden können. Im unteren Teil der Tabelle ist zusätzlich der statische Speicherbedarf für das gesamte System der konkreten Umsetzung des Referenzszenarios aus Kapitel 4.4 dargestellt. Der *Hosting Service* findet sich im oberen und unteren Teil der Tabelle, da dieser in allgemeingültige und anwendungsspezifische Funktionen und Daten unterteilt werden kann. Die Dienst-Module (*Hosted Services, Air Conditioner*) implementieren die konkreten auszuführenden Funktionen und Aktionen der Anwendung und können daher bezüglich des Speicherbedarfes stark variieren. Der auf dem TelosB verfügbare ROM-Speicher ist zu etwa 85% belegt, wobei rund ein Drittel auf uDPWS entfällt.

Die Tabelle 5.2 zeigt den dynamischen Speicherbedarf von uDPWS. Die Werte in Tabelle 5.2 sind

	Programm .text [Byte]	**Nachrichtenfragmente .text [Byte]**	**Initialisierte Daten .data [Byte]**	**Summe [Byte]**
Core	1592	0	90	1682
SOAP + Hosting Service	4278	1935	50	6263
HTTP	940	0	12	952
Sending-Machine	438	0	0	438
Common	871	0	0	871
uDPWS Gesamt	**8119**	**1935**	**152**	**10206**
System/Treiber	1598	0	4	1602
Contiki	26314	0	80	26394
Hosting Service	26	660	0	686
Air Conditioner	154	1818	6	1978
Gesamt	**36211**	**4413**	**242**	**40866**

Tabelle 5.1 ROM Speicherbedarf von uDPWS

[5] Programmcode und konstante Daten im ROM
[6] Initialisierte Daten im ROM, werden beim Starten durch Bootloader in RAM geladen

getrennt für die Linker Sections .bss[7] und .data[8] dargestellt. Wie auch Tabelle 5.1 zeigt die Tabelle 5.2 im oberen Teil den anwendungsunabhängigen Speicherbedarf und im unteren Teil die Werte der konkreten Umsetzung des Referenzszenarios. Der größte Teil des dynamischen Speicherbedarfs wird zum einen durch Contiki selbst und zum anderen durch den Puffer für die eingehenden Nachrichten verursacht. Der Puffer von uDPWS kann grundsätzlich angepasst und somit weiter verkleinert werden. Eine Verringerung unterhalb der für IPv6 mindestens erforderlichen MTU von 1280 Byte ist nicht sinnvoll.

Zusammengefasst kann festgestellt werden, dass DPWS auf der Zielplattform TelosB umgesetzt werden kann, die über 48 kB ROM und 10 kB RAM verfügt und somit zur Klasse 1 gehört. Etwa 7 kB des ROM-Speichers sind in der gewählten Konfiguration ungenutzt. Der RAM-Speicher ist nahezu erschöpft. Wie bei SOAP-basierten Web Services üblich, wird mit der gewählten Implementierungsstrategie der größte Teil des benötigten Quellcodes automatisch aus der WSDL generiert. Für uDPWS gehören hierzu die individuellen Nachrichtenfragmente. Lediglich die Verarbeitung der Nutzdaten der Anwendung (vgl. SOAP-Body) ist durch den eigens entwickelten

	Daten + Puffer .bss [Byte]	**Daten .data [Byte]**	**Summe [Byte]**
Core	2534	90	2624
SOAP + Hosting Service	168	50	218
HTTP	5	12	17
Sending-Machine	0	0	0
Common	0	0	0
uDPWS Gesamt	**2707**	**152**	**2859**
System/Treiber	0	4	4
Contiki	7034	80	7114
Hosting Service	0	0	0
Air Conditioner	0	6	6
Gesamt	**9741**	**242**	**9983**

Tabelle 5.2 RAM Speicherbedarf von uDPWS

[7] Nullinitialisierte Werte
[8] Initialisierte Daten im ROM, werden beim Starten durch Bootloader in RAM geladen

Implementierung	ROM [kB]	RAM [kB]	XML-Parser	Protokolle
uDPWS	10,2	2,9	ja	HTTP
				SOAP
				DPWS[9]
WS4D-gSOAP	488,5	208,5	ja	HTTP
				SOAP
				DPWS[9]
Erbium	8,5	1,5	nein	CoAP
Yazar et al.	11,6	n.a.	ja	HTTP
				SOAP

Tabelle 5.3 Vergleich uDPWS mit anderen Implementierungen

XML-Parser anders gestaltet. Die unteren Schichten sowie die Behandlung der Steuerinformationen der WS-* Spezifikationen wird wie üblich vor der Anwendung verborgen.

Zur weiteren Bewertung des Speicherbedarfes zeigt Tabelle 5.3 die Gegenüberstellung des RAM- und ROM-Bedarfes sowie der jeweils enthaltenen Protokolle von uDPWS und drei weiteren Implementierungen. WS4D-gSOAP [106] ist ein ebenfalls in C implementierter DPWS-Stack der WS4D-Initiative für eingebettete Systeme. Obgleich WS4D-gSOAP auf dem effizienten gSOAP-Toolkit [108] basiert, ist der Speicherbedarf verglichen mit uDPWS immens. Der Vorteil von WS4D-gSOAP liegt eher in der performanten Bearbeitung von Anfragen. Die beiden anderen ausgewählten Implementierungen hingegen sind, wie auch uDPWS, direkt vor dem Hintergrund von stark ressourcenlimitierter Zielplattformen entstanden. Erbium ist eine Implementierung von CoAP für Contiki (siehe [115]). Yazar et al. haben in [116] eine Implementierung von nativen SOAP Web Services vorgestellt, die nur triviale Anfragen und nicht wie uDPWS ein komplexeres Web Services Profil unterstützt. Im Vergleich zur Implementierung von Yazar et al. benötigt uDPWS dennoch weniger ROM. Im Vergleich mit Erbium benötigt uDPWS etwas mehr ROM, was durch den zusätzlichen XML-Parser von uDPWS erklärt werden kann. Behandlung von Nutzdaten und Datenformaten wie XML ist kein Bestandteil von Erbium. Weiterhin setzt Erbium im Vergleich zu DPWS mit CoAP ein vergleichsweise einfaches Protokoll um.

[9] Inklusive der WS-* Protokolle WS-Addressing, WS-Discovery, WS-Transfer, WS-MetadataExchange und WS-Eventing

Zum Vergleich mit anderen XML-verarbeitenden Implementierungen kann uXMPP herangezogen werden, das Hornsby et al. in [117] beschreiben. uXMPP ist eine Implementierung des Instant Messaging Protokolls XMPP [118] (ehemals Jabber) für Contiki. Hornsby et al. geben für den ROM-Bedarf aber lediglich den Gesamtwert des Systems an und nicht, welche Komponenten des Betriebssystems Contiki genutzt werden bzw. wie groß der Anteil von uXMPP am Speicherbedarf ist. Mit den von Hornsby et al. angegebenen 48.451 Byte ist der statische Speicher des TelosB fasst vollständig ausgelastet.

5.3.3 Laufzeitmessungen

Da uDPWS die Speicheranforderungen der TelosB-Plattform erfüllt, können somit erstmals dedizierte Laufzeitmessungen zur Evaluierung von DPWS in 6LoWPANs vorgenommen werden. Der Datendurchsatz bezüglich der Nutzdaten sowie der Energieverbrauch und die Lebensdauer eines 6LoWPANs sind nicht nur durch die zugrundeliegende Anwendung und die Protokolle auf Anwendungsschicht bestimmt. Weitere essentielle Faktoren sind unter anderem Komplexitäten und Nachrichtenaufkommen auf den unteren Schichten zur Konfiguration und für die Verwaltung des Netzwerkes (z.B. Routenverwaltung, IPv6 Neighbor Discovery, Netzwerktopologie). Durch die voneinander getrennten und entkoppelten Protokollschichten sind diese Mechanismen und Faktoren zu großen Teilen aber unabhängig von der konkreten Anwendung und den Anwendungsschichtprotokollen. Folglich wird auf weitere Betrachtungen dieser Faktoren hinsichtlich der Leistungsfähigkeit verzichtet. Aus dem gleichen Grund wird als Versuchsaufbau ein einfaches Single-Hop Szenario gewählt, bei dem zwischen dem Client und dem Dienst eine direkte Kommunikation möglich ist (siehe Abbildung 5.15). In einer Multi-Hop Topologie sind deutlich komplexere Routingmechanismen nötig, wodurch die verfügbare Bandbreite sinkt. Weiterführende Betrachtungen bezüglich der Leistungsfähigkeit von 6LoWPAN Multi-Hop Netzwerken sind in [119] beschrieben.

Auf der Zugriffsschicht kommt ein sehr einfaches Zugriffsverfahren zum Einsatz, da dieses ebenfalls einen erheblichen Einfluss auf die Laufzeiten bei der Übertragung hat [115]. Das

Abbildung 5.15 Schematischer Messaufbau Umlaufzeit von uDPWS

Nachricht	**Probe + Probe Match**		**Resolve + Resolve Match**		**Dienst**
Binding	**HTTP**	**UDP**	**HTTP**	**UDP**	**HTTP**
SOAP-Anfrage [Byte]	645	645	774	774	547
SOAP-Antwort [Byte]	1125	1101	1015	1015	638
Binding-Anfrage [Byte]	76	0	76	0	76
Binding-Antwort [Byte]	92	0	92	0	91
Summe [Byte]	**1938**	**1746**	**1957**	**1789**	**1352**
Umlaufzeit [ms]	**582,19**	**399,2**	**579,78**	**409,67**	**450,24**
Verarbeitungszeit [ms]	**12,82**	**11,82**	**13,69**	**12,05**	**8,94**

Tabelle 5.4 Laufzeitmessungen uDPWS

gewählte Zugriffsverfahren unterstützt keine stromsparenden Schlafzustände. Weiterhin wird ein TCP-Window von 1280 Byte verwendet. Die im Folgenden dargestellten Aufrufe müssen somit nicht auf Transportschicht fragmentiert werden und werden in einem TCP-Segment übertragen.

In Tabelle 5.4 sind Laufzeitmessungen für drei exemplarische Aufrufe und deren Antworten aus dem Referenzszenario dargestellt. Die Discovery-Aufrufe *Probe* und *Resolve* können sowohl Unicast mittels des HTTP-Binding als auch Multicast mittels des UDP-Bindings übertragen werden. Der Dienstaufruf erfolgt nur Unicast mittels des HTTP-Bindings. Die Tabelle enthält im oberen Teil die versendeten Bytes der SOAP-Dokumente und des jeweiligen Bindings. Der untere Teil der Tabelle 5.4 zeigt die Summe der insgesamt versendeten Bytes sowie die vollständige Umlaufzeit (engl. Round Trip Time) jeweils einer Anfrage inklusive Antwort. Die internen Verarbeitungszeiten von uDPWS, die den Differenzen aus Zeitpunkt des Eintreffens der Anfragen bis zum Aussenden der Antworten auf dem TelosB entsprechen, sind ebenfalls in Tabelle 5.4 in der untersten Zeile dargestellt.

Auch ohne jeden Vergleich mit anderen Ansätzen und Implementierungen können aus den Werten zwei grundlegende Schlussfolgerungen gezogen werden:

(1) Trotz der Verarbeitung von in nativem Unicode kodierten XML-Dokumenten ist die Verarbeitungszeit auf der Zielplattform vernachlässigbar klein gegenüber der insgesamt benötigten Umlaufzeit. Das Paradigma von drahtlosen Sensornetzwerken, dass mit Bezug auf den Energieverbrauch die Verarbeitung

selbst gegenüber dem Senden bzw. Empfangen vernachlässigt werden kann, gilt somit auch für SOAP Web Services in 6LoWPAN-basierten Netzwerken.

(2) Die unverhältnismäßig hohen Umlaufzeiten resultieren zum einen aus den Nachrichtengrößen (Vergleich der über 600 Byte größeren Aufrufe Probe bzw. Resolve mit der Dienstnutzung) sowie zum anderen aus dem TCP-basierten HTTP-Binding für den Transport der SOAP-Dokumente (Vergleich der Aufrufe mittels HTTP bzw. UDP). Für beide Schwerpunkte, die Nachrichtenkodierung und den Transport, werden daher in den beiden nachfolgenden Kapiteln geeignetere Ansätze vorgestellt.

5.4 Zwischenfazit

DPWS und die damit einhergehenden WS-* Spezifikationen sind durch einen hohen Grad an Flexibilität, Modularität und Erweiterbarkeit gekennzeichnet. Die Entkopplung der Schnittstellenmechanismen, der Nachrichten- bzw. Dokumentendarstellungsform und der Nachrichtenübertragung ermöglicht es, diese drei Komponenten getrennt voneinander zu untersuchen und weiterzuentwickeln.

In diesem Kapitel wurden vorrangig neuartige Schnittstellenmechanismen spezifiziert, die notwendig sind, damit DPWS auch in stark ressourcenlimitierten Umgebungen eingesetzt werden kann. Konkret beinhalten die konzipierten Ansätze ein effizienteres Discovery sowie feingranulares Eventing. Das verbesserte Discovery erhöht die Effizienz vorrangig in dynamischen Szenarien und Infrastrukturen mit sich oft ändernden und unbekannten Geräten und Clients. Das weiterentwickelte Eventing kann die Leistungsfähigkeit des Protokolls vorrangig in Szenarien erhöhen, in denen Sensorwerte überwacht werden müssen (vgl. Schwellwertüberwachung).

Insgesamt sind die benötigten Kernfunktionalitäten Gruppenkommunikation und Transportsicherung als besonders kritisch einzuschätzen.

Gruppenkommunikation mittels IP-Multicast ist zwar prinzipiell möglich, allerdings skalieren die existierenden Lösungen mit zunehmender Größe des 6LoWPANs und der damit ansteigenden Anzahl von Endpunkten und physikalischen Hops nicht. Gruppenkommunikation in 6LoWPANs stellt eine sehr komplexe Problemstellung dar, die Betrachtungen der Übertragungsschicht sowie der Routing- und Verwaltungsprotokolle erfordert. Zum Zeitpunkt des Entstehens dieser Arbeit

werden bei der IETF mögliche Lösungen für eine effizientere Form der Gruppenkommunikation entwickelt. Ob und wann sich spezifische Lösungen durchsetzen und diese standardisiert werden, ist bislang nicht abzusehen. Ist eine Gruppenkommunikation in einer spezifischen Netzwerkinfrastruktur nicht möglich, müssen vor allem für das Suchen und Finden von Diensten im größeren Netzwerk zentralere Entitäten, wie im Beispiel von DPWS ein Discovery Proxy, eingesetzt werden.

Weiterhin existiert derzeit keine ganzheitliche Lösung, um eine Ende-zu-Ende Transportsicherung zu ermöglichen. Existierende Lösungen, wie beispielsweise SSL bzw. TLS, sind ohne Anpassungen in ihrer nativen Form aufgrund der Ressourcenbeschränkungen der Zielplattformen nicht anwendbar. Auch hier existieren innerhalb der IETF Bemühungen hinsichtlich einer ganzheitlichen Lösung, die unabhängig von dem konkreten Anwendungsprotokoll eingesetzt werden kann.

Die beiden genannten Funktionen sind unabhängig von DPWS und bedürfen umfassender und ganzheitlicher Betrachtungen, die aufgrund des Umfangs nicht Bestandteil dieser Arbeit sind.

Mittels des implementierten und in diesem Kapitel vorgestellten weltweit ersten DPWS-Stacks für Zielplattformen von 6LoWPAN-Netzwerken konnte nachgewiesen werden, dass DPWS auch auf stark ressourcenlimitierten Plattformen spezifikationskonform realisiert werden kann. Der Speicherbedarf der Implementierung ist dabei ähnlich zu dem von speziell für diese Plattformen entwickelten Protokollen. Der entstandene Stack setzt dabei allerdings ein weitaus komplexeres und höherwertiges Protokoll um. Die evaluierten Laufzeitmessungen mit Hardwareplattformen und Softwarelösungen, die dem aktuellen Stand der Technik entsprechen, ergeben aber unzweckmäßig hohe Umlaufzeiten für natives DPWS. Diese resultieren zum einen aus der Art der Daten- bzw. Dokumentenübertragung und zum anderen aus der Darstellungsform der Nachrichten.

Im nachfolgenden Kapitel 6 wird eine Lösung aufgezeigt, wie die zu übertragende Nachrichtengröße auf ein Minimum reduziert wird. Im darauf folgenden Kapitel 7 wird ein Transportmechanismus vorgestellt, durch den die Nachrichten deutlich effizienter als mit den bislang existierenden Lösungen übertragen werden können. Beide aufgezeigten Ansätze genügen den in Kapitel 4 definierten Kriterien und Anforderungen und sind somit mit der WS-Architecture kompatibel.

6 Verringerung der Nachrichtengröße und Komplexität von DPWS

Inhaltsübersicht

Das Geräteprofil DPWS stützt sich auf SOAP als Nachrichtenformat. SOAP nutzt zur Datenrepräsentation die Auszeichnungssprache XML, die sich vor allem in heterogenen Umgebungen durchgesetzt hat, da sie von Plattform- und Programmiersprachenspezifika unabhängig ist.

Der stärkste Kritikpunkt von XML bezieht sich auf den großen Overhead, der in keinem verträglichen Verhältnis zu den eigentlichen Nutzdaten steht. Dieser Overhead wird vornehmlich durch die Kodierung mit dem Unicode-Zeichensatz verursacht. Dabei wird jeder Strukturbezeichner (vgl. Informationsträger, z.B. Tags und Attribute) und jeder Wert als Zeichenkette kodiert. Daher wird diese Darstellungsform von XML auch als menschenlesbar bezeichnet. Werden die Bezeichner nicht optimal gewählt, entsteht ein Nachteil im Vergleich zu statischeren und rein binären Darstellungsformen. Weiterhin erfordert diese Darstellungsform die Typenumwandlung der Werte, da durch die Kodierung als Zeichenkette nicht zwischen einzelnen Datentypen unterschieden werden kann.

Die zugrundeliegenden WS-Spezifikationen nutzen XML-Namensräume (engl. Namespaces), um eine Überlappung von Bezeichnungen innerhalb eines Dokuments zu vermeiden. Die XML-Namensräume werden durch Uniform Resource Identifier (URI) referenziert. Obwohl eine maximal zulässige Länge der URIs nicht definiert ist, hat sich in der Praxis eine typische Länge zwischen 40 und 100 Byte durchgesetzt. Da DPWS mehrere WS-Spezifikationen nutzt, nimmt

alleine die Kodierung der genutzten XML-Namensräume in einer Nachricht oft einen verhältnismäßig großen Anteil der Gesamtnachricht ein.

Alle geschilderten Effekte haben einen umso größeren Einfluss auf das Verhältnis von Nutzdaten zu Nachrichtengröße, je einfacher die Dienste und umso einfacher die zu übermittelnden Werte sind. In typischen Szenarien von 6LoWPANs, in denen nur einfache Zahlenwerte übertragen werden, ist somit der Overhead maximal.

Durch das SOAP Binding Framework [29] wird die Serialisierung von SOAP-Dokumenten als valides XML-Infoset gefordert. Dadurch wird eine Entkopplung von SOAP zu konkreten Programmiersprachen, Betriebssystemen und deren Datenstrukturen erreicht. Das SOAP Binding Framework fordert aber keine spezifische Kodierung. Die Dokumente können somit eine von dem weit verbreiteten Unicode-XML abweichende Darstellung haben. Dennoch fordert im Gegensatz zum SOAP Binding Framework die DPWS-Spezifikation, dass ein Dienst mindestens Unicode-kodierte SOAP-Dokumente unterstützt. Dies ist insofern kein Widerspruch, als dass weder die DPWS-Spezifikation noch die im Rahmen von DPWS referenzierten Spezifikationen die Verwendung von weiteren Kodierungen verbieten.

Eine effizientere Darstellungsform für SOAP muss folglich die Serialisierung entsprechend XML-Infoset unterstützen. Nur dadurch bleibt das Format mit der WS-Architecture und weiterführenden Spezifikationen wie z.B. XML-Schema und WSDL konform. Auch wird dadurch die Kompatibilität mit existierenden Werkzeugen wie XML-Parsern sichergestellt. Allerdings muss die Kodierung der Informationsträger wie z.B. Tags, Attribute, Namensräume und wenn möglich der Werte effizienter ausgestaltet werden, als es mit Unicode-Zeichenketten möglich ist. Die notwendigen Anpassungen beziehen sich somit direkt auf das atomare Element der Nachricht entsprechend der WS-Architecture und nur indirekt auf DPWS-spezifische Aspekte.

In diesem Kapitel wird eine solche effizientere Form der Kodierung vorgestellt. Das hohe Maß an Interoperabilität, welches durch die Kodierung mittels Unicode erreicht wird, wird durch das neue Konzept nicht beeinträchtigt. Dennoch weisen die Leistungsfähigkeit und Effizienz ein Maß auf, wie es für proprietäre, binäre Kodierungen üblich ist. Hierzu werden im Folgenden zunächst verschiedene Ansätze vorgestellt, wie Zeichenketten komprimiert und XML-Dokumente kodiert werden können. Im Anschluss folgt eine Bewertung hinsichtlich der Effizienz der existierenden

Algorithmen und Formate, was die Grundlage für die Konzeption einer zweckmäßigen Lösung bildet. Zuletzt wird eine konkrete Implementierung des aufgezeigten Ansatzes vorgestellt.

6.1 Existierende Datenformate und Kompressionsverfahren

Die existierenden Datenformate und Kompressionsverfahren können in die Klassen der generische Kompressionen und XML-spezifische Kompressionen bzw. Formate unterteilt werden. Die Klasse der generischen Kompressionen behandelt die umzuwandelnden Nachrichten als rein binäre Datenströme. Die Form der Serialisierung, die Syntax sowie die Semantik sind für generische Kompressionsverfahren nicht relevant. Dadurch sind diese auf beliebige Datenströme anwendbar. Im Kern werden in den entstehenden Datenformaten Wiederholungen von Bytefolgen vermieden. Dazu werden identische Bytefolgen innerhalb des ursprünglichen Datenstroms identifiziert. Diese Bytefolgen sind im resultierenden Datenstrom nur noch einmal vorhanden und werden bei jedem weiteren Auftreten durch einen binären Schlüsselwert referenziert (siehe Abbildung 6.1).

Die Klasse der XML-spezifischen Kompressionen behandelt den Datenstrom nicht als reinen Text, sondern extrahiert den Informationsgehalt der Dokumente und stellt diesen effizienter dar. Beispielsweise beinhalten Kommentare, Zeilenumbrüche und Leerzeichen in XML-Dokumenten oft keine Informationen und können gänzlich vermieden werden. Auch die Informationsträger der XML-Strukturen selbst werden beachtet. Die einfachen XML-spezifischen Kompressionsverfahren nutzen zur Vermeidung von Redundanzen in den Strukturen ähnliche Strategien wie die generischen Kompressoren. Dazu verwenden diese einfachen Kompressionsverfahren bei mehrfachem Auftreten von XML-Informationsträgern und Werten ebenfalls Referenzen. Das hierfür notwendige Vokabular wird während der Verarbeitung des eingehenden Datenstroms generiert und beinhaltet z.B. konkrete Bezeichner für Informationsträger und nicht reine binäre Bytefolgen wie bei den generischen Kompressoren.

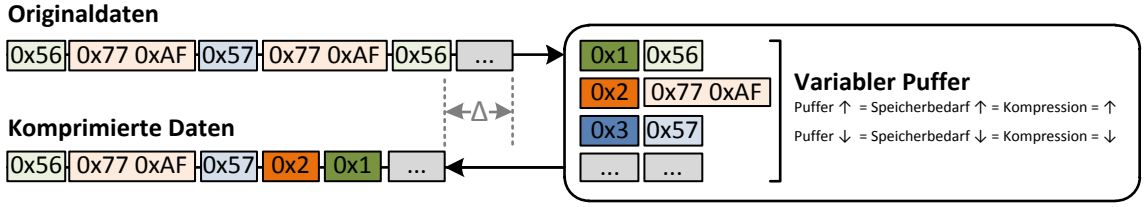

Abbildung 6.1 Schematische Funktionsweise generischer Kompressoren

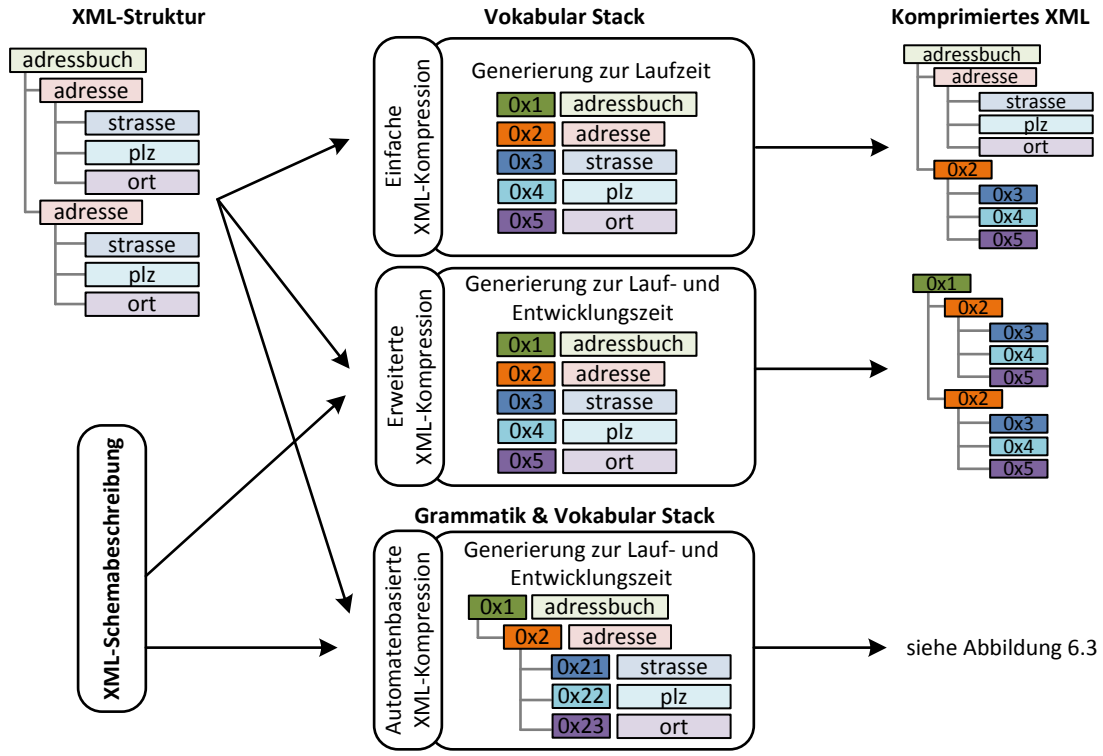

Abbildung 6.2 Funktionsweise XML-Kompressoren schematisch

Eine erweiterte Form der XML-spezifischen Kompression bildet die Möglichkeit, das Vokabular bereits vor der Verarbeitung der Nachrichten zu generieren. Durch das bekannte Vokabular müssen die Informationsträger gar nicht mehr als Zeichenketten im Datenstrom kodiert sein, sondern werden sofort beim ersten Auftreten durch binäre Schlüssel referenziert. Für die Generierung des Vokabulars werden im Bereich der W3C Web Services meist Document Type Definitions (DTD) oder XML-Schemabeschreibungen genutzt, wobei die Unterstützung von XML-Schema durch die WS-Architecture verbindlich vorgeschrieben ist.

Die nächste Weiterentwicklung der XML-spezifischen Kompressionsverfahren beachtet neben einem vor der Verarbeitung bekannten Vokabular ebenfalls die konkrete Grammatik der XML-Dokumente. Durch die Nutzung eines bekannten Vokabulars und einer bekannten Grammatik kann bereits vor der Verarbeitung ein abstrakter Automat generiert werden. Die Zustände des Automaten bilden die XML-Informationsträger, deren Bezeichner und zum Teil auch inhaltliche Werte. Durch die Grammatik sind für jeden Zustand mögliche Übergange zum nachfolgenden Zustand bekannt. Die zu verarbeitenden XML-Strukturen von SOAP Web Services können sowohl bekannte Elemente, aber auch anwendungsabhängige unbekannte Elemente,

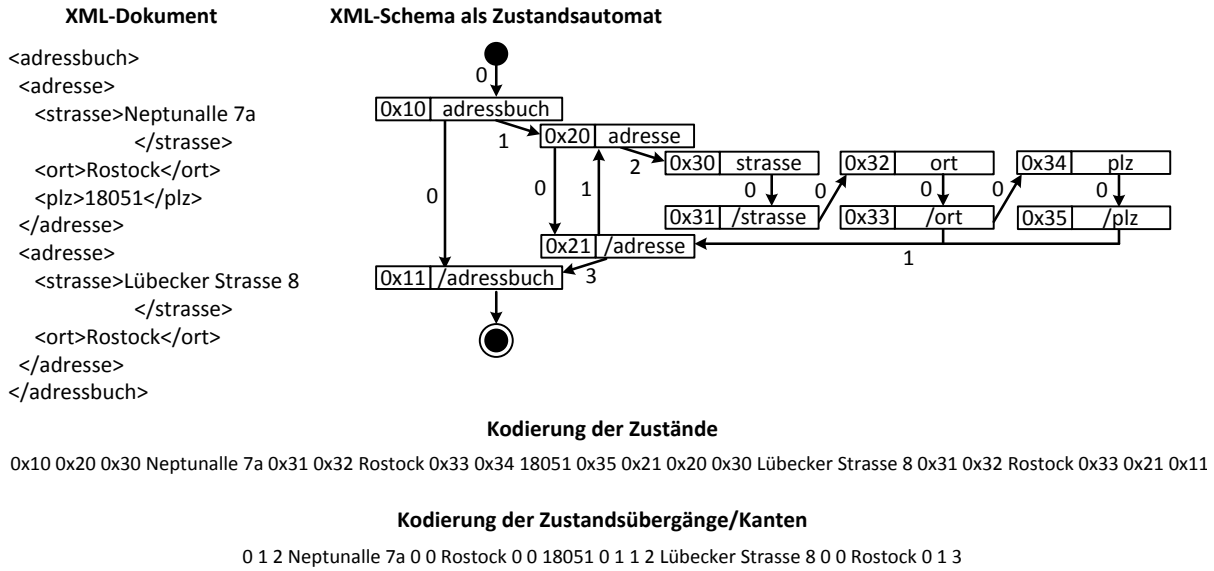

Abbildung 6.3 Kodierungsstrategien für Zustände automatenbasierter XML-Kompressoren

enthalten. Für solche Verfahren werden meist Kellerautomaten verwendet (siehe [120]). Für die Generierung der Automaten werden ebenfalls XML-Schema-Dokumente genutzt.

Eine schematische, stark vereinfachte Darstellung der Funktionsweisen einfacher, erweiterter und automatenbasierter XML-Kompressoren findet sich in Abbildung 6.2.

Ein Vorteil der automatenbasierten Verfahren liegt in der Möglichkeit, unterschiedliche Strategien zur Kodierung der Zustände zu nutzen Zwei mögliche Strategien sind in Abbildung 6.3 dargestellt. Die grafische Darstellung zeigt lediglich zwei Ansätze schematisch auf und beinhaltet keine Wertung über deren Effizienz. Im einfachsten Fall werden im Datenstrom die Zustände direkt kodiert. Der vollständige Automat besteht aber meist aus einer Vielzahl von möglichen Zuständen. Für einen konkreten Zustand existiert hingegen nur eine geringe Anzahl von möglichen Übergängen zum nächsten Zustand. Daher ist es effizienter, im Datenstrom nicht den konkreten Zustand, sondern nur den Übergang zum jeweils nächsten Zustand (vgl. Kanten des Automaten) zu kodieren. Dieses Vorgehen benötigt meist weniger Bit zur Kodierung, erfordert es aber, den konkreten Kontext (z.B. die aktuelle Position im XML-Dokument) während der Verarbeitung der Nachricht zu verwalten. Ohne die Verarbeitung der Nachricht von Beginn an kann ein so kodiertes Element unter Umständen keinem konkreten Zustand bzw. Element zugeordnet werden.

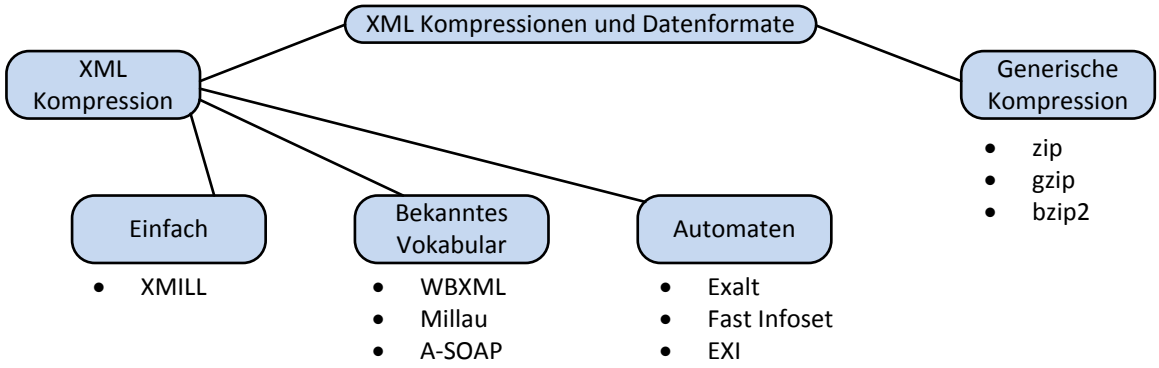

Abbildung 6.4 Einteilung Kompressionsverfahren und Datenformate

In Abbildung 6.4 ist die Unterteilung existierender Datenformate und Kompressionsverfahren in Klassen inkl. einiger typischer Vertreter grafisch veranschaulicht. Nicht alle Formate und Verfahren können eindeutig einer bestimmen Klasse zugeteilt werden. Beispielsweise können viele der automatenbasierten Verfahren, ebenfalls ohne die Verwendung einer bekannten Grammatik und eines bekannten Vokabulars genutzt werden. Dadurch entsprechen diese in der Funktionsweise den einfachen XML-Kompressoren. Im Folgenden werden die relevantesten existierenden Darstellungsformate und Kompressionsmöglichkeiten vorgestellt und diskutiert.

Werner hat sich in seiner Dissertation [121] ebenfalls mit der Kompression in WS-Umgebungen beschäftigt. Die Thematik der Datenkompression wurde von Werner in [121] vor allem aus informationstheoretischer Sicht ausführlich beleuchtet. Daher werden im Folgenden die existierenden Kompressoren und deren Besonderheiten nur knapp dargestellt. Die Ergebnisse der Arbeit von Werner fließen in die nachfolgenden Betrachtungen mit ein. Größter Unterschied zwischen den Betrachtungen von Werner und dieser Arbeit ist das Anwendungsgebiet. Während in Werners Anwendungsgebieten vor allem die Bandbreite limitiert ist, müssen die Lösungen dieser Arbeit allen Anforderungen von 6LoWPANs (Bandbreite, Speicher, Rechenleistung, Energie...) genügen und sich zugleich in DPWS, die WS-Architecture und die in Kapitel 4 konzipierte Infrastruktur einbinden lassen.

6.1.1 Generische Kompressoren

Die Klasse der generischen Kompressoren wird zum einen genutzt, um kostenintensiven Speicherplatz auf Systemen einzusparen. Darüber hinaus werden generische Kompressoren ebenfalls in verteilten Anwendungen genutzt, um die Datenmenge während der Übermittlung zu verringern. Einen Überblick über die Klasse der generischen Kompressoren bietet unter anderem

Sayood in [122]. Die bekanntesten Vertreter sind gzip [123], bzip2, zip, rar und arj. Von diesen genannten Verfahren ist lediglich gzip in einer Spezifikation festgehalten. Die anderen Kompressoren stellen vor allem Implementierungen dar, die durch den Einsatz in Betriebssystemen eine größere Verbreitung erfahren haben.

Vor allem der gzip-Algorithmus wird in heutigen Browser-Interaktionen mit Web Servern oft genutzt. Bei solchen Interaktionen werden vornehmlich HTML-Dateien übertragen. Die Auszeichnungssprache HTML hat ein begrenztes und wohl definiertes Vokabular. Die entstehenden Datenströme enthalten zahlreiche sich wiederholende Zeichenketten, wodurch generische Kompressoren wie gzip sehr gute Kompressionsraten erreichen. Die verwendeten Zielplattformen sind dabei generell nicht als ressourcenschwach zu bezeichnen. Selbst mobile Plattformen, wie z.B. Mobiltelefone, verfügen über mehrere hundert MB dynamischen Speicher und teilweise über Mehrkern-Prozessoren in der Größenordnung von GHz.

Die generischen Kompressoren beachten den Informationsgehalt und die besonderen Struktureigenschaften von XML-Dokumenten nicht. Optimale Kompressionsraten sind bei generischen Kompressoren nur zu erwarten, wenn im Datenstrom viele Wiederholungen von gleichen Byteabfolgen auftreten. Zwar können generische Kompressoren bezüglich der Ressourcenanforderungen dahingehend angepasst werden, als dass die maximale Größe des Datenstroms, in dem zu referenzierende Elemente enthalten sein können oder die Anzahl referenzierter Elemente selbst, eingeschränkt wird. Bei großen Datenmengen kann so der maximal benötigte Pufferspeicher nach oben begrenzt werden. Bei kleinen Nachrichten haben die entsprechenden Konfigurationen aber oft keinen Einfluss auf den benötigten Speicher oder die Effizienz der Kompression. Die jeweils genutzte Konfiguration wird beispielsweise durch einen abstrakten Kompressionsparameter *C* ausgedrückt. Dennoch sind generische Kompressoren durch die hohen Ressourcenanforderungen für den Einsatz auf stark ressourcenlimitierten Plattformen weniger geeignet.

6.1.2 Adaptive SOAP

Bei vielen Szenarien, die SOAP Web Services nutzen, ist auffällig, dass aufeinander folgende Nachrichten eine große Ähnlichkeit aufweisen. Vor allem für sehr einfache Dienste ist der Anteil der sich ändernden Nutzdaten im Vergleich zu statischen Strukturinformationen besonders gering. Auf dieser Grundlage basiert unter anderem die Implementierung von DPWS, die bereits in

Kapitel 5.3 vorgestellt wurde. Adaptive-SOAP (A-SOAP) [124] beschreibt einen Ansatz, der diesen typischen Overhead bei der SOAP-basierten Kommunikation vermeidet.

Dazu entstehen während der Kommunikation zwischen zwei Endpunkten ein gemeinsames Vokabular und eine gemeinsame Grammatik. Das Vokabular und die Grammatik bilden die inhaltlichen Informationen ab und nicht, wie bei generischen Kompressoren, einfache Bytefolgen im Datenstrom. Die Informationen entsprechen den statischen Anteilen der SOAP-Dokumente, die gegenüber den vorherigen Dienstaufrufen keine geänderten Daten enthalten.

Bekannte Strukturelemente werden im Datenstrom aufgrund des bekannten Vokabulars durch effizientere Kodierungsformen dargestellt (siehe Abbildung 6.5). Die Grammatik und das Vokabular selbst werden beim A-SOAP Ansatz nicht durch abstrakte Beschreibungen, wie zum Beispiel DTDs oder XML-Schemas, generiert. Die Informationen werden während des Eingangs der Nachricht auf sich wiederholende Elemente untersucht und zur Grammatik bzw. dem Vokabular hinzugefügt. Im Gegensatz zu generischen Kompressoren stehen die Wiederholungen dabei nicht nur im Kontext der aktuellen Nachricht. Die Informationen über bekannte Elemente bleiben über mehrere Nachrichten erhalten. Somit wird die Kompressionsrate von A-SOAP über mehrere Nachrichten effizienter. Dies steht im Widerspruch zu der Zustandslosigkeit von

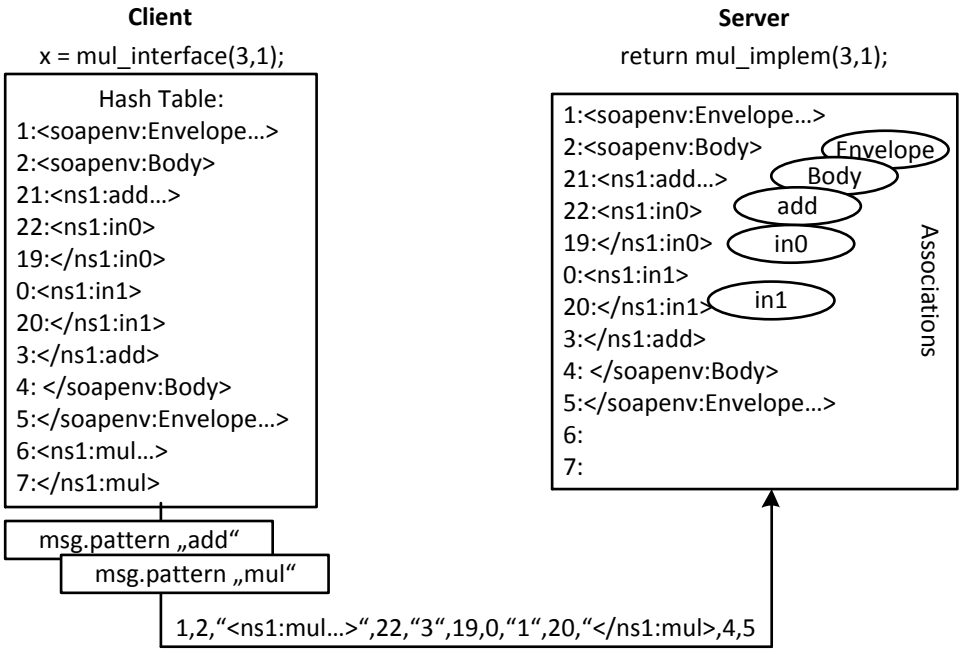

Abbildung 6.5 SOAP-Kompression mit A-SOAP [124]

aufeinander folgenden Dienstaufrufen. Jeder Endpunkt muss für jeden anderen Endpunkt zur Kommunikation ein eigenes Vokabular und eine eigene Grammatik verwalten. Darüber hinaus wurde der A-SOAP Ansatz patentiert und ist somit nur für akademische Zwecke uneingeschränkt nutzbar. Eine Implementierung von A-SOAP ist nicht zugänglich.

6.1.3 Differenzkodierung

Ein ähnlicher Ansatz wie A-SOAP wird von Werner [121] unter dem Begriff der SOAP-Differenzkodierung vorgestellt. Bei der Differenzkodierung wird lediglich der dynamische Anteil der Nachrichten übertragen. Die statischen, nicht zu übertragenden Informationen werden aber im Gegensatz zu A-SOAP bereits vor der eigentlichen Kommunikation ermittelt. Zum einen wird dadurch bereits in der ersten Nachricht die größtmögliche Effizienz bei der Übertragung erreicht. Zum anderen wird die permanente Verwaltung eines Zustandes zu vermieden.

Werner hat dazu eine Lösung entwickelt, mit der eingehende und ausgehende Nachrichten aus der WSDL eines Dienstes generiert werden können. Diese generierten Nachrichtenrümpfe werden als Skeleton bezeichnet. Da die WSDL zur Laufzeit statisch ist, können die Skeletons von den Kommunikationsteilnehmern nach der Generierung vorgehalten werden. Sie können aber ebenfalls für jede Kommunikation neu generiert werden. Zur Übermittlung wird dann die Differenz aus der zu übertragenen Nachricht und dem Skeleton gebildet. Für diesen als *XML-Differencing* bezeichneten Vorgang stehen nach Werner zwei Implementierungen zur Verfügung, die unter den sehr ähnlichen Bezeichnungen *xmldiff* und *diffxml* vertrieben werden. Diese Implementierungen

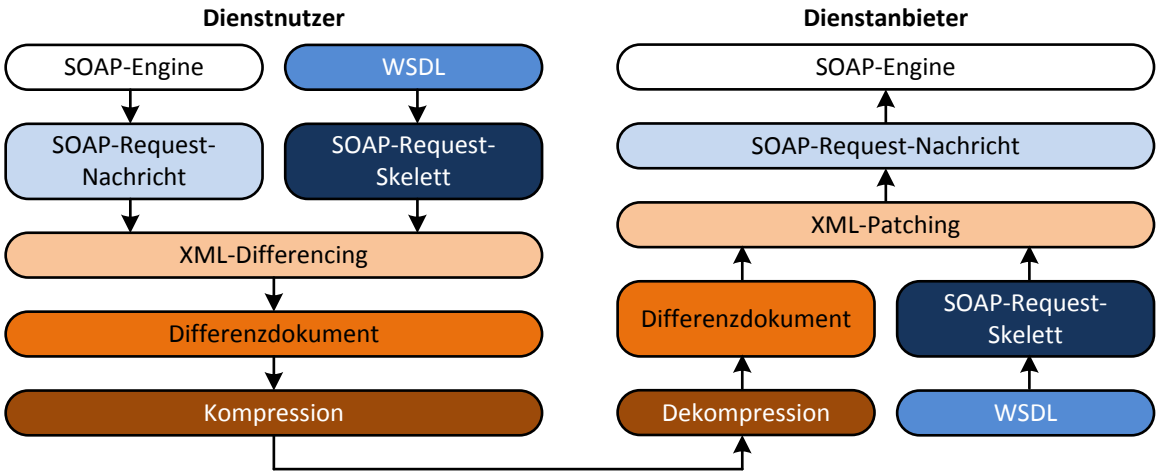

Abbildung 6.6 SOAP-Differenzkodierung nach Werner

können auch zum Generieren der ursprünglichen Nachricht aus der empfangenen Differenz und dem Skeleton genutzt werden. Dieser Vorgang wird als *XML-Patching* bezeichnet. Nach den Untersuchungen von Werner ist die entwickelte Differenzkodierung umso wirksamer, je ähnlicher das Skeleton und die zu übertragene Nachricht sind sowie je größer der Anteil der statischen Nachrichten ist.

Im Kontext der Verwendung der SOAP-Differenzkodierung in stark ressourcenlimitierten Umgebungen ist die entwickelte Lösung nur bedingt geeignet. Die Generierung der Skeletons aus der WSDL ist sehr rechen- und speicheraufwendig. Dieser Vorgang muss daher bereits zur Entwicklungszeit erfolgen und die Skeletons somit statisch eingebunden werden. Die Skeletons können aber nur einer konkreten Nachrichtenstruktur entsprechen. So kann die zur Laufzeit auftretende Struktur der eingehenden Nachrichten von den eingebunden Skeletons deutlich abweichen. Ein Beispiel hierfür ist die Verwendung von XML-Namensräumen. Diese können global für das gesamte XML-Dokument definiert sein oder lokal für einzelne Abschnitte gelten. Ein globaler Gültigkeitsbereich erfordert lediglich einmalig die Festlegung eines Präfixes zur weiteren Verwendung innerhalb von XML-Qualified-Names. Ein lokaler Gültigkeitsbereich hingegen erfordert es im schlechtesten Fall, dass der Namensraum bei jedem lokalen Auftreten in der Nachricht vollständig enthalten sein muss. Der eigentliche Informationsgehalt würde aber bei beiden Varianten identisch bleiben. Die Effizienz der Differenzkodierung ist folglich abhängig von den Kommunikationsteilnehmern und kann vorab nicht bestimmt werden. Zur Laufzeit ist es für das *XML-Differencing* und *XML-Patching* nötig, die übertragene Nachricht mit dem Skeleton zu vergleichen. Dies ist ebenfalls speicher- und rechenaufwendig.

Die Nutzung der WSDL zur Generierung der Skeletons bietet lediglich die Möglichkeit, die Nachrichtenübertragung für einen konkreten Dienst zu optimieren. Dienste sind anwendungsspezifische Umsetzungen konkreter Funktionalitäten. Um auch in stark heterogenen Umgebungen eine effiziente Kommunikation zu ermöglichen, sollten Optimierungen protokollspezifisch und nicht dienstspezifisch anwendbar sein.

6.1.4 WAP Binary XML - WBXML

WAP Binary XML (WBXML) [125] ist ein protokollspezifisches Datenformat, welches bereits 1999 vom W3C für die Datenkompression von XML im Mobilfunkbereich entwickelte wurde. WBXML ist damit einer der ersten standardisierten XML-spezifischen Kompressoren, der die

Struktureigenschaften von XML ausnutzt. WBXML arbeitet nur dann effizient, wenn es auf eine spezielle XML-basierte Beschreibungssprache angewendet wird. WBXML konzentrierte sich im Rahmen der Anwendungsszenarien vor allem auf die Wireless Markup Language (WML) [126], die eine stark reduzierte Form der Auszeichnungssprache HTML darstellt. Grundsätzlich lässt sich der WBXML-Algorithmus aber auch auf andere XML-basierte Sprachen anwenden.

Das Dokument wird beim WBXML-Verfahren auf Elemente untersucht, die in der Grammatik bzw. dem Vokabular der Sprache bekannt sind. Diese bekannten Elemente werden als *Token* bezeichnet und zur Übermittlung binär kodiert. Beiden Empfängern muss die Zuordnung der *Token* zu den Informationsträgern bekannt sein. Hierfür wurde in WBXML kein formaler anwendungs- und sprachenunabhängiger Mechanismus definiert. Für WBXML werden das gesamte Vokabular und die *Tokens* zuvor in der Spezifikation definiert. Dies macht WBXML ungeeignet für SOAP-basierte Protokolle, die unbekannte Nachrichtenstrukturen und frei wählbare Strukturbezeichner haben. Zu diesem Schluss kommt auch eine Untersuchung der Firma Nokia im Rahmen der WBXML-Standarisierung beim W3C, die in einem Positionspapier in [127] veröffentlich wurde.

6.1.5 Millau

Millau ist eine Weiterentwicklung von WBXML durch Giradot und Sundaresan [128]. Eine wesentliche Verbesserung gegenüber WBXML stellt die Behandlung von unbekannten Strukturelementen zur Laufzeit dar. Bei Millau werden die unbekannten und bekannten Strukturelemente voneinander getrennt und in separaten Containern übertragen. Durch die Trennung ergeben sich, vor allem im Container der die bekannten Elemente enthält, verhältnismäßig viele aufeinander folgende gleiche Elemente. Generische Kompressoren können diese Redundanzen effizienter ausnutzen und somit das Format ergänzen, indem diese nachträglich auf den Datenstrom angewendet werden.

6.1.6 XMill

Der XMill-Kompressor [129] wurde bereits 2000 von Liefke und Suciu publiziert und ist der erste XML-spezifische Kompressor, der im Gegensatz zu WBXML und Millau ohne bekanntes Wissen über die Dokumentenstruktur auf XML-Daten angewendet werden kann. Dazu werden Strukturinformationen und Daten in vollständig getrennten Containern im Datenstrom kodiert. Die Zuordnung der Daten zu den Strukturinformationen geschieht dabei über Referenzierung. Dadurch

werden Redundanzen im Daten-Container vollständig vermieden. Liefke und Suciu haben in Experimenten eine ähnliche Kompressionsrate wie der generische bzip2-Kompressor erreicht, wobei die benötigten Systemressourcen mit denen des gzip-Algorithmus vergleichbar sind (siehe [129]). XMill ist aber nicht als Alternative zu den generischen Kompressoren zu verstehen. Vielmehr können solche generischen Kompressoren XMill ergänzen und genutzt werden, um den entstandenen Datenstrom in ein reines Binärformat zu überführen und die resultierende Dokumentgröße weiterhin zu reduzieren.

6.1.7 ASN.1 und Fast Web Services

Die Beschreibungssprache Abstract Syntax Notation One (ASN.1) [130] ist wie auch XML in der Lage, Datentypen und Datenstrukturen abstrakt zu definieren. Die aktuellste Version der ASN.1 Spezifikation ist im Standard X.680ff [130] der International Telecommunication Union (ITU) festgeschrieben. Ziel von ASN.1 ist die kompakte Repräsentation durch eine Schreibweise, die der Backus-Naur-Form (BNF) ähnlich ist. Mit dem ITU Standard X.694 [131] wurde eine Möglichkeit der Überführung von XML-Schema-Definitionen in ASN.1 geschaffen. Die formale ASN.1-Sprache wird zur Übermittlung mittels konkreter Repräsentationen kodiert, die in den ASN.1 Encoding Rules definiert sind.

Im August 2003 wurde von Sandoz et al. ein Artikel publiziert [132], der unter dem Begriff der *Fast Web Services* bekannt ist und beschreibt, wie sich mithilfe der ASN.1-Technologie SOAP Web Services deutlich effizienter realisieren lassen. Durch die Nutzung von XML-Schema-Dokumenten ist ASN.1 damit protokollspezifisch und anwendungsunabhängig auf SOAP-Dokumente anwendbar. Allerdings ist derzeit keine Implementierung der *Fast Web Services* öffentlich zugänglich.

6.1.8 XML-Conscious PPM

Der XML-Conscious PPM Kompressor (XMLPPM), von Cheney in [133] publiziert, ist nicht standardisiert. Eine Open-Source-Implementierung wird in der Programmiersprache C unter der GNU General Public License (GPL) zur Verfügung gestellt. XMLPPM kombiniert generische Textkompression mit SAX-Parsern (Simple API for XML) und dem Prediction by Partial Match (PPM) Algorithmus [134]. Die generischen Kompressoren werden nach dem Binarisieren der Daten genutzt, um im Datenstrom vorhandene Redundanzen zu entfernen. Dadurch erreichte

XMLPPM zur Zeit der Veröffentlichung eine deutlich bessere Kompression als jeder existierende XML-spezifische Kompressor.

6.1.9 Exalt

Wie XMLPMM ist auch Exalt (Experimental XML Archiving Library/Toolkit) [135] eine unter GPL frei zugängliche Implementierung und nicht weiter als Spezifikation publiziert. Exalt ist ebenfalls ein XML-spezifischer Kompressor. XML-Elemente werden beim Auftreten erlernt. Mithilfe der erlernten Elemente werden Automaten zur syntaktischen Kompression generiert. Allerdings fehlt für Exalt ein Mechanismus zur formalen Beschreibung der Automatenstrukturen zur Entwicklungszeit. Diese müssen zur Laufzeit für jedes XML-Dokument neu erlernt und angelegt werden.

6.1.10 Xaust

Die Xaust-Implementierung (XML compression with Automata and a Stack) [136] konstruiert ähnlich zu Exalt Automatenstrukturen. Xaust ist im Gegensatz zu Exalt aber in der Lage, vor dem Verarbeiten des XML-Dokuments DTDs einzulesen und daraus die Automatenstrukturen zu generieren. Messungen anhand von großen Datensätzen in [136] haben gezeigt, dass Xaust eine bessere Kompression als XMLPPM bei geringerem Speicherbedarf und geringerer Rechenlast erreicht. Im Vergleich mit XMill benötigt Xaust allerdings trotz schnellerer Verarbeitungszeiten mehr Systemressourcen. Eine Implementierung von Xaust für Vergleichsmessungen ist nicht verfügbar.

6.1.11 Xebu

Durch Kangasharju et al. wurde mit Xebu [137] erstmals ein XML-Kompressor realisiert, der alle zuvor genannten Strategien und Verfahren ineinander vereint. Nach Kangasharju et al. erreicht Xebu dabei gute bis sehr gute Kompressionsraten bei vergleichsweise geringer Bearbeitungskomplexität. Nachteilig an der Lösung ist allerdings, dass die zugrundeliegende Grammatik aus der Relax NG [138] Sprache generiert wird. Relax NG ist zwar seit 2001 bei der OASIS standardisiert, hat sich aber im Bereich der SOAP Web Services nicht durchgesetzt. Eine mögliche Lösung könnte die Übersetzung von Relax NG in XML-Schema beinhalten. Die einzige bekannte Implementierung von Xebu wurde von Kangasharju et al. selbst veröffentlicht und ist nicht mehr verfügbar.

6.1.12 Fast Infoset

Auch Fast Infoset (FI) ermöglicht eine optimierte, binäre Darstellung von XML. Der Vorteil von Fast Infoset ist die Möglichkeit der konkreten Darstellung von XML-Infoset Dokumenten. Damit ist Fast Infoset eines der wenigen Formate, das neben dem Informationsgehalt der XML-Strukturen auch die konkrete Serialisierungsform berücksichtigt. Die in den vorrangegangenen Abschnitten aufgezeigten Kompressoren beachten diese abstrakte XML-Strukturdefinition gar nicht oder nur rudimentär. Wie ASN.1 ist auch Fast Infoset bei der ITU standardisiert [139].

Zur Repräsentation eines FI-Dokumentes und dessen formaler Beschreibung kann die ASN.1-Notation genutzt werden. Obwohl FI konsistent mit ASN.1-basierten Standards ist, ist FI grundsätzlich davon unabhängig und kann auch entkoppelt eingesetzt werden. FI erreicht im einfachsten Modus aber keine außerordentlich guten Kompressionsraten.

Der FI-Standard sieht daher vor, vor der Nachrichtenverarbeitung Automatenstrukturen zu erzeugen, die bei der Kodierung der Fast Infoset Dokumente genutzt werden können, um bereits bekannte Strukturelemente referenzieren zu können. Das Vokabular und die Grammatik wird im Standard als *Initial Vocabularie* bezeichnet und kann direkt aus XML-Schema-Dokumenten extrahiert werden. In diesem Modus ist Fast Infoset deutlich effizienter und nutzt im Gegensatz zu den anderen hier vorgestellten Lösungen direkt XML-Schema-Definitionen zur Generierung der Automaten.

6.1.13 XGrind

Das von Tolani und Haritsa entwickelte XGrind-Format [140] ist nach Werner [121] sehr ähnlich zum Fast Infoset Format. Der Vorteil von XGrind liegt nach Tolani und Haritsa nicht in der besonders effizienten Kodierung, sondern in der Möglichkeit, direkt die binären Daten verarbeiten zu können. Damit sind bekannte XML-Verarbeitungsmechanismen auf die komprimierten Daten anwendbar.

6.1.14 Efficient XML Interchange

In der Folge des aufkommenden Einsatzes von XML und SOAP hat sich auch das W3C mit der optimierten Darstellung von Web Services beschäftigt. Hierzu wurde 2004 die *XML Binary Characterization Working Group* [141] gegründet. Ziel der Arbeitsgruppe war es unter anderem, verschiedene Anwendungsszenarien zu ermitteln. Auf die XML-Dokumente aus den

Anwendungsszenarien wurden existierende Kompressoren angewendet und die Ergebnisse anhand spezifischer Kriterien bewertet. Die Arbeitsgruppe kam zum Schluss, dass eine deutlich optimalere binäre Repräsentation von XML als die Unicode Kodierung durch die existierenden Verfahren und Formate möglich ist und produktiv eingesetzt werden kann. Im Dezember 2005 wurde daraufhin die *Efficient XML Interchange Working Group (EXI)* beim W3C gegründet. Aufgabe der EXI-Arbeitsgruppe war die Spezifizierung einer effizienten Kodierung von XML-Infoset-basierten Dokumenten. Mit dieser spezielleren Fokussierung auf die konkrete Serialisierung hat auch die EXI-Arbeitsgruppe geeignete Kandidaten unter den existierenden XML-Kompressoren evaluiert. Im Juli 2007 wurde daraufhin von der EXI- Arbeitsgruppe der erste Entwurf für das *Efficient XML Interchange (EXI) Format 1.0* veröffentlicht. Erst im März 2011 erreichte das Dokument den Status der W3C-Recommondation [142] und wurde damit offiziell zur W3C-Spezifikation. Das EXI-Format gehört zur Klasse der automatenbasierten Verfahren.

Bei EXI werden aus der bekannten Grammatik und dem bekannten Vokabular Automaten generiert, die als *EXI-Grammar* bezeichnet werden. Zur Generierung der Automaten werden XML-Schemabeschreibungen genutzt. Das entstehende EXI-Format, welches als *EXI-Stream* bezeichnet wird, enthält als Strukturinformationen lediglich Informationen über die Kanten zwischen den jeweiligen Zuständen der Automaten. Um parallel auch unbekannte Strukturen und Daten in einem Dokument verarbeiten zu können, können zusätzlich neue Automatenbestandteile während der Verarbeitung des *EXI-Streams* gelernt werden. Solche zu lernenden Strukturinformationen nutzen die bekannten Mechanismen zur Referenzierung, bei denen Bezeichner und Inhalte beim ersten Auftreten indiziert werden und im Folgenden nur die Indizes zur Kodierung genutzt werden. Das EXI-Format ist durch den Mechanismus des Erlernens neuer Elemente zur Laufzeit sehr flexibel und kann im einfachsten Fall auch ohne vorab bekannte Grammatik bzw. Vokabeln eingesetzt werden. Dieser Modus wird als *schema-less* bezeichnet. Das Pendant dazu, bei dem Automaten bereits vor der Verarbeitung generiert werden, wird als *schema-informed* Modus bezeichnet.

Das Auftreten von Übergängen (vgl. Kanten) innerhalb des Automaten sowie das Auftreten spezifischer Elementinhalte werden als *EXI-Event* bezeichnet. Ein *EXI-Stream* besteht damit aus einer Reihe *EXI-Events* zuzüglich der Elementinhalte bzw. –werte. Ähnlich wie bei XMill können die *EXI-Events* und die Daten in getrennten Containern im Datenstrom angelegt werden und müssen nicht direkt aufeinander folgend im *EXI-Stream* eingebettet sein (siehe Abbildung 6.7).

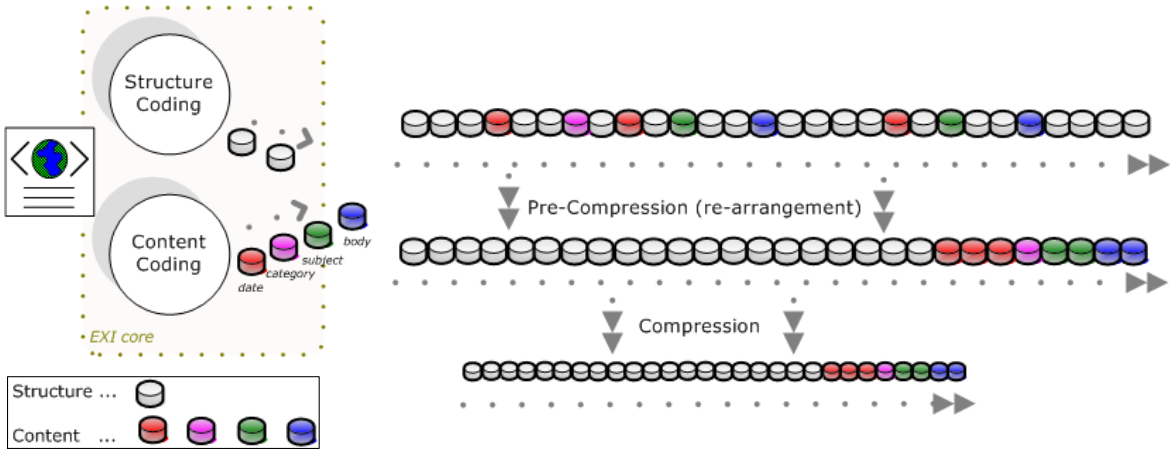

Abbildung 6.7 EXI-Kompression (vgl. [142])

Durch die Trennung können generische Kompressoren nachträglich auf den *EXI-Stream* angewendet werden, was bei größeren *Streams* mit zahlreichen Wiederholungen die Kompression verbessert. Daher ist zwischen dem *compressed* und dem *uncompressed* Modus zu unterscheiden. Im *compressed* Modus wird der bei der IETF in RFC1951 [143] standardisierte Deflate-Algorithmus auf den entstehenden Datenstrom nachträglich angewendet. Im *uncompressed* Modus wird der im EXI-Format kodierte Datenstrom direkt übermittelt.

Die meisten datenverarbeitenden Systeme nutzen als kleinste mögliche Speichereinheit das Byte. Ein Byte besteht dabei aus 8 Bit und ist somit in der Lage 256 verschiedene Werte zu repräsentieren. So wird zum Beispiel in der Programmiersprache C der Datentyp *bool* über ein Byte im Speicher abgebildet, obwohl nur zwischen zwei Werten unterschieden werden kann. Für die meisten möglichen Zustände des EXI-Automaten sind deutlich weniger als 256 nachfolgende *EXI-Events* möglich. Daher weicht das EXI-Format von der Byteorientierung ab und nutzt zur Kodierung lediglich so viele Bit wie nötig sind, um zwischen den möglichen *EXI-Events* unterscheiden zu können. In diesem bitorientierte Modus wird der resultierende *EXI-Stream* als *bit-aligned* bezeichnet. Die Entwickler von EXI haben speziell den *bit-aligned* Modus verfeinert und diesen mit der Huffmann-Kodierung [144] kombiniert. Bei dieser Form der *EXI-Event*-Kodierung werden häufig auftretende *Events* mit weniger Bit repräsentiert als selten auftretende *Events*. Dies führt im Mittel zu einer kompakteren Darstellung. Für Entwicklungszwecke ist dieser *bit-aligned* Modus allerdings sehr unübersichtlich, da im Datenstrom enthaltene Zeichenketten nicht erkennbar sind. Aus diesem Grund kann ein EXI-Stream auch im *byte-aligned* Modus kodiert werden, bei dem immer auf volle Byte aufgefüllt

wird. Der zuvor genannte Deflate-Algorithmus, der im *compressed* Modus nachträglich zur Komprimierung auf den Datenstrom angewendet werden kann, vermeidet Redundanzen in aufeinander folgenden Bytefolgen. Daher impliziert der *compressed* Modus die Nutzung des *byte-aligned* Modus zur Abbildung der *EXI-Events*.

In einem XML-Dokument können Elemente auftreten, die z.B. immer genau ein spezielles Attribut enthalten müssen, um der Grammatik zu entsprechen. Nach dem *EXI-Event* für das Auftreten des Elements ist somit impliziert, dass anschließend das *EXI-Event* für das Auftreten des Attributes folgen muss. Solche eindeutigen Folgen, die keine Auswahl zulassen, werden vom EXI-Parser eigenständig bei der Verarbeitung hinzugefügt und nicht im *EXI-Stream* kodiert und übermittelt. Der *bit-aligned* Modus in Kombination mit der Huffmann-Kodierung [144] arbeitet also dann optimal, wenn für einen Zustand nur genau ein möglicher Übergang existiert.

Es sind Szenarien möglich, in denen von der Grammatik und dem Vokabular abweichende Elemente genutzt werden. Trotz eindeutiger Zuordnung der aufeinander folgenden Elemente in der Grammatik entsteht durch diesen Erweiterungsmechanismus somit immer eine Auswahl zwischen dem bekannten und einem neu zu erlernenden Element. Das zuvor beschriebene Auslassen von *EXI-Events* bei der Kodierung steht also teilweise im Widerspruch mit dem Mechanismus des Erlernens von Elementen des Automaten während der Verarbeitung.

Da nicht alle Anwendungen solche dynamischen bzw. unbekannten Elemente beinhalten, sondern die Kompaktheit des *EXI-Streams* im Vordergrund steht, wird im *schema-informed* Modus weiterhin zwischen dem *strict* und dem *non-strict* Modus unterschieden. Im *strict* Modus dürfen keine Abweichungen von der im Schema vorgegebenen Grammatik auftreten. Im *non-strict* Modus sind solche Abweichungen, zu Lasten der Kompaktheit, zulässig. Diese Abweichungen betreffen unter anderem auch die Verwendung von den in den XML-Schemas definierten Datentypen. In Tabelle 6.1 sind die verschiedenen Modi des EXI-Formates dargestellt.

Durch die Verwendung eines automatenbasierten Verfahrens können Strukturelemente eines XML-Dokuments im EXI-Format deutlich effizienter kodiert werden. Die gleiche Strategie wie für die Strukturelemente wird in EXI ebenfalls auf die Werte bzw. Inhalte der Elemente angewendet, die in der Folge Bestandteil der Automatenstruktur sind. Für die Beschreibung der Werte kann beispielsweise der XML-Schema *restriction*-Mechanismus genutzt werden. Innerhalb einer *restriction*-Definition wird der Wertebereich durch den XML-Schema Mechanismus *enumeration*

	schema-less	schema-informed strict	schema-informed non-strict
compressed	byte-aligned	byte-aligned	byte-aligned
uncompressed	bit und byte-aligned	bit und byte-aligned	bit und byte-aligned
Abweichung von Grammatik	ja	nein	ja

Tabelle 6.1 Modi EXI-Format

eingeschränkt. Da aber in den seltensten Fällen alle Werte eines XML-Dokuments vordefiniert werden können, können parallel zu vordefinierten Werten immer neue Inhalte während der Verarbeitung gelernt werden. Die Unterscheidung zwischen den Modi *strict* und *non-strict* für Strukturelemente und Datentypen gilt somit nicht für eigentliche Werte bzw. Inhalte der Elemente. Die Werte können immer während des Parsens gelernt werden und dynamisch zur Automatenstruktur hinzugefügt werden.

Eine Voraussetzung für die Anwendung des *schema-informed* EXI-Formates ist das Vorhandensein von Beschreibungen der Grammatik und des Vokabulars in Form von XML-Schema-Beschreibungen. Da die EXI-Grammatik aus diesen Schemabeschreibungen generiert wird, müssen allen Kommunikationsendpunkten identische Schemabeschreibungen vorliegen. Alternativ muss in einer anderen Form die EXI-Grammatik ausgetauscht werden. Ist dies nicht möglich, können *EXI-Events* nicht korrekt zugeordnet werden. Die EXI-Spezifikationen definieren keinen obligatorischen Mechanismus für die Aushandlung oder den Austausch der zu verwendenden Schema-Dokumente. Innerhalb des *EXI-Headers*, der einem *EXI-Stream* immer vorangestellt sein muss, kann ein entsprechendes Feld genutzt werden, um die zur Kodierung und Dekodierung des *EXI-Streams* genutzten Schemas anzuzeigen. Die EXI-Spezifikation definiert zum einen dieses Feld als optional. Darüber hinaus wird weder die Syntax noch die Semantik des Feldes vorgeschrieben. Ist das Feld nicht im *EXI-Header* enthalten, so müssen andere externe Mechanismen genutzt werden, damit sich die Kommunikationspartner auf die gleichen Schemabeschreibungen und somit die gleiche Automatenstruktur synchronisieren können.

6.1.15 Xenia

Werner hat in seiner Dissertation [121] eine Lösung entwickelt, die dem EXI-Format konzeptionell sehr ähnlich ist. Der Ansatz wurde von Werner prototypisch in Java unter dem Namen Xenia implementiert. Da die Entstehung der Dissertation von Werner zeitlich vor der Entstehung des EXI-Formates liegt, sollten die Ergebnisse der Dissertation laut Werner in die Arbeit der W3C

EXI-Arbeitsgruppe einfließen. Hinweise hierauf sind in den verfügbaren Dokumenten der W3C EXI-Arbeitsgruppe und der vorangegangenen W3C Binary Characterization Arbeitsgruppe nicht zu finden. Allerdings wurde der von Werner entwickelte Ansatz patentiert und ist nicht als Implementierung zugänglich. Es ist denkbar, dass die Patentierung die Einbeziehung der Ergebnisse in das resultierende EXI-Format verhindert.

6.2 Effizienzuntersuchungen bestehender Lösungen

Die existierenden Kompressionsverfahren und Datenformate basieren jedes für sich auf spezifischen Voraussetzungen und verfolgen jeweils unterschiedliche Strategien, um bestmögliche Kompressionsraten zu erreichen. Einige der entwickelten Lösungen haben lediglich akademischen Charakter und es wurden nie Implementierungen öffentlich zugänglich gemacht. Einige der Lösungen wiederum sind durch Patente geschützt.

Das EXI-Format ist die aktuellste Entwicklung im Bereich der XML-spezifischen. Als besonders vorteilhaft ist einzuschätzen, dass EXI als W3C-Recommondation veröffentlicht ist. Somit stellt EXI eine offene standardisierte Lösung dar, die mit XML-Infoset und ebenfalls mit SOAP kompatibel ist.

Im Rahmen der Standardisierung des EXI-Formates wurden diverse Evaluierungen vorgenommen, um die Leistungsfähigkeit existierender Lösungen mit der des neu entwickelten EXI-Formates zu vergleichen. Die hierfür genutzten Daten und die Ergebnisse der Evaluierungen wurden durch die EXI-Arbeitsgruppe des W3C in [145] veröffentlicht. Bislang nicht untersucht wurde das EXI-Format im Kontext von DPWS in 6LoWPANs. Solche Szenarien unterscheiden sich grundlegend von anderen SOAP-basierten Anwendungen, da die Dienste der Geräte deutlich einfacher sind. Dadurch sind die Nachrichten weniger komplex strukturiert und weniger umfangreich. Anders als in höherwertigen Anwendungen erzielen die Dienste und Daten des 6LoWPANs einen Mehrwert durch die Vernetzung selbst bzw. durch die Vernetzung mit höherwertigen Diensten und nicht durch die Komplexität der angebotenen Dienste. Vor diesem Hintergrund werden im Folgenden die existierenden Formate und Kompressoren mit dem speziellen Fokus der Anwendung auf DPWS in 6LoWPANs evaluiert.

Die zur Verfügung stehende Bandbreite in 6LoWPANs und die Ressourcenbeschränkungen der Zielplattformen erfordern eine minimale Nachrichtengröße und zugleich die Inanspruchnahme

minimaler Systemressourcen zur Verarbeitung. Dazu ist es erforderlich, dass die Formate der XML-Dokumente direkt von den Zielplattformen verarbeitet werden können. Erfordert die Verarbeitung der XML-Strukturen eine Vorverarbeitung, wie das Dekomprimieren in eine zum Parsen geeignete Repräsentation, so ist das Format eher ungeeignet.

Die Systemarchitektur (siehe Kapitel 4) sieht vor, dass alle Optimierungen nur das Subnetzwerk betreffen und andere, bestehende, ressourcenstärkere Netzwerke und Systeme keinen Änderungen unterliegen. Daher muss eine anwendungsunabhängige, verlustfreie und zustandslose Umwandlung des im 6LoWPAN zum Einsatz kommenden Formates in natives Unicode XML möglich sein. Eine optimale Repräsentation ist somit eine Darstellungsform, die sich verlustfrei in Unicode XML dekodieren lässt und zugleich maximale Kompressionsraten gegenüber Unicode XML erreicht. Da sich XML trotz verschiedener Kodierungen interoperabel zeigt, ist ebenfalls die binäre Interoperabilität eine essentielle Anforderung an das Datenformat. Nur so lässt sich die Datenkompression transparent in die konzipierte Architektur aus Kapitel 4 integrieren.

6.2.1 Anwendung von EXI auf DPWS in 6LoWPAN-Netzwerken

Da als Zugriffsschicht in 6LoWPANs energieeffiziente Technologien zum Einsatz kommen, die lediglich sehr kleine Paketgrößen unterhalb der Vermittlungsschicht bereitstellen, muss zur effizienten Übertragung die resultierende Nachrichtengröße unterhalb der PDU der Zugriffsschicht liegen. Für die Eignung von Kompressoren in 6LoWPANs ist somit nicht das Kompressionsverhältnis entscheidend, sondern lediglich die nach der Kompression verbleibende Nachrichtengröße. Für IEEE 802.15.4 bedeutet dies, dass für die Anwendungsschicht lediglich etwa 80-100 Byte zur Verfügung stehen.

In der nachfolgenden Evaluierung können lediglich Kompressoren betrachtet werden, für die entsprechende Implementierungen zugänglich sind. Für die zu untersuchenden Nachrichten wird das aus Kapitel 4.4 bekannte Referenzszenario der Temperaturregelung genutzt. Die Nachrichten des Referenzszenarios wurden erfasst und ausgehend von den erfassten XML-Nachrichten offline (ohne Integration in eine DPWS Implementierung) in das jeweilige Datenformat überführt. Die Ergebnisse der Messungen sind in Tabelle 6.2 dargestellt. Die erste Spalte bezeichnet das jeweilige Format. Die zwei folgenden Spalten stellen die Mittelwerte der Größen über alle Nachrichten jeweils absolut und prozentual dar. Die restlichen vier Spalten zeigen die Größe der jeweils größten

bzw. kleinsten Nachricht aus dem Referenzszenario ebenfalls absolut und prozentual. Die gesamte Tabelle ist nach dem absoluten Mittelwert geordnet.

Trotz gänzlich unterschiedlicher Strategien der Verfahren und Formate erreichen die meisten Lösungen im Mittel nahezu identische Resultate zwischen 50 % und 60 % der Unicode kodierten XML-Nachrichten. Der Grund hierfür ist der sehr einfache Nachrichtenaufbau. Die Schnittstellen basieren auf komplexen, flexiblen und erweiterbaren Protokollen, nutzen aber aufgrund der

Format	Mittelwert		Minimum		Maximum	
	Byte	%	Byte	%	Byte	%
EXI[1]	205,94	25,27	122,00	29,19	416,00	19,91
EXI[2]	263,00	32,27	122,00	29,19	690,00	33,03
EXI[3]	288,11	35,36	132,00	31,58	764,00	36,57
EXI[4]	316,17	38,80	192,00	45,93	639,00	30,59
XMLPPM	425,22	52,18	274,00	65,55	749,00	35,85
gzip (C=9)[7]	425,56	52,22	297,00	71,05	755,00	36,14
gzip (C=1)[7]	437,44	53,68	300,00	71,77	799,00	38,25
Xmill (C=9)[7]	459,39	56,37	303,00	72,49	824,00	39,44
Xmill (C=1)[7]	463,72	56,91	304,00	72,73	852,00	40,79
EXI[5]	469,55	57,62	234,00	55,98	1.146,00	54,86
bzip2 (C=1)[7]	474,78	58,26	315,00	75,36	852,00	40,79
bzip2 (C=9)[7]	474,78	58,26	315,00	75,36	852,00	40,79
EXI[6]	503,06	61,73	247,00	59,09	1.251,00	59,89
Fast Infoset[6]	561,89	68,95	315,00	75,36	1.301,00	62,28
UTF-8 XML	814,89	100,00	418,00	100,00	2.089,00	100,00

[1]schema-informed / compressed

[2]schema-informed / bit-aligned

[3]schema-informed / byte-aligned

[4]schema-less/ compressed

[5]schema-less / bit-aligned

[6]schema- less / byte-aligned

[7]Formatspezifischer Parameter: Verhältnis aus Kompressionsrate zu Komplexität und Ressourcenverbrauch (C=1: schlechte Kompression und geringe Komplexität, C=9: gute Kompression und hohe Komplexität)

Tabelle 6.2 Nachrichtengrößen bei verschiedenen Datenformaten und Kompressoren

Limitierungen für ein konkretes Szenario nur einen Bruchteil der Funktionalitäten. Die starke Ressourcenlimitierung der Plattformen hat somit sehr einfache und funktional stark limitierte Schnittstellen und Dienste zur Folge. Die daraus entstehenden einfachen Nachrichten beinhalten kaum sich wiederholende Elemente und Bytefolgen im Datenstrom, wodurch alle Lösungen nahezu identische Ergebnisse liefern.

Lediglich das EXI-Format liefert im *schema-informed* Modus deutlich bessere Resultate. Als XML-Schemas, die zur Generierung der EXI-Grammatik genutzt werden, wurden die Schema-Dokumente von DPWS sowie die dazugehörigen Schemas der W3C Web Services Spezifikationen verwendet. Für die Messungen wurden die genutzten Schemas im Original verwendet und nicht weiter angepasst, um einen objektiven Vergleich zu gewährleisten. Der *schema-less* Modus des EXI-Formates ähnelt stark dem XMill-Kompressor. Dennoch resultiert das EXI-Format durch die Nutzung des *bit-aligned* Modus bzw. die nachträglich auf den *byte-aligned* kodierten Datenstrom angewandte Kompression mit Hilfe des generischen Deflate-Algorithmus in einer merklich besseren Kompressionsrate.

Aus den Untersuchungen ist zu entnehmen, dass bei einfachen Diensten mit einfachem Nachrichtenaufbau automatenbasierte Lösungen mit bekanntem Vokabular und bekannter Grammatik klar zu bevorzugen sind.

Weiterentwicklung von EXI für den Einsatz von DPWS in 6LoWPAN-Netzwerken

Die Leistungsfähigkeit der beiden automatenbasierten Formate EXI und Fast Infoset kann durch Anpassungen der genutzten Grammatiken und des Vokabulars verbessert werden. Im Nachfolgenden wird daher untersucht, wo das Optimum bezüglich der zu erreichenden minimalen Nachrichtengröße liegt. Weiterhin gilt es zu untersuchen, welcher der Modi des jeweiligen Formates am effizientes ist. Die Erweiterungen der Grammatiken kann dabei in protokoll- und anwendungsspezifische Anpassungen unterteilt werden.

DPWS stellt zugleich eine Einschränkung und eine Erweiterung existierender WS-* Protokolle für deren Einsatz im gerätenahen Umfeld dar. Die Anpassung der Schema-Dokumente der nur zu Teilen verwendeten WS-* Protokolle ist nicht Bestandteil von DPWS. Die protokollspezifischen Anpassungen der EXI-Grammatik beziehen sich daher auf die Anpassung an das konkrete Geräteprofil DPWS und sind unabhängig von einem dedizierten Anwendungsszenario.

Anwendungsspezifische Erweiterungen fügen zu den protokollspezifischen Optimierungen weiterhin Informationen über das konkrete Szenario hinzu.

Solche anwendungsspezifischen Optimierungen eignen sich eher in statischen Umgebungen mit bekannten Sensoren. Besteht das Netzwerk beispielsweise ausschließlich aus Temperatursensoren, die einen bestimmten Bereich überwachen, so kann durch die Schemas die Automatenstruktur der Nachrichten vorab vollständig definiert werden. In dynamischen Netzwerken mit vielen unbekannten Geräten und Diensten sind protokollspezifische Optimierungen aufgrund eines eher generischen Charakters zu bevorzugen. Der *non-strict* Modus des EXI-Formates ermöglicht grundsätzlich die Abweichung von den in der Grammatik definierten Elementen. Daher können ebenfalls Mischformen verwendet werden, bei denen die Schemas für dedizierte Szenarien optimiert werden, die definierte Grammatik aber zur Laufzeit nicht obligatorisch ist.

Die protokollspezifischen Anpassungen zielen konkret darauf ab, bereits durch die Spezifikationen bekannte Bezeichner und Werte in die Automatenstruktur aufzunehmen. Beispiele hierfür sind die Namen der Operationen und Methoden, die der *Hosting Service* eines DPWS-Gerätes immer anbieten muss. Die XML-Schema-Dokumente der jeweiligen Spezifikationen enthalten im Allgemeinen keine solche Werte, sondern lediglich Strukturelemente wie Tag- und Attributnamen. Für die Definition bekannter Inhalte kann der *xs:restriction* Mechanismus in Verbindung mit *xs:enumerations* zum Einsatz kommen. Die Abkürzung *xs:* steht stellvertretend für Elemente der XML-Schema Sprache. Mit Hilfe von *xs:restrictions* kann der Wertebereich von XML-Elementen eingeschränkt werden. Mit Hilfe des *xs:enumeration* Mechanismus werden die konkreten, validen Elemente definiert. Für die protokollspezifischen Optimierungen kann dadurch im Nachfolgenden ausschließlich der *non-strict* Modus des EXI-Formates genutzt werden, um zusätzlich zu den bekannten Inhalten und Werten auch abweichende Daten ermöglichen zu können. Für die Nutzung des *strict* Modus müssten bei einigen Elementen ebenfalls anwendungsspezifische Inhalte in die *xs:enumerations* aufgenommen werden. Die konkreten protokollspezifischen Änderungen sind im Anhang getrennt nach den WS-* Spezifikationen dargestellt.

Die Ergebnisse der Messungen nach der protokollspezifischen Anpassung der EXI-Grammatik zeigt Tabelle 6.3. Auch diese Messungen erfolgten offline ohne die Einbindung in einen DPWS-Stack. Um den neuen Datentypen zu entsprechen und eine bestmögliche Kodierung zu ermöglichen, sind ebenfalls Anpassungen an den Ausgangsnachrichten des ursprünglichen Referenzszenarios nötig. Daher unterscheiden sich die Ausgangsgrößen der Unicode kodierten

XML-Dokumente in Tabelle 6.3 von den Ausgangsgrößen der Initialen Messung in Tabelle 6.2. Aus den Ergebnissen in Tabelle 6.3 wird deutlich, dass das EXI-Format merklich bessere Resultate als Fast Infoset liefert. Der Grund hierfür liegt unter anderem im *bit-aligned* Modus, der nur von EXI unterstützt wird. In Bezug auf das EXI-Format ist zu beobachten, dass der einfache *bit-aligned* Modus dem *compressed* Modus vorzuziehen ist, da die resultierenden Nachrichten durchschnittlich in etwa gleich groß sind, aber die Komplexität des Formates durch die Vermeidung des Deflate-Algorithmus beim *bit-aligned* Modus geringer ist. Dies begründet sich mit dem Fakt, dass der *compressed* Modus den *byte-aligned* Modus vor der Anwendung des Deflate-Algorithmus impliziert. Bei den einfachen Diensten der limitierten Plattformen entstehen in den Nachrichten weder in den Strukturelementen noch in den Werten der Elemente signifikante Redundanzen.

Zuzüglich zu den protokollspezifischen Erweiterungen der Automatenstrukturen können weiterhin anwendungsspezifische Informationen in die EXI-Grammatik eingebracht werden. Als konkretes Szenario wurde im Rahmen dieser Arbeit das bekannte Referenzszenario genutzt. Die Erweiterungen der Schemas sind ebenfalls nach Spezifikationen getrennt im Anhang dargestellt.

Format	Mittelwert		Minimum		Maximum	
	Byte	%	Byte	%	Byte	%
EXI[2]	116,33	15,19	17,00	4,38	350,00	17,30
EXI[1]	118,72	15,50	29,00	7,47	297,00	14,68
EXI[3]	144,39	18,86	28,00	7,22	430,00	21,26
Fast Infoset[1]	218,39	28,52	103,00	26,55	455,00	22,49
Fast Infoset[3]	242,00	31,60	97,00	25,00	563,00	27,83
EXI[4]	280,72	36,66	168,00	43,30	586,00	28,97
EXI[5]	419,67	54,81	205,00	52,84	1.060,00	52,40
EXI[6]	453,22	59,19	218,00	56,19	1.166,00	57,64
UTF-8 XML	765,72	100,00	388,00	100,00	2.023,00	100,00

[1]schema-informed / compressed
[2]schema-informed / bit-aligned
[3]schema-informed / byte-aligned
[4]schema-less/ compressed
[5]schema-less / bit-aligned
[6]schema- less / byte-aligned

Tabelle 6.3 Automatenbasierte Verfahren mit protokollspezifischen Anpassungen

In Tabelle 6.4 sind die Ergebnisse der Messungen für eine bestmögliche anwendungsspezifische Erweiterung der Automatenstruktur an das Referenzszenario dargestellt. Die resultierenden Nachrichten können gegenüber den protokollspezifischen Anpassungen im Mittel um weitere 51 Byte verkleinert werden, was einer Verbesserung von rund 44 % entspricht. Die kleinste Nachricht ist identisch geblieben. Die umfangreichste Nachricht, welche die übertragenen Metadaten des Gerätes darstellt, kann durch die anwendungsspezifische Optimierung um etwa 196 Byte hinsichtlich der Größe verringert werden. Der grundsätzliche Trend aus den protokollspezifischen Optimierungen, dass der reine *bit-aligned* Modus dem *compressed* Modus vorzuziehen ist, lässt sich auch in den dargestellten Werten in Tabelle 6.4 feststellen.

Als Ergebnis der Untersuchungen lässt sich schlussfolgern, dass das automatenbasierte EXI-Format im Hinblick auf die resultierenden Nachrichtengrößen allen anderen untersuchten Formaten und Kompressoren vorzuziehen ist. Die Erweiterung der genutzten Automatenstruktur kann dabei zum einen protokollspezifisch und zum anderen anwendungsspezifisch erfolgen. Welche Erweiterungen der Grammatik für eine konkrete Umsetzung zu nutzen sind, ist abhängig davon, ob für die Umgebung die Geräte- und Dienstinformationen vorab bekannt sind. In einem statischen Fall kann durch die anwendungsspezifische Erweiterung die kompakteste Form erreicht werden. In dynamischeren Szenarien, die sich oft ändernde Geräte und Dienste enthalten, sind generische,

Format	Mittelwert		Minimum		Maximum	
	Byte	%	Byte	%	Byte	%
EXI[2]	65,33	8,53	17,00	4,38	154,00	7,61
EXI[1]	83,61	10,92	29,00	7,47	179,00	8,85
EXI[3]	93,33	12,19	28,00	7,22	239,00	11,81
EXI[4]	281,06	36,70	168,00	43,30	586,00	28,97
EXI[5]	420,33	54,89	205,00	52,84	1.060,00	52,40
EXI[6]	453,89	59,28	218,00	56,19	1.166,00	57,64
UTF-8 XML	765,72	100,00	388,00	100,00	2.023,00	100,00

[1]schema-informed / compressed
[2]schema-informed / bit-aligned
[3]schema-informed / byte-aligned
[4]schema-less/ compressed
[5]schema-less / bit-aligned
[6]schema- less / byte-aligned

Tabelle 6.4 Automatenbasierte Verfahren mit anwendungsspezifischen Anpassungen

protokollspezifische Erweiterungen ausreichend. Mit der Möglichkeit der Nutzung des *non-strict* Modus von EXI, bei dem von der bekannten Automatenstruktur abweichende Elemente auftreten können, sind ebenfalls Mischformen aus protokollspezifischen und anwendungsspezifischen Anpassungen möglich.

Folglich existiert eine Lösung, die den Bandbreitenanforderungen von 6LoWPAN-Netzwerken genügt. Die ermittelte resultierende Gesamtgröße der Nachrichten, die Konformität von EXI zu XML-Infoset, XML-Schema und somit zur WS-Architecture sowie die Flexibilität bei der Definition des Vokabulars und der Grammatik prädestinieren das erweiterte EXI-Format im *schema-informed non-strict bit-aligned* Modus als den geeignetsten Kandidaten für den Einsatz in 6LoWPANs. Daher wird nachfolgend untersucht, wie sich das EXI-Format in den DPWS-Protokollstapel integrieren lässt. Darauf aufbauend wird im Kapitel 6.3 untersucht, ob die Minimierung Nachrichtengrößen durch die Erweiterung der Grammatik nicht einen unverhältnismäßig großen Anstieg der Komplexität der Implementierung und des Speicherbedarfs verursacht.

6.2.2 Integration von EXI in DPWS im Kontext von 6LoWPANs

Soll das EXI-Format im Zusammenhang mit SOAP bzw. DPWS eingesetzt werden, muss die zusätzliche Kodierung in die Gesamtarchitektur integriert werden. Da die Serialisierung der Strukturen entsprechend XML-Infoset obligatorisch ist, wozu natives Unicode XML wie auch EXI in der Lage sind, fügt sich die Nutzung des EXI-Formates nahtlos in die angestrebte Gesamtarchitektur (siehe Kapitel 4) ein. EXI und XML-basierte SOAP-Dokumente sind zustandslos und ohne Kenntnis über die Anwendung ineinander überführbar. Einzige benötigte Information ist die EXI-Grammatik, die kontextfrei aus den XML-Schemabeschreibungen generiert werden kann. Die Transformation zwischen Unicode XML und EXI kann zum Beispiel in speziellen zustandslosen Proxys geschehen. Innerhalb des ressourcenlimitierten Netzwerkes kann ausschließlich das EXI-Format verwendet werden, während in bestehenden Netzwerken weiterhin das verbreitete XML-Format genutzt werden kann. Auch Mischformen sind möglich.

Ob ein Dokument jeweils in XML oder EXI kodiert ist, kann direkt aus dem Datenstrom abgelesen werden. Das EXI-Format unterscheidet sich bereits im ersten Byte von einem Unicode XML-Datenstrom und kann somit getrennt behandelt werden. Hinweise auf die verwendete Kodierung können ebenfalls durch das Transportbinding gegeben sein. Das existierende

SOAP-over-HTTP Binding zeigt beispielsweise die verwendete Kodierung durch ein HTTP-Headerfeld an.

Für die Nachrichtenkodierung im EXI-Format sowie für die nahtlose Umsetzung der beiden Formate XML und EXI ineinander müssen beide Endpunkte über die gleichen EXI-Grammatik verfügen. EXI schreibt für die Aushandlung der genutzten Automatenstruktur zur Laufzeit keine Vorgehensweise vor. Lediglich für das Anzeigen der genutzten Automatenstruktur für den jeweiligen EXI-Stream steht im EXI-Header ein Informationsfeld bereit. Der Inhalt, die Semantik und der Wertebereich dieses Feldes sind aber nicht weiter eingeschränkt. Mögliche Werte sind zum Beispiel eine Auflistung aller URIs der Namensräume der zu den Schemas gehörenden Spezifikationen. Da das Feld und dessen Inhalt selbst zu Kompatibilitätszwecken *schema-less* kodiert sind, würde eine solche Auflistung als Zeichenkette kodiert werden und somit zu umfassend und nicht zweckmäßig sein. Im Zusammenhang mit dem speziellen DPWS-Profil wären stellvertretend für die involvierten Spezifikationen auch eine einzelne URI (z.B. Namensraum von DPWS) bzw. eine noch kompaktere Darstellung in Form eines Hashwertes möglich. Da DPWS aber durch andere WS-Spezifikationen und auch eigene Erweiterungen ergänzt werden kann, ist eine verallgemeinerte Festlegung des Wertebereichs des Feldes auch im Zusammenhang mit DPWS nicht sinnvoll. Wird ausschließlich das reine DPWS-Profil verwendet, so kann auf die erwähnte stellvertretende URI zurückgegriffen werden. Weiterführende Festlegungen obliegen folglich Herstellern oder Standardisierungsgremien, welche spezifische Geräte bzw. Gerätetypen definieren. Für die Definition der Gerätetypen kann der in Kapitel 5.1.3 konzipierte Template-Ansatz zur Beschreibung von DPWS-Gerätetypen verwendet und um Angaben zur EXI-Grammatik ergänzt werden.

Für eine möglichst dynamische Integration verschiedener Sensoren sind Mischformen aus *schema-informed* und *schema-less* möglich. Welcher der Modi genutzt wird, kann ebenfalls im EXI-Header angezeigt werden. Während der Discovery-Phase kann beispielsweise ausschließlich der *schema-less* Modus genutzt werden. Hierdurch vergrößert sich die Nachrichtengröße. Das verwendete Format wird aber potenziell mit mehr Endpunkten kompatibel. Bestandteil der Discovery-Phase kann ebenfalls der Austausch von Informationen über die Automatenstrukturen sein. Für den nachfolgenden Nachrichtenaustausch, wie beispielsweise das Aufrufen von konkreten Diensten und deren Methoden, kann der effizientere *schema-informed* Modus verwendet werden.

DPWS nutzt laut Spezifikation in einigen Nachrichten Präfixe für XML-Namensräume als inhaltliche Werte von Elementen (z.B. in *Hello* und *Probe Match* im *Types* Feld). Solche Präfixe sind innerhalb eines SOAP-Dokuments in der Regel frei wählbar und dienen lediglich der Zuordnung von Bezeichnern zu einem spezifischen Namensraum, ohne für diese Zuordnung bei jedem Auftreten immer die vollständige URI des Namensraumes zu verwenden. Das EXI-Format behandelt die Strukturbezeichner und inhaltliche Werte aber unterschiedlich. Die Strukturbezeichner und die zugehörigen Namensräume können in der bekannten EXI-Grammatik enthalten sein und somit vollständig binär im Datenstrom kodiert werden. Die inhaltlichen Werte hingegen werden unter Umständen als abstrakte Zeichenketten bzw. als Bytefolge behandelt. Tritt innerhalb eines Wertes ein Namensraum auf (z.B. als XML-Qualified-Name), so ist dieser beim Parsen unter Umständen nicht eindeutig einem spezifischen Wert der EXI-Grammatik zuzuordnen. Daher müssen bei der Kodierung von SOAP-Dokumenten, die Präfixe in Werten enthalten, darauf geachtet werden, dass Präfixdefinitionen erhalten bleiben. Eine mögliche Lösung bildet die Option *Preserve Prefixes* des EXI-Formates.

Es ist abhängig von der Implementierung von EXI, ob auf den Systemen zur Laufzeit dynamischen zwischen den verschiedenen Modi von EXI und zwischen verschiedenen EXI-Grammatiken unterschieden werden kann. Im trivialsten Fall werden von den konkreten Implementierungen nur eine spezifische Automatenstruktur und ein dedizierter Modus des EXI-Formates unterstützt.

6.3 Implementierung von EXI

Nachdem im vorrangegangenen Unterkapitel der Nachweis über die Effizienz des erweiterten EXI-Formates erbracht werden konnte, benötigt eine vollständige Evaluation weiterhin den Nachweis der Umsetzbarkeit auf Zielplattformen von 6LoWPANs. Eine existierende Implementierung von EXI ist unter dem Begriff EXIficient als Open Source verfügbar [146]. EXIficient ist in der Programmiersprache Java implementiert und durch die fehlende Unterstützung von Java auf Zielplattformen von 6LoWPANs nicht geeignet (siehe Kapitel 4.4.3). Weitere Implementierungen in den Programmiersprachen Java und C++ werden von der Firma AgileDelta kommerziell vertrieben. Auch von der Firma AgileDelta existiert keine Lösung, die auf Zielplattformen der Klasse 1 direkt eingesetzt werden kann.

Somit entstand im Rahmen dieser Arbeit mit der Unterstützung zweier studentischer Arbeiten von Gotzmann [147] und Rethfeldt [148] erstmals ein EXI-Parser, der die Ressourcenanforderungen

von Zielplattformen von 6LoWPANs erfüllt. Ziel ist es zu untersuchen, welchen minimalen Speicherbedarf eine Implementierung von EXI besitzt und welchen Einfluss die Komplexität der verwendeten Automatenstruktur (vgl. EXI-Grammar) in Bezug auf den Speicherbedarf aufweist. Der Parser heißt uEXI und wird unter der Bezeichnung WS4D-uEXI im Rahmen der WS4D-Initiative als Open Source Projekt publiziert. Konkrete Details bezüglich der Implementierung sind über die Webseite des WS4D-uEXI Projektes erhältlich. Diese Arbeit beschränkt sich auf eine konzeptionelle Beschreibung von uEXI sowie die Darstellung der Ergebnisse hinsichtlich des Ressourcenbedarfes.

uEXI ist ein reiner Parser (vgl. Dekodierer) für EXI-Streams. Die Kodierung von Daten im EXI-Format ist mit uEXI nicht möglich, da die Anwendung von EXI im Kontext von DPWS in 6LoWPANs dies nicht zwingend erfordert. Die ausgehenden und zu kodierenden Daten sind bereits zur Entwicklungszeit bekannt und können statisch eingebunden werden, wie dies in Kapitel 5.3 für die Implementierung von uDPWS dargestellt ist. Lediglich eingehende Nachrichten unterliegen zur Laufzeit dynamischen Änderungen und müssen verarbeitet werden. uEXI kann durch die grundsätzliche Konzeption der Implementierung dahingehend erweitert werden, auch Daten kodieren zu können. Durch die Implementierung soll aber die unterste Grenze des Speicherbedarfes bestimmt werden. Aufgrund dieser Anforderung zur Minimierung des Speicherbedarfes wurde auf die Realisierung von Verfahren zur Kodierung bei der Implementierung von uEXI verzichtet.

6.3.1 Implementierung

uEXI ist nicht an ein spezielles Betriebssystem oder einen Protokollstack gebunden, wie es bei uDPWS bezüglich der Nutzung von Contiki der Fall ist. uEXI ist in der Programmiersprache C implementiert und wurde im Rahmen der Entwicklung sowohl auf Linux-Systemen wie auch auf Contiki evaluiert.

Die uEXI-Implementierung besteht aus zwei Modulen: dem uEXI-Parser und der uEXI-Grammatik. Der uEXI Parser umfasst die Verarbeitungseinheit von uEXI und stellt Funktionen für das Dekodieren des EXI-Streams bereit. Dies beinhaltet beispielsweise die Unterscheidung der verschiedenen Modi von EXI und die Behandlung der spezifischen Datentypen, wie beispielsweise Zeichenketten, Integer-Zahlen, Boolean und Datumsangaben. Der uEXI-Parser unterstützt die Modi *schema-informed* und *schema-less* im *bit-* und *byte-aligned* Format. Weiterhin stehen die beiden Modi *strict* und *non-strict* zur Verfügung. Das uEXI-Parser

Modul ist unabhängig von der Anwendung und der genutzten Automatenstruktur. Die uEXI-Grammatik wird daher in einem zweiten, unabhängigen Modul bereitgestellt. Dies ermöglicht es, zur Laufzeit parallel verschiedene Modi und Automatenstrukturen zu unterstützen.

Aufgrund der Ressourcenlimitierungen ist es nicht zweckmäßig, die uEXI-Grammatik zur Laufzeit zu erzeugen. Die für die Generierung der uEXI-Grammatik genutzten XML-Schemas haben jeweils eine typische Größe von mehreren kB. Für das genutzte Referenzszenario beträgt die Größe der XML-Schemas bereits mehr als 30 kB.

Die für den *schema-informed* Modus benötigten Automaten werden somit zur Entwicklungszeit generiert und statisch in die Implementierung im ROM eingebettet. Zur Laufzeit kann die uEXI-Grammatik nicht angepasst werden. Ist genügend Speicher auf der Zielplattform vorhanden, können mehrere Automatenstrukturen vorgehalten und die jeweilige uEXI-Grammatik abhängig vom Kontext ausgewählt werden. Ob und welche uEXI-Grammatik von uEXI bzw. dem Protokollstack zur Laufzeit unterstützt wird, kann z.B. durch Mechanismen von EXI oder von DPWS ausgehandelt werden.

Die Realisierung der uEXI-Grammatik erfolgt als Zustandsmaschine in C. Ein auftretendes EXI-Event im EXI-Stream zeigt den Übergang zum nächsten Zustand an. Für die Implementierung wurden *switch*-Anweisungen genutzt. Der Vorteil der Nutzung von *switch*-Anweisungen zur Abbildung der Zustände und der Übergänge liegt darin, dass die Anweisungen ausschließlich aus sich referenzierenden Zeigern im Speicherbereich bestehen. Entsprechende Compiler für C können mehrere aufeinander folgende Zeiger optimieren und zu einem einzelnen Sprung im Speicher zusammenfassen. Für die Behandlung der Zustände bzw. EXI-Events innerhalb der *switch*-Anweisung, die jeweils das Auftreten eines bestimmten Elementes oder Wertes innerhalb des EXI-Streams repräsentieren, können in uEXI Callback-Funktionen definiert werden. Der für die uEXI-Grammatik benötigte Quelltext in C wird mittels eines eigens entwickelten

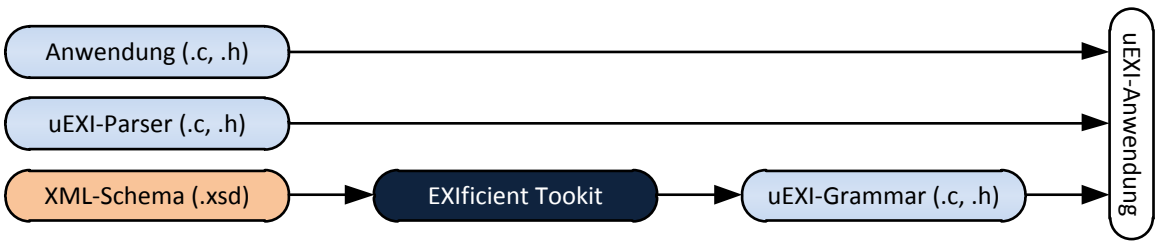

Abbildung 6.8 uEXI Entwicklungsprozess

Code-Generators aus dem existierenden EXIficient Toolkit generiert (siehe Abbildung 6.8).

6.3.2 Speicherbedarf

Tabelle 6.5 stellt den statischen Speicherbedarf des ROMs von uEXI mit verschiedenen Konfigurationen dar. Die aufgezeigten Werte entsprechen der Kompilierung für die 16 Bit Zielplattform TelosB. Jede Zeile der Tabelle enthält die Module bzw. uEXI-Grammatiken, die in der Zeile angegeben sind zuzüglich der darüber dargestellten Module bzw. uEXI-Grammatiken. uEXI, inklusive einer vollständigen anwendungsspezifischen Automatenstruktur, weist einen ROM-Bedarf von etwa 18 kB auf. Lediglich rund 1 kB entfällt dabei auf den uEXI-Parser. Es wird somit deutlich, dass der eigentliche grammatikunabhängige Parser nur einen geringen Anteil zum ROM-Bedarf beiträgt. Der Speicherbedarf wird maßgeblich von der verwendeten Automatenstruktur und dessen Komplexität bestimmt.

Ebenfalls fällt auf, dass das Hinzufügen der anwendungsspezifischen Grammatikelemente (Air Conditioner) den ROM-Bedarf um lediglich 308 Byte/1,7 % steigert. Die vorrangegangenen Untersuchungen haben aber gezeigt, dass die resultierende Nachrichtengröße durch das Hinzufügen anwendungsspezifischer Grammatikelemente im Mittel um etwa 44 % gesunken ist. Es kann somit geschlussfolgert werden, dass die anwendungsspezifische Anpassung bzw. Erweiterung der EXI-Grammar eine sinnvolle Maßnahme darstellt, die den ROM-Bedarf nur geringfügig beeinflusst.

Bei den aufgeführten Werten gilt es weiterhin zu beachten, dass die Grammatiken jeweils die

Module und Grammatiken	**ROM [Byte]**	**Diff [Byte]**
uEXI-Parser (entspricht schema-less)	1.088	-
+SOAP, XML, XML-Schema, XML-Namespace	4.372	3.284
+WS-Addressing	7.002	2.630
+WS-Eventing	10.588	3.586
+WS-MetadataExchange	11.672	1.084
+WS-Discovery	15.246	3.574
+DPWS	17.692	2.446
+Air Conditioner (Referenzszenario)	18.000	308

Tabelle 6.5 Statischer ROM Speicherbedarf uEXI

vollständigen Automatenstrukturen der WS-* Spezifikationen und von DPWS enthalten. Eine mögliche, weiterführende, anwendungsspezifische Optimierung wäre somit die Entfernung der Elemente des Automaten, die zur Laufzeit von dem jeweiligen Endpunkt (Gerät, Gerätetyp oder Dienst) nicht unterstützt werden. Dennoch wird aus den Werten deutlich, dass EXI prinzipiell auf Zielplattformen der Klassen 1 und 2 möglich und zweckmäßig ist. Diese Aussage war vor der Implementierung von uEXI nicht möglich.

Der dynamische Speicherbedarf des RAMs durch uEXI kann nicht eindeutig ermittelt werden. Grundsätzlich benötigt uEXI keinen eigenen Nachrichtenspeicher, sondern verarbeitet direkt die Daten aus dem Nachrichtenpuffer, die z.B. als Referenz übergeben werden. Der dynamische Speicherbedarf ist somit ausschließlich von der Menge der während der Verarbeitung eines EXI-Streams auftretenden Elemente und Werte abhängig, die nicht bereits durch die EXI-Grammar bekannt sind. Diese unbekannten Elemente und Werte treten innerhalb eines EXI-Streams nur einmal auf und werden bei jedem weiteren Auftreten referenziert. Diese Elemente müssen also für die Dauer der Verarbeitung eines EXI-Streams gepuffert werden. Wie viele unbekannte Elemente und Werte in einem EXI-Stream auftreten und wie viel Speicher diese für das Puffern benötigen, hängt von der konkreten Kommunikation ab und kann nur für die jeweils konkrete Nachricht in Abhängigkeit von der verwendeten Grammatik bestimmt werden.

Ein Vergleich von uEXI mit anderen Ansätzen ist ebenfalls nur schwer möglich. Die existierende Implementierung EXIficient und einige kommerzielle Produkte der Firma AgileDelta sind in Java und somit nicht primär für Zielplattformen von 6LoWPANs programmiert. Die Implementierungen sind als Grundvoraussetzung auf eine CLDC-konforme Java-Laufzeitumgebung angewiesen, die entsprechend der Diskussion in Kapitel 4.4.3 für einen TelosB-Knoten bereits rund 34,5 kB des ROM-Speichers benötigt (vgl. max. 18 kB für uEXI inklusive EXI-Grammar). EXIficient selbst hat zuzüglich einen ROM-Bedarf von einigen hundert kB. Die kleinste von AgileDelta verfügbare Lösung, die im Rahmen dieser Arbeit zur Evaluation als Binärdatei (ohne Quelltext) kostenlos zur Verfügung gestellt wurde, benötigt zuzüglich etwa 22 kB. Allerdings beinhalten beide genannten Lösungen nur die grundlegenden Behandlungsroutinen und noch keine EXI-Grammar. Damit sind diese Implementierungen funktional dem reinen uEXI-Parser ähnlich. Der uEXI-Parser weißt mit lediglich rund 1 kB einen Bruchteil des ROM-Bedarfes von EXIficient sowie der Lösung von AgileDelta auf. Auch ist uEXI dabei auf keine weitere Laufzeitumgebung oder eine externe Bibliothek angewiesen. Vor allem die EXI-Grammar übt entsprechend den Untersuchungen mit

uEXI einen immensen Einfluss auf den ROM-Bedarf aus. Ein weiterführender, objektiver Vergleich zwischen den verschiedenen Implementierungen unter Einbeziehung des Einflusses unterschiedlicher Automatenstrukturen erweist sich als nicht möglich.

Käbisch et al. haben in [149] eine Implementierung von EXI vorgestellt, die wie auch uEXI in C programmiert und mit Contiki kompatibel ist. Die vorgestellte Implementierung wurde allerdings für eine ARM Cortex-M3-Plattform mit 32 Bit-Architektur entwickelt, wodurch direkte Vergleiche mit der 16 Bit-Architektur des TelosB kaum möglich sind. Die Implementierung von Käbisch et al. steht für weiterführende Untersuchungen nicht zur Verfügung. Der insgesamt benötigte ROM-Bedarf für Contiki und die EXI-Implementierung zusammen beträgt nach Käbisch et al. 63 kB. Nicht angegeben ist, wie groß der Anteil von EXI ist, welche Schemas zur Generierung der EXI-Grammar genutzt wurden und welchen Speicherbedarf die verwendete Automatenstruktur verursacht.

Weitere Implementierungen von EXI oder anderen automatenbasierten XML-Kompressoren für Zielplattformen der Klasse 1 sind derzeit nicht vorzufinden.

6.4 Zwischenfazit

In diesem Kapitel wurden, nach intensiver Analyse der Anforderungen, die Stärken und Schwächen geeigneter Formate und Komprimierungen für SOAP-basierte Nachrichten im Kontext von 6LoWPANs identifiziert. Gegenstand der Evaluierung waren einfache Dienste, wie sie von Sensoren oder Aktoren in 6LoWPANs bereitgestellt werden. Dabei haben sich automatenbasierte Verfahren, mit einem hohen Grad an bekanntem Vokabular und bekannter Grammatik als am effizientesten herausgestellt. Alle anderen existierenden generischen Kompressoren und einfache XML-Kompressoren, die kein bekanntes Vokabular und keine bekannte Grammatik nutzen, erreichen im Vergleich untereinander bei der Anwendung auf die einfachen Dienste und somit einfachen Nachrichten im Schnitt ähnliche Kompressionsraten. Der Grund hierfür liegt in den wenigen sich wiederholenden Elemente und Bytefolgen in den Nachrichten, wodurch nur wenige der Mechanismen der spezifischen Formate effizient sind.

Bei den automatenbasierten Verfahren hat sich das EXI-Format, welches seit März 2011 den Status einer W3C-Recommendation besitzt, als besonders effektiv und zugleich flexibel herausgestellt. Durch EXI werden einige Freiheitsgrade von nativem XML und XML-Infoset eliminiert bzw.

deren Einfluss sinkt. Der Informationsgehalt bleibt dennoch identisch. Somit hat z.B. die Länge der Zeichenketten zur Benennung von Informationsträgern und teilweise auch von Inhalten keinen Einfluss auf die Größe der Kodierung. Der Vorteil gegenüber anderen binären Kodierungen besteht darin, dass trotz der binären Kodierung keine Überlappungen bei der Kodierung auftreten können, da mittels XML-Schemadokumente die Kodierung formal beschrieben werden kann. Diese XML-Schemadokumente sind durch die WS-Architecture bei SOAP-basierten Web Services obligatorisch. Somit fügt sich das EXI-Format nahtlos in die konzipierte Gesamtarchitektur und das entwickelte Protokollframework ein. Unter der immensen Vielzahl der Modi des EXI-Formates konnte anhand feingranularer Untersuchungen der Modus *schema-informed bit-aligned non-strict* als die optimale Variante selektiert werden.

Anschließend wurde mit der neuartigen Implementierung uEXI erstmals der Nachweis erbracht, dass ein EXI-Parser auch für stark ressourcenlimitierte Plattformen realisiert werden kann. Dieser Nachweis war vorher nicht gegeben, da keine andere Implementierung dieser Art existierte. uEXI wird aktuell im Rahmen WS4D-Initiative als Open Source Projekt gepflegt. Basierend auf uEXI sind somit erstmals Untersuchungen zum Speicherbedarf und zum Einfluss der Komplexität der Automatenstrukturen (vgl. EXI-Grammatiken) auf den ROM-Bedarf möglich. uEXI zeichnet sich durch einen statischen Speicherbedarf von wenigen kB aus. Die verwendeten Automatenstrukturen haben den größten Einfluss auf den statischen Speicherbedarf. Die Untersuchungen haben aber ebenfalls ergeben, dass anwendungsspezifische Anpassungen und Erweiterungen der Automatenstruktur kaum einen Einfluss auf den Speicherbedarf haben, zugleich aber die Nachrichtengrößen auf etwa die Hälfte verringern.

uEXI kann bezüglich des Speicherbedarfes aber noch weiter optimiert werden. Eine mögliche Optimierung kann darauf fokussiert sein, Elemente zur Entwicklungszeit aus der Automatenstruktur zu entfernen, die zur Laufzeit nicht benötigt werden. Diese Erweiterung ist anwendungsabhängig und für die allgemeinen Untersuchungen bezüglich der Machbarkeit im Rahmen dieser Arbeit nicht relevant. Der effektive Nutzen der Anpassung ist ebenfalls anwendungsabhängig und kann nicht pauschal abgeschätzt werden.

Für die grammatikbasierte Kodierung der Nachrichten müssen beide Kommunikationspartner allerdings identische Automatenstrukturen nutzen. Voraussetzung zur Kommunikation ist daher die Existenz von Mechanismen, um zum einen die genutzten Automaten anzuzeigen bzw. diese zur Laufzeit auszuhandeln und zum anderen ggfs. die entsprechenden Automaten zur Laufzeit zu

generieren. Für die Aushandlung der EXI-Grammatiken stehen in EXI und auch in DPWS verschiedene solcher Mechanismen zur Verfügung. Kann der jeweils andere Kommunikationspartner nicht die vorgegebenen Automatenstrukturen unterstützen, muss die Kommunikation im schlechtesten Fall vollständig ohne bekanntes Vokabular und ohne bekannte Grammatik durchgeführt werden (vgl. *schema-less*). In diesem schlechtesten Fall erreicht EXI ähnliche Kompressionsraten wie generische Kompressoren und einfache XML-Kompressoren. Aus diesem Grund müssen protokoll- bzw. anwendungsspezifische Anpassungen der Automaten das Verhältnis von Kompressionsrate zu Interoperabilität beachten.

Zusammenfassend konnte somit eine Nachrichtenkodering konzipiert werden, die (1) den Bandbreitenbeschränkungen von 6LoWPANs, (2) der nahtlosen Integration in DPWS, SOAP und die WS-Architecture und (3) den Ressourcenanforderungen der Zielplattformen von 6LoWPANs gerecht wird. Weiterhin fügt (4) sich das erweiterte EXI-Format durch die Möglichkeit der zustandslosen Überführung in natives Unicode XML mittels Proxys in die skalierbare Netzwerkinfrastruktur ein, die in Kapitel 4 beschrieben ist.

Bezüglich der Leistungsfähigkeit von SOAP Web Services und von DPWS haben bei hohen Kompressionsraten und den entsprechend kleineren zu übertragenden Nachrichten die Komplexität und der Protokolloverhead des Transportmechanismus für die Nachrichten in der Relation einen größeren Einfluss als bei nicht komprimierten Nachrichten. Im nachfolgenden Kapitel 7 werden daher existierende SOAP-Transportmechanismen tiefgehend evaluiert und erstmals eine vollständig neue Lösung spezifiziert, die sowohl die Ressourcenlimitierungen als auch die ganzheitliche Architektur berücksichtigt.

7 Effizienter Transport für SOAP-Dokumente

Inhaltsübersicht

Wie SOAP-Dokumente zwischen zwei Endpunkten transportiert werden, ist vom W3C im SOAP-Messaging-Framework beschrieben [29]. Das Framework bezieht sich dabei auf kein konkretes Protokoll. Hierdurch wird eine Entkopplung von den konkreten Protokollen und den von diesen Protokollen benötigten Funktionalitäten erreicht. Eine dedizierte Nutzung eines konkreten Protokolls für den Transport von SOAP-Dokumenten wird als *Binding* bezeichnet. Das SOAP-over-HTTP Binding ist die am weitesten verbreitete konkrete Realisierung eines SOAP-Transportprotokolls, das im Anhang zum SOAP-Messaging-Framework beschrieben ist [150].

Zu den allgemeingültigen Mechanismen eines Bindings zählt unter anderem die Definition verschiedener Kommunikationsmuster (engl. Message Exchange Pattern). Ein typisches SOAP-Kommunikationsmuster ist das Anfrage-Antwort-Muster (engl. Request-Response), bei dem auf jede SOAP-Anfrage eine direkte SOAP-Antwort gesendet werden muss. Ein weiteres verbreitetes Muster ist das Anfrage-Muster (engl. Request). Dieses Muster besteht lediglich aus einer SOAP-Anfrage, auf die keine SOAP-Antwort erfolgt. Die beiden genannten Kommunikationsmuster sind in DPWS obligatorisch.

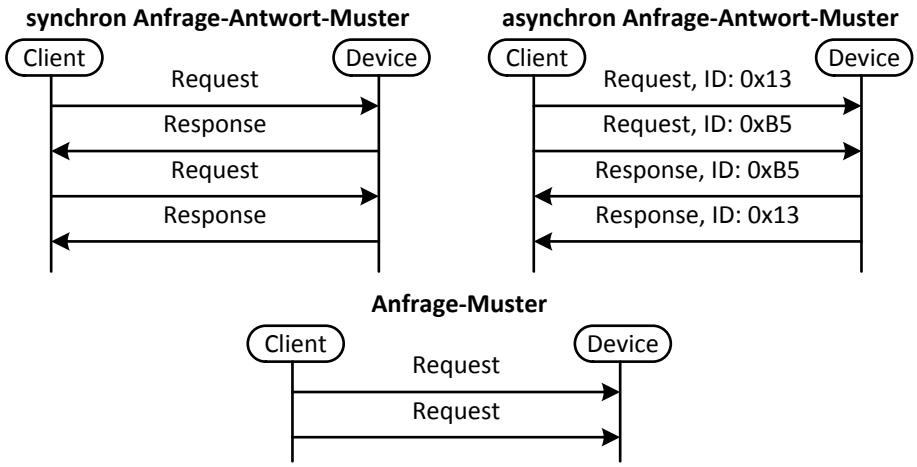

Abbildung 7.1 Schematische Darstellung von SOAP-Transportmechanismen

Protokolle für den Transport von SOAP haben oftmals eigene Kommunikationsmuster. Es ist daher hervorzuheben, dass das genutzte SOAP-Kommunikationsmuster nicht mit dem Muster des Transportprotokolls übereinstimmen muss. Die Transportprotokolle können zwar ebenfalls aus einer Anfrage und einer Antwort bestehen, sind darauf aber nicht beschränkt. Verfügt der Transportmechanismus über ein verbindungsorientiertes Anfrage-Antwort-Muster, können für den Transport des SOAP Anfrage-Antwort-Musters die SOAP-Anfrage und SOAP-Antwort direkt in die Nachricht des dazugehörigen Transportbindings eingebettet werden. Obgleich dies die offensichtlichste Nutzungsform eines so gearteten Transportmechanismus ist, können ebenfalls getrennte Transportaufrufe für die Kommunikation genutzt werden. Im Falle eines verbindungsorientierten Transportprotokolls impliziert dies unter Umständen zwei getrennte Verbindungen. Für das Anfrage-Muster von SOAP wird lediglich in die Anfrage ein SOAP-Dokument eingebunden. Auf Ebene des Bindings muss dann zwar gegebenenfalls auch eine Antwort gesendet werden. Diese Antwort enthält beim reinen Anfrage-Muster aber kein SOAP-Dokument oder andere Nutzdaten.

Basierend auf den Protokolleigenschaften können Transportprotokolle für SOAP daher als synchron oder asynchron klassifiziert werden (siehe Abbildung 7.1). Einige Protokolle sind auch in der Lage, beiden Klassifizierungen zu entsprechen. Bei synchroner Kommunikation erfolgt auf jede SOAP-Anfrage eine direkte SOAP-Antwort. Bei aufeinander folgenden Anfragen ist somit die Reihenfolge entscheidend. Vor dem Senden einer erneuten SOAP-Anfrage muss zuvor auf die SOAP-Antwort der vorrangegangenen Anfrage gewartet werden. Eine Zuordnung der eingehenden Antwort zur Anfrage ergibt sich bei synchroner Kommunikation implizit aus der Reihenfolge. Im

Fall des Anfrage-Kommunikationsmusters von SOAP schließt synchrone Kommunikation die Antwort des Transportmechanismus‘ mit ein. Ausgenommen von dieser Einschränkung bei der synchronen Kommunikation ist das gleichzeitige Nutzen paralleler Kommunikationskanäle. Jeder einzelne Kanal ist in sich wiederum synchron.

Im Gegensatz zu rein synchroner Kommunikation ist bei asynchroner Kommunikation die Reihenfolge der Übermittlung von SOAP-Anfragen und SOAP-Antworten nicht definiert. In Szenarien mit vielen parallelen Aufrufen kann so bei asynchronem Transport ein Kommunikationskanal mehrfach genutzt werden. Dadurch wird der entstehende Aufwand vor allem auf den darunterliegenden Schichten minimiert, da dort nur eine Verbindung verwaltet werden muss. Allerdings benötigen asynchrone Transportmechanismen Erweiterungen auf höheren Schichten, um Anfragen und Antworten einander zuordnen zu können.

7.1 Überblick über existierende SOAP-Bindings

Existierende SOAP-Transportmechanismen können in die beiden Gruppen der UDP- bzw. der TCP-basierten Bindings unterteilt werden. Andere Protokolle auf Transportschicht des OSI-Referenzmodells haben bislang keine nennenswerte Durchdringung erreicht und werden nicht weiter betrachtet. UDP-basierte Bindings sind durch die Zustandslosigkeit des UDP-Protokolls nicht zuverlässig und müssen diese Funktionen auf höheren Protokollebenen abbilden. TCP-basierte Bindings sind durch die Verbindungsorientierung des TCP-Protokolls zuverlässig und bieten darüber hinaus integrierte Mechanismen zur Flusskontrolle. Aufgrund der andersartigen Routingstrategien in vermaschten 6LoWPANs und dem Fakt, dass stark ressourcenlimitierte Plattformen oft nur eine einzelne oder wenige Verbindungen parallel verarbeiten können, ist die Flusskontrolle von TCP in solchen Umgebungen nur bedingt einsetzbar [8]. Somit verbleibt als Vorteil von TCP lediglich die Zuverlässigkeit des Transports.

Das OASIS-Konsortium hat mit WS-ReliableMessaging [151] eine Lösung spezifiziert, die zuverlässigen Transport durch SOAP-interne Mechanismen realisiert. Der Vorteil einer Lösung auf Anwendungsschicht ist die Sicherstellung, dass die Daten beim Empfänger an die Anwendungsschicht weitergereicht werden. Das Senden von Empfangsbestätigungen auf unteren Schichten stellt dies nicht sicher. Nachteilig an einer solchen Lösung ist, dass diese vergleichsweise schwergewichtig gegenüber ressourcenoptimierten Protokollen auf unteren Schichten ist. Im

Kontext stark ressourcenlimitierter Zielplattformen sind leichtgewichtigere Mechanismen zu bevorzugen.

Auf Transportebene können ausschließlich Anwendungen adressiert werden, die durch Ports zugeordnet werden. Da an einer Transportadresse mehrere Dienste angeboten werden können, müssen auf höheren Ebenen weitere Funktionen bereitgestellt werden, um zwischen den Diensten und den Operationen zu unterscheiden. Das W3C hat hierfür WS-Addressing [152] spezifiziert. WS-Addressing ist ebenfalls integraler und obligatorischer Bestandteil des DPWS-Profils. Da dies nicht auf alle WS-* Profile zutrifft, nutzen andere Bindings neben reinem TCP bzw. UDP auch weiterführende Anwendungsschichtprotokolle für den Transport und die Adressierung der Dienste. Ein Beispiel hierfür ist das bereits genannte SOAP-over-HTTP Binding.

Im folgenden Abschnitt werden existierende SOAP-Bindings vorgestellt und auf Erfüllung der Anforderungen für stark ressourcenlimitierte Umgebungen unter Berücksichtigung der Architektur (siehe Kapitel 4) und der bereits vorgestellten Datenkompressionslösung (siehe Kapitel 6), bewertet.

7.1.1 User Datagram Protocol

Der leichtgewichtigste Transportmechanismus basiert auf der Verwendung von reinem UDP. Im Kontext der 6LoWPAN-Protokolle ist hervorzuheben, dass in RFC6282 [73] eine Protokollkompression für UDP beschrieben wird. Neben der Kompression des IPv6 Protokolls ist die UDP-Kompression die einzige in den 6LoWPAN-Protokollen definierte konkrete Anpassung von Protokollköpfen. Beim SOAP-over-UDP Binding wird das SOAP-Dokument direkt als Nutzlast in das UDP-Datagramm eingefügt. Da UDP zustandslos und verbindungslos arbeitet, ist die maximale Nachrichtengröße begrenzt. Die theoretische Grenze von 64 kB wird in stark ressourcenlimitierten Umgebungen aber nur sehr unwahrscheinlich überschritten. Es ist zu beachten, dass unabhängig von der Nachrichtengröße auf Transportebene zusätzlich eine Fragmentierung auf der Vermittlungsschicht erfolgen kann (z.B. durch IPv6 und/oder 6LoWPAN).

Der besondere Vorteil von UDP liegt in der Möglichkeit, nicht nur Unicast, sondern auch IP-Multicast-Adressen ansprechen zu können. IP-Multicast erweist sich immer dann als zweckmäßig und notwendig, wenn der Sender kein Wissen über die Empfänger besitzt. Bei IP-Multicast müssen sich potenzielle Empfänger für den Empfang zuvor bei den zuständigen Routern registrieren. Die Router sind im Anschluss für die Weiterleitung der Nachrichten

zuständig, was keine Empfängerverwaltung beim eigentlichen Sender notwendig macht. DPWS nutzt IP-Multicast unter anderem für die Ankündigung und das Suchen von Geräten, da die Endpunkte den aktuellen Zustand des Netzwerkes und die darin befindlichen Kommunikationsteilnehmer nicht kennen. Auch für das Versenden von asynchronen Benachrichtigungen über Zustandsänderungen eines Gerätes (vgl. Events) an eine Gruppe von Empfängern kann IP-Multicast und somit das UDP-Binding sinnvoll eingesetzt werden (siehe Kapitel 5.2).

Da UDP keinen zuverlässigen Transport gewährleistet, müssen hierfür weitere Mechanismen auf höheren Schichten genutzt werden. Weil UDP nur einzelne Anwendungen, nicht aber Dienste adressieren kann, ist das UDP-Binding zur Unterscheidung einzelner Dienste weiterhin auf Erweiterungen auf höheren Schichten angewiesen. Solche Erweiterungen müssen ebenfalls die Zuordnung von Antworten zu Anfragen gewährleisten, da das UDP-Binding ausschließlich asynchrone Übermittlungen ermöglicht.

7.1.2 Transmission Control Protocol

Microsoft hat im Rahmen der Entwicklung des .NET Frameworks die Web Services Enhancements (WSE) spezifiziert. Ein Bestandteil von WSE ist ein natives TCP-Binding für SOAP, bei dem die Daten ohne ein weiteres Anwendungsschichtprotokoll übertragen werden. Im Vergleich zum nativen UDP-Binding bietet das TCP-Binding einen zuverlässigen Transport Flusskontrolle und keine Beschränkung der Nachrichtengröße durch die Nutzung von Fragmenten. Ein weiterer Unterschied gegenüber reinem UDP ergibt sich aus der Möglichkeit, synchrone und asynchrone Kommunikationsmuster zu nutzen. Bei synchroner Kommunikation wird die bestehende TCP-Verbindung der Anfrage auch für die Antwort genutzt. Bei asynchroner Kommunikation werden für Anfrage und Antwort zwei getrennte TCP-Verbindungen genutzt.

Da auch TCP lediglich in der Lage ist, Anwendungen anhand von Ports zu identifizieren, kann das TCP-Binding nur im Zusammenhang mit Erweiterungen wie WS-Addressing eingesetzt werden, um die Unterscheidung einzelner Dienste vornehmen zu können.

7.1.3 File Transfer Protocol

Das heute weit verbreitete File Transfer Protocol (FTP) [153] wurde bereits 1980 in der ersten Version von der IETF spezifiziert und ist ein grundlegendes Protokoll zur Übermittlung von

Dokumenten, die in Form von Dateien dargestellt werden. Obwohl FTP immer wieder in der Literatur als geeignetes Protokoll zur Übertragung von SOAP-Dokumenten genannt wird, existiert bis zum Zeitpunkt der Entstehung dieser Arbeit keine konkrete Spezifikation.

Der grundlegende Ansatz eines FTP-Bindings besteht darin, SOAP-Nachrichten als einzelne Dateien auf einem FTP-Server abzulegen. Im trivialsten Fall kann somit nur eine Einweg-Kommunikation realisiert werden. Für eine Zweiweg-Kommunikation muss auch der Client über einen FTP-Server verfügen oder der Client muss den FTP-Server regelmäßig auf Antwortdokumente überprüfen (vgl. Polling). Daher ist mit dem FTP-Binding ausschließlich eine asynchrone Kommunikation möglich. Die Zuordnung der SOAP-Dokumente zu den Diensten kann entweder durch den Dateinamen oder durch SOAP-Erweiterungen erfolgen.

Da FTP auf TCP für den Transport aufsetzt, sind die Eigenschaften von TCP impliziert. Darüber hinaus bietet FTP die Möglichkeit der Authentifizierung sowie rollen- und clientspezifische Berechtigungen. Die Berechtigungsverwaltung auf dem FTP-Server kann durch die Nutzung entsprechend hierarchischer Ordnerstrukturen und feingranularer Rechtevergabe erfolgen.

7.1.4 E-Mail

Im Zuge der Entwicklung des Internets wurde mit dem Simple Mail Transfer Protocol (SMTP), welches in der aktuellsten Version in RFC2821 [154] bei der IETF spezifiziert ist, die Abhängigkeit der E-Maildienste vom FTP-Protokoll beseitigt. Der allgemeine Begriff E-Mail wird dabei durch weitere Protokolle, wie das Post Office Protocol (POP) oder Internet Message Access Protocol (IMAP) gestützt.

Das W3C hat die Situation der weiten Verbreitung E-Mailbasierter Kommunikation aufgegriffen und ein entsprechendes Binding für SOAP in [155] spezifiziert. Dabei wird vorrangig das Nachrichtenformat definiert. Über welchen Mechanismus und welche Infrastruktur (z.B. SMTP, POP, IMAP usw.) das Format übertragen wird, ist nicht vorgeschrieben.

Durch den grundlegenden asynchronen Charakter des E-Mailaustausches ermöglicht das E-Mail Binding keine synchrone Kommunikation zwischen Dienstnutzer und Dienstanbieter. Im Gegensatz zum FTP-Binding bietet das E-Mailbinding zwei Vorteile. Zum einen können Punkt-zu-Mehrpunkt Verbindungen (siehe Gruppenkommunikation) auf Anwendungsschicht erreicht werden, da als Empfänger von E-Mails mehrere Adressen eingetragen werden können.

Darüber hinaus ist der Nachrichtenkopf des E-Mail-Formates flexibel erweiterbar. Dadurch kann auf Erweiterungen wie WS-Addressing in vielen Szenarien verzichtet werden und die notwendigen Steuerinformationen direkt in den E-Mail-Nachrichtenkopf integriert werden.

7.1.5 Extensible Messaging and Presence Protocol

Aus dem ursprünglichen Instant Messaging Protokoll Jabber ist im Laufe der Entwicklung das Extensible Messaging and Presence Protocol (XMPP) hervorgegangen. Die Bezeichnung XMPP wurde vornehmlich durch die Standardisierung des Kernprotokolls [118] bei der IETF geprägt. Erweiterungen zu XMPP werden von der XMPP Standards Foundation entwickelt und gepflegt. Obwohl ursprünglich für den Einsatzzweck des Instant Messagings konzipiert kann XMPP in der aktuellen Form als allgemeines Protokoll für den Austausch von strukturierten Daten genutzt werden. Die strukturierte Darstellung wird bei XMPP mittels XML-Strukturen abgebildet. Einzelne Informationseinheiten eines XMPP Datenstroms werden als Stanzas bezeichnet. Für das SOAP-Binding können Stanzas ein komplettes SOAP-Dokument als Inhalt enthalten.

Jeder XMPP-Endpunkt verbindet sich, wie in Abbildung 7.2 dargestellt, mit genau einem Server. Zur Übertragung der Daten werden eine oder mehrere TCP-Verbindungen zu diesem Server

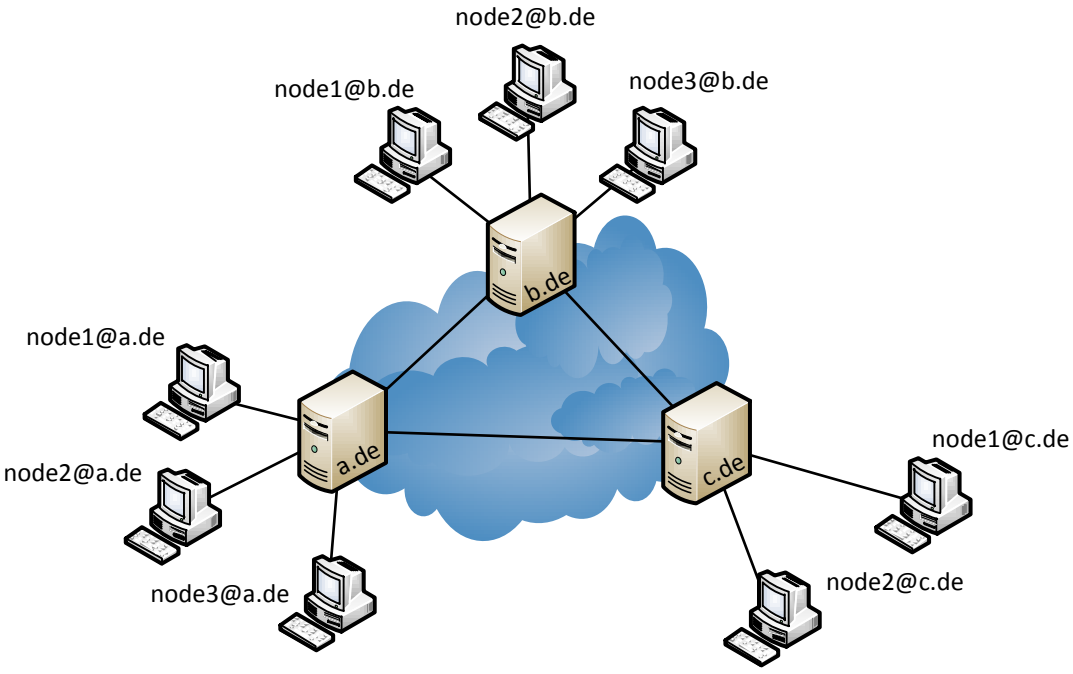

Abbildung 7.2 Netzwerktopologie XMPP

genutzt, die dauerhaft bestehen bleiben und nach einer Anfrage nicht geschlossen werden. Die Server eines komplexeren XMPP-Netzwerkes sind untereinander ebenfalls verbunden und ermöglichen damit den Austausch von Daten mit Endpunkten, die nicht mit dem gleichen Server verbunden sind. Daher wird XMPP oft auch als XML-basiertes Routingprotokoll bezeichnet, da die Server die Weiterleitung der Nachrichten anhand der Adressierung in den Datenfragmenten der Anwendungsschicht vornehmen.

XMPP unterstützt offiziell keine anderen Darstellungen als Unicode XML. Obwohl XMPP nicht XML-Infoset unterliegt, ist es dennoch grundlegend mit EXI kompatibel. Ein entsprechendes Dokument zu EXI-basierter Kodierung des Datenstroms findet sich bereits bei der XMPP Standards Foundation. Allerdings ist das Dokument bisher lediglich ein Entwurf und noch keine offizielle Erweiterung.

Das SOAP-over-XMPP Binding [156] hingegen gehört zu den bereits akzeptierten und veröffentlichten Erweiterungen. XMPP ermöglicht dabei asynchrone und synchrone Kommunikation. Da das XMPP-Binding eine ausgehende Verbindung auch für eingehende Kommunikation verwendet, bietet es vor allem in Szenarien Vorteile, bei denen eine Kommunikation zwischen Endpunkten erfolgen soll, die von außen nicht durch öffentliche Adressierung erreicht werden können. Beispiele hierfür sind Netzwerke, die Network Address Translation (NAT) oder Firewalls nutzen. Das XMPP-Binding ist dabei deutlich leichtgewichtiger als bestehende SOAP-Erweiterungen (z.B. WS-Routing oder WS-Referral), bietet aber dennoch die bereits in XMPP enthaltenden Funktionalitäten wie Authentifizierung und Kanalverschlüsselung.

7.1.6 Hypertext Transfer Protocol

Das Hypertext Transfer Protocol (HTTP) ist vornehmlich durch die Verbreitung bei browserbasierten Interaktionen mit Internetseiten bekannt. Die aktuellste Version von HTTP ist als RFC2616 [74] bei der IETF spezifiziert, wird aber durch diverse weitere Dokumente erweitert, die nicht alle bei der IETF gepflegt werden. Durch die Dissertation von Roy Fielding [39], einer der Autoren der HTTP-Spezifikation und Mitbegründer der Apache Foundation [157], wurde HTTP auch als Protokoll bekannt, welches den Prinzipien des REST-Architekturstils folgt.

Obwohl HTTP als eigenständiges Anwendungsschichtprotokoll konzipiert ist, kann es ebenfalls durch weiterführende Anwendungsschichtprotokolle für den Transport der Daten dieser Protokolle genutzt werden. Dieser Mechanismus wird in der Literatur oft als Tunnel bezeichnet. Solche

HTTP-Tunnel sind von jeglichen Basisarchitekturen und Architekturprinzipien, wie beispielsweise SOA oder REST, losgelöst zu betrachten, da sie HTTP entgegen der ursprünglichen Konzeption lediglich als Transportmechanismus nutzen. Der anfängliche Vorteil des Transports von anderen Protokollen über HTTP lag darin, dass beispielsweise Firewalls die HTTP-Datentransfers über die dazugehörigen Ports nicht blockieren. Neuere Firewalls sind aber in der Lage, Daten und Datenströme deutlich feingranularer zu untersuchen, Protokolltunnel zu erkennen und ggfs. darauf zu reagieren.

Einer der bekanntesten HTTP-Transporttunnel ist das SOAP-over-HTTP Binding [150]. Obwohl auch SOAP-over-HTTP ursprünglich dafür genutzt wurde, SOAP durch Firewalls hindurch zu transportieren, liegt der aktuelle Vorteil vornehmlich in der großen Verbreitung. Verglichen mit der Gesamtheit der Protokollmechanismen in HTTP ist die Nutzung von HTTP für den Transport von SOAP vergleichsweise einfach. Hierzu zählt beispielsweise die ausschließliche Nutzung der HTTP-Methode POST, was ebenfalls die Einschränkung auf eine Untermenge der HTTP-Response-Codes impliziert. Dennoch nutzt das Binding HTTP auch zur Übermittlung von Steuerinformationen. Hierzu gehört die Adressierung der Dienste im HTTP-Kopf, was SOAP-Erweiterungen wie WS-Addressing nicht zwingend erforderlich macht. Auch Rückmeldungen über Erfolg bzw. Fehler können z.B. durch die HTTP-Response-Codes angezeigt werden. Der HTTP-Kopf wird zwingend im ASCII-Zeichensatz kodiert. Diese Einschränkung gilt aber nicht für die zu übermittelnden Nutzdaten, die in beliebigen Formaten kodiert werden können. Da SOAP-Dokumente beim HTTP-Binding als Nutzdaten übertragen werden, sind somit neben Unicode XML auch effizientere Kodierungen, wie z.B. EXI (siehe Kapitel 6), möglich. Die typische minimale Größe des HTTP-Protokollkopfes bei der Nutzung von HTTP als SOAP-Transportmechanismus beträgt ca. 100-200 Byte.

Das HTTP-Binding eignet sich besonders für Unicast-Kommunikation. Beim Anfrage-Kommunikationsmuster wird durch die Antwort auf HTTP-Ebene im Gegensatz z.B. zum einen UDP-Binding dennoch eine Statusmeldung zurückgeliefert. Beim Anfrage-Antwort-Muster können die SOAP-Dokumente direkt in die entsprechenden Pendants auf HTTP-Ebene eingebettet werden. Da HTTP selbst aus einer Anfrage und einer Antwort besteht und in der Lage ist, Dienste direkt im HTTP-Kopf zu adressieren, kann mittels HTTP ein weiteres drittes Kommunikationsmuster realisiert werden. Dieses dritte Muster wird als Antwort-Muster bezeichnet und besteht im Gegensatz zum Anfrage-Muster auf SOAP-Ebene nicht aus einer einzelnen

Anfrage, sondern aus einer einzelnen Antwort. Die einzelne Antwort auf SOAP-Ebene erfordert eine Anfrage außerhalb von SOAP, wie es mit nativem HTTP möglich ist.

In der Spezifikation von SOAP-over-HTTP wird ausschließlich synchrone Kommunikation unterstützt. HTTP bietet durch den Keep-Alive-Mechanismus die Möglichkeit, die bestehende TCP-Verbindung nach der Antwort aufrecht zu erhalten und für weitere Kommunikation zu nutzen. Eingesetzt wird diese Funktion insbesondere zum Abrufen von Internetseiten durch HTTP, da diese aus einer Vielzahl von einzelnen Elementen bestehen und der Auf- und Abbau von TCP-Verbindungen für jedes einzelne Element einen deutlichen Overhead bedeuten würde. Diese Funktion bedarf aber des Einverständnisses beider Kommunikationspartner. Ein Endpunkt kann eine Anfrage zur Aufrechterhaltung des Kommunikationskanals ablehnen, z.B. aufgrund von Ressourcenknappheit.

7.1.7 UDP-basierte Erweiterungen

Die bisher genannten Protokolle transportieren SOAP-Dokumente auf diverse Arten. Außer UDP und TCP sind dabei alle anderen genannten Protokolle ursprünglich für andere Anwendungszwecke als den Transport bzw. das Tunneln anderer Protokolle entwickelt worden. Zwar sind diese Protokolle teilweise flexibel genug, dahingehend erweitert zu werden, auch Dienste zu adressieren, ohne dabei SOAP-Erweiterungen zu nutzen. Dennoch implizieren diese Lösungen einen grundlegenden Mehraufwand im Hinblick auf Komplexität und Datenlast.

Werner et al. haben mit PURE [159] eine Lösung entwickelt, die speziell auf die Bedürfnisse von SOAP Web Services zugeschnitten ist. Da auch Werner et al. ihre Lösung vor dem Hintergrund ressourcenlimitierter Umgebungen entwickelt haben, stützt sich PURE auf Transportebene auf das leichtgewichtige UDP. Mithilfe eines fünf Byte großen Protokollkopfes ist PURE in der Lage,

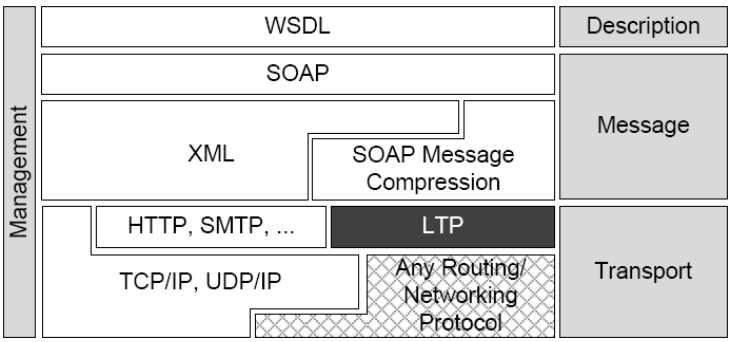

Abbildung 7.3 Lean Transport Protocol nach Glombitza et al. [158]

einfache Basismechanismen wie Nachrichtenfragmentierung, Erkennung von Duplikaten und Empfangsbestätigungen zu realisieren. Dadurch gleicht PURE die Nachteile von UDP gegenüber TCP mit leichtgewichtigen Mechanismen auf Anwendungsschicht aus.

Eine ähnliche Lösung wie PURE wurde von Gehlen et al. in [160] vorgestellt. Die Lösung von Gehlen et al. zielte aber vornehmlich auf den Einsatz im Mobilfunkbereich ab. Glombitza et al. haben den PURE Ansatz in [158] ebenfalls weiterentwickelt. Der Vorteil des von Glombitza et al. entwickelten Lean Transport Protocols (LTP) liegt in der möglichen Vermeidung von UDP, TCP und IP auf unteren Schichten. LTP kann grundsätzlich auch über andere proprietäre Protokolle übertragen werden (siehe Abbildung 7.3).

7.1.8 Enterprise Infrastrukturen

Neben den bisher genannten Transportmechanismen existieren eine Reihe weiterer Enterprise-Infrastrukturen, die grundsätzlich ebenfalls in der Lage sind, SOAP-Dokumente auszutauschen. Beispiele hierfür sind WebSphere MQ von IBM [161], Message Queuing von Microsoft (MSMQ) [162] und Java Messaging Service (JMS) [163]. Diese Infrastrukturen sind deutlich komplexer und teilweise von spezifischen Plattformen oder Programmiersprachen abhängig. Im Kontext des Einsatzes in stark ressourcenlimitierten und heterogenen Umgebungen können solche Infrastrukturen daher nicht weiter betrachtet werden.

7.2 Bewertung existierender Transportmechanismen

In Anwendungen mit SOAP Web Services, die nicht im gerätenahen Umfeld liegen, erzeugen die in Unicode kodierten Nachrichten in der Regel den größten Protokolloverhead. Dies ist durch die Komplexität der angebotenen Dienste und der damit einhergehenden komplexen Dokumentenstruktur bedingt. Der entstehende Overhead wird infolge einfacherer Dienste im gerätenahen Umfeld leicht verringert, bleibt aber grundlegend erhalten und hat den größten Einfluss auf die Datenlast. Mit der Hilfe von neuartigen Darstellungsformen können native XML-Dokumente deutlich effizienter dargestellt werden (siehe Kapitel 6). Die resultierenden Nachrichtengrößen der optimierten Darstellungsformen erreichen dabei deutlich unter 100 Byte je Nachricht und erhebliche Kompressionsraten gegenüber nativem Unicode kodiertem XML. Unter Berücksichtigung kompakter und optimierter Darstellungsformen der SOAP-Dokumente gewinnen zunehmend auch die unterhalb von SOAP liegenden Protokollschichten an Bedeutung (z.B. IP,

UDP, TCP...). Sie haben somit einen essenziellen Einfluss auf die entstehenden, zu übermittelnden Datenpakete, die Leistung und die Effektivität der Gesamtlösung.

Für die Nutzung von SOAP und DPWS im Kontext von stark ressourcenlimitierten Umgebungen ist eine möglichst geringe Paketgröße nötig, um die Sende- und Empfangseinheiten auf den Plattformen nur für möglichst kurze Zeiten aktivieren zu müssen. Weiterhin sind unzuverlässige Kommunikationskanäle sowie eine hohe Verzögerungszeit für Ende-zu-Ende Verbindungen in diesem Zusammenhang hervorzuheben. Ein effizienter Transportmechanismus für SOAP-Dokumente muss alle diese Anforderungen beachten.

In den nachfolgenden Abschnitten wird die Effizienz der vorgestellten Transportmechanismen untersucht. Da diese grundsätzlich auf IP auf der Adressierungsschicht aufsetzen und unabhängig gegenüber der Version von IP (IPv4 oder IPv6) sowie der darunterliegenden Zugriffsschicht sind, werden die unterhalb der Transportschicht liegenden Ebenen nicht weiter betrachtet. Der Einfluss dieser unteren Schichten ist auf alle Transportmechanismen identisch.

7.2.1 Bewertung TCP-basierter Protokolle

Das verbindungsorientierte TCP führt jeweils für den Aufbau und den Abbau einer Transportverbindung einen Drei-Wege-Handshake durch. Beim Verbindungsaufbau wird so sichergestellt, dass die Gegenstelle bereit ist, Daten zu empfangen. Beim kontrollierten Verbindungsabbau wird durch den Handshake sichergestellt, dass alle Daten korrekt empfangen wurden und keine weiteren Daten gesendet werden sollen. Jede der Nachrichten zur Steuerung der Verbindung enthält einen vollständigen TCP-Protokollkopf und umfasst somit alleine auf der Transportschicht typischerweise 20 Byte (siehe Abbildung 7.4). Tritt während des gesamten Prozesses kein Fehler auf, sind im schlechtesten Fall bis zu acht Nachrichten und somit mindestens 160 Byte (vgl. <100 Byte für SOAP durch EXI) auf Transportschicht nötig, um ein einzelnes Paket in eine Richtung zu versenden. In der Praxis werden einige der Kontrollnachrichten, die für den Verbindungsaufbau bzw. -abbau benötigt werden, mit Nachrichten für den Datentransfer zusammengelegt. Dennoch sind nicht weniger als sechs Nachrichten möglich, was der Anzahl der Kontrollnachrichten der Drei-Wege-Handshakes entspricht (sechs Pakete sind bereits ein Sonderfall, bei dem die Verbindung simultan von beiden Endpunkten beendet wird). Die Kontrollnachrichten müssen dabei in der korrekten Reihenfolge gesendet werden und dürfen sich nicht überlagern bzw. durch unterschiedliche Routingpfade überholen. Vor dem Senden der

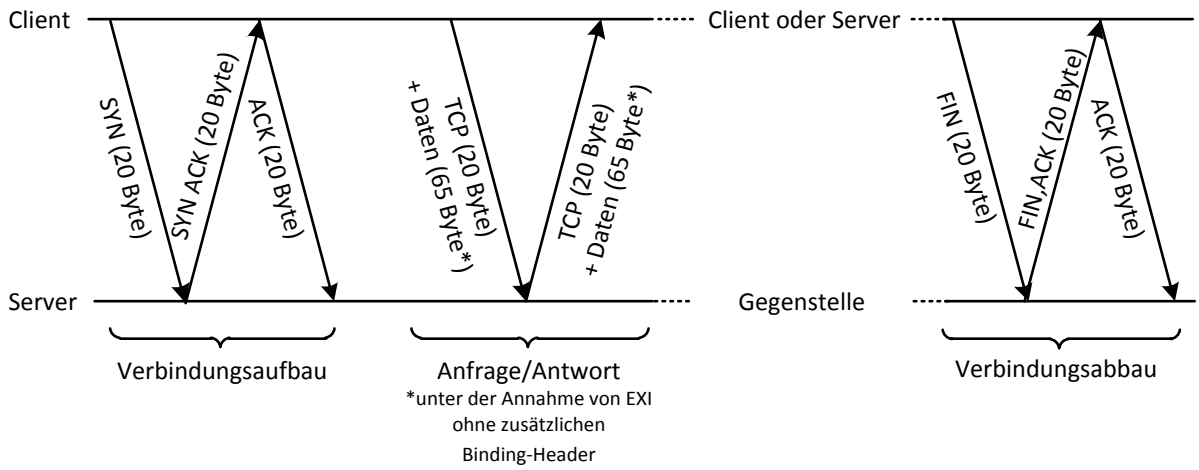

Abbildung 7.4 Ablauf einer SOAP-Anfrage mittels TCP-basiertem Binding

nächsten Kontrollnachricht muss auf die Antwort der vorherigen Nachricht von der Gegenstelle gewartet werden. In Netzwerken mit hohen Verzögerungszeiten sind die vergleichsweise komplexen Mechanismen für den Verbindungsaufbau und –abbau nicht geeignet. Bei einem einfachen Einweg-Datenaustausch von einer vergleichsweise geringen Datenmenge impliziert die Nutzung von TCP auf Transportebene damit einen nicht akzeptablen Mehraufwand.

Existierende Transportmechanismen für SOAP verwenden zum Teil neben TCP weiterführende Anwendungsschichtprotokolle (z.B. HTTP, XMPP), um die in TCP fehlenden Adressierungsmöglichkeiten einzelner Dienste zu ermöglichen, ohne sich dabei auf schwergewichtige Erweiterungen von SOAP zu stützen. Diese Protokolle sind in der Lage, eine TCP-Verbindung für mehrere aufeinander folgende Aufrufe zu nutzen, wodurch nur einmalig ein Verbindungsaufbau notwendig ist. Beide Endpunkte müssen dabei die Verbindungen dauerhaft verwalten, was bei speicherlimitierten Plattformen die Anzahl der möglichen gleichzeitigen Verbindungen zu stark einschränkt. Alternativ ist es möglich, eine persistente TCP-Verbindung zu einem einzelnen Endpunkt zu nutzen. Dieser Endpunkt leitet die Nachrichten entsprechend an den oder die Empfänger weiter (siehe SOAP-over-XMPP). Eine solche zentrale Instanz kann im Falle einer Fehlfunktion aber den Defekt des gesamten Netzwerkes bedeuten. Weiterhin ist eine direkte Kommunikation zwischen zwei direkt benachbarten Endpunkten eines vermaschten Netzwerkes nicht möglich und verursacht unter Umständen Datenverkehr für Netzwerkteilnehmer, die nicht direkt an der Kommunikation beteiligt sein müssten. Es gilt ebenfalls zu untersuchen, ob sich persistente TCP-Verbindungen für den Einsatz in Umgebungen mit stark fehleranfälligen Kommunikationskanälen und sich dynamisch ändernden Routen eignen.

7.2.2 Bewertung UDP-basierter Protokolle

Im Gegensatz zu TCP ist UDP vollständig verbindungslos und bedarf somit keiner Zustandsverwaltung zwischen den Aufrufen. Handshakemechanismen für den Verbindungaufbau, Verbindungsabbau und Quittierungsmechanismen sind mit nativem UDP nicht nötig. Der Nachteil liegt somit in der Unzuverlässigkeit des Transports. Solche Funktionen müssen durch die höheren Schichten direkt realisiert werden. Der Vorteil von UDP besteht in der Möglichkeit, IP-Multicastadressen ansprechen zu können. Es existieren UDP-konforme Erweiterungen, wie das NACK-Oriented Reliable Multicast (NORM) Transport Protocol [164], die in der Lage sind, auch Multicastkommunikation zuverlässig zu gestalten. Diese Erweiterungen haben in bestehenden Netzwerken aber bisher keine nennenswerte Durchdringung erfahren.

Bei dem einfachen Anfrage-Kommunikationsmuster von SOAP wird vom Dienst keine weitere Antwort gesendet. Aus diesem Grund muss für die Einweg-Kommunikation bei nativem UDP meist auf redundante Übermittlung der gleichen Nachricht zurückgegriffen werden, um die Übermittlungswahrscheinlichkeit zu erhöhen. Da die redundanten Nachrichten von den drahtlosen Sensorknoten erst nach dem Empfang als bereits verarbeitet erkannt werden können, steigt der Mehraufwand direkt proportional mit der Anzahl der Wiederholungen. Dies wiederum erhöht den Energieverbrauch unnötig.

7.2.3 Gegenüberstellung TCP-basierter und UDP-basierter Bindings

Um die Effizienz der existierenden Bindings bewerten zu können, wurden die beiden Transportprotokolle UDP und TCP genauer evaluiert. Batteriebetriebene Plattformen erreichen nur dann einen geringen Energieverbrauch, wenn sie die Sende- und Empfangshardware möglichst oft und lange deaktivieren können und alle nicht benötigten Module in einen Energiesparmodus versetzen (siehe Kapitel 3). Damit besteht ein direkter Zusammenhang zwischen dem Energieverbrauch der Plattformen und der Zeit, um einen kompletten Aufruf zu verarbeiten und die Antwort zu senden. Die Effizienz bezieht sich in diesem Zusammenhang somit auf die Zeit, um einen vollständigen Aufruf (bestehend aus Anfrage und Antwort) zu übermitteln.

Ziel der Untersuchungen ist es zu bewerten, wie stark sich die Handshakes von TCP und die Größe der zu übertragenden Nachrichten auf Anwendungsschicht auf die Effizienz auswirken. Hierzu wurde ein Experiment durchgeführt, bei dem Daten von einem PC an einen 6LoWPAN-Sensorknoten gesendet wurden. Der Sensorknoten hat die Daten empfangen und sendet

diese nach dem Empfang ohne weitere Verarbeitung vollständig an den PC zurück. Somit tritt keine Verarbeitungszeit auf dem Sensorknoten auf. Auf dem PC wurde die Zeit erfasst die nötig war, den gesamten Aufruf durchzuführen. Diese Zeit wird als Umlaufzeit (engl. Round Trip Time) bezeichnet. Der PC und der Sensorknoten befanden sich in direkter Funkreichweite, was einem einzelnen Hop zwischen den Endpunkten entspricht. Somit war der PC zugleich der Border Router des 6LoWPANs. Die Voraussetzung eines einzelnen Hops ist nötig, da ausschließlich UDP und TCP gegenübergestellt werden sollen. Bei Multihop-Kommunikation haben weitere Parameter, wie beispielsweise das genutzte Routingprotokoll, einen zu starken Einfluss auf die Gesamteffizienz. Optimierungen für Multihop-Verbindungen haben unter anderem Dunkels et al. in [119] diskutiert. Als Sensorknoten wurde die TelosB-Plattform genutzt, auf dem Contiki [98] in der Version 2.5 implementiert wurde.

Für die Messungen wurden die Puffer der Netzwerkstacks sowie das TCP-Window so gewählt, dass die Daten in ein TCP-Paket passen und Fragmentierung lediglich durch IPv6 bzw. die 6LoWPAN-Protokolle erfolgt. Wie in Kapitel 6 dargestellt ist, sind durch optimierte Darstellungsformen Nachrichtengrößen für SOAP-Dokumente unter 10 % der minimalen IPv6 MTU von 1280 Byte realistisch zu erreichen. Übertragungsfehler sind nicht Bestandteil dieser Betrachtungen. Es gilt aber zu beachten, dass Korrekturmechanismen von TCP in drahtlosen Sensornetzwerken grundsätzlich nicht optimal funktionieren. Das Design der Korrekturmechanismen von TCP geht davon aus, dass Fehler nur auftreten, wenn Router die Pakete z.B. durch zu geringe Ressourcen verwerfen müssen. In drahtlosen Sensornetzwerken treten Fehler aber meist durch den unzuverlässigen Kommunikationskanal selbst auf. Weiterführende Informationen hierzu finden sich in [45].

Die Ergebnisse der experimentellen Messungen sind in Abbildung 7.5 und Abbildung 7.6 dargestellt. In Abbildung 7.5 ist die benötige Zeit für den gesamten Aufruf über der Nachrichtengröße aufgetragen, die in Abständen von je 10 Byte erhöht wurde.

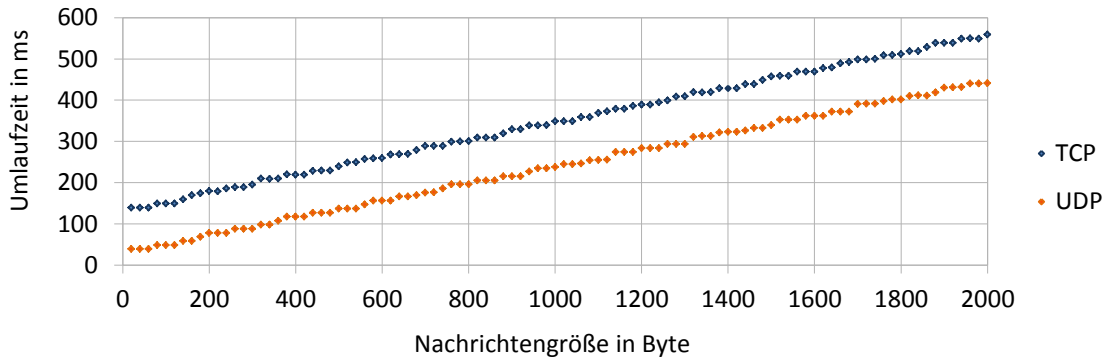

Abbildung 7.5 Messungen Umlaufzeit von UDP und TCP

Die Ergebnisse der linearen Approximation aus Abbildung 7.5 durch eine Regressionsgerade können wie folgt dargestellt werden:

$$Zeit\ in\ ms = 0{,}2076\,\frac{ms}{Byte} \times Bytes + 30{,}468\,ms$$

Formel 7.1 Regression TCP

$$Zeit\ in\ ms = 0{,}2127\,\frac{ms}{Byte} \times Bytes + 133{,}028\,ms$$

Formel 7.2 Regression UDP

Die Datenraten von UDP und TCP sind nahezu identisch. Der Versatz der beiden Kennlinien entsteht durch den Handshakemechanismus, welchen TCP zum Aufbau der Verbindung benötigt. Die resultierenden Kennlinien sind stark von Faktoren, wie beispielsweise den Hardwaretreibern auf dem PC, abhängig. Aus diesem Grund stellt die Abbildung 7.6 jeweils das Verhältnis von UDP zu TCP dar, was einen unabhängigen Vergleich ermöglicht. Vor allem bei kleinen Nachrichtengrößen benötigt UDP für die gesamte Übertragung lediglich etwa ein Drittel der Zeit von TCP. Bei einer Nachrichtengröße von ca. 350 Byte beträgt die benötigte Zeit noch etwa die Hälfte.

Bezugnehmend auf Kapitel 6 ist zu beobachten, dass ohne Komprimierung der Nachrichtengröße der Nachteil von TCP durch den kontrollierten Verbindungsaufbau bzw. -abbau vernachlässigt werden kann. TCP und UDP benötigen für diesen Fall der Nutzung von Unicode XML und somit Nachrichtengrößen von mehr als 1.000 Byte in etwa die gleiche Übertragungszeit. Für effizientere Darstellungsformen, die bis zu unter 100 Byte betragen (z.B. mittels EXI), ist der Vorteil allerdings immens und UDP-basierte Protokolle sind klar zu bevorzugen. Ausgenommen hiervon sind persistente TCP-Verbindungen, die allerdings aufgrund der benötigen Ressourcen zur Verwaltung der Verbindung ungeeignet sind.

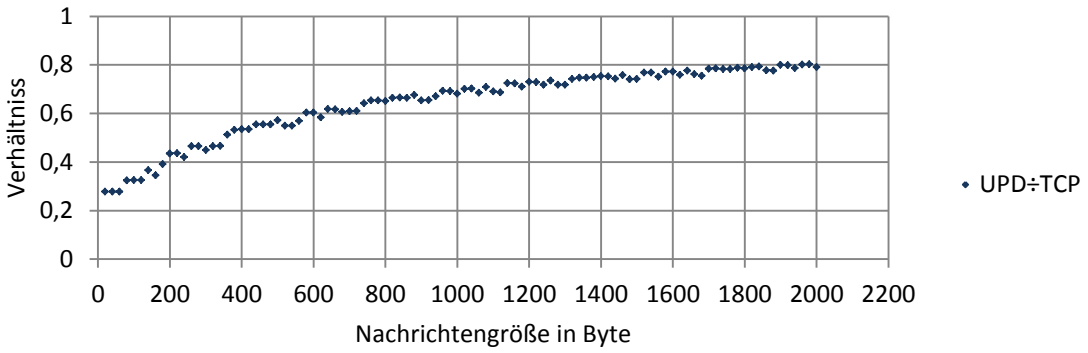

Abbildung 7.6 Verhältnis Umlaufzeiten UDP zu TCP

Resultierend aus den Untersuchungen lässt sich feststellen, dass nur UDP-basierte Bindings in Verbindung mit sehr geringen Nachrichtengrößen als SOAP-Transportmechanismus eine hohe Effizienz in 6LoWPANs erreichen. In [115] wurde von Kovatsch et al. herausgearbeitet, dass weiterhin die Anzahl der zu übertragenden 6LoWPAN-Fragmente keinen merklichen Einfluss hat. Kovatsch et al. haben in ihren Untersuchungen gezeigt, dass bezüglich der Effizienz lediglich relevant ist, ob die zu übermittelnden Daten größer als ein Paket auf der Zugriffsschicht sind und somit fragmentiert werden muss oder nicht. Sobald zur Erreichung der minimalen MTU von IPv6 Fragmentierung durch die 6LoWPAN-Protokolle nötig ist, spielt die Anzahl der Fragmente eine untergeordnete Rolle. Die benötigte Zeit für die Übermittlung der Daten steigt dann proportional mit der Nachrichtengröße.

Im Hinblick auf die Effizienz und die fehlende Transportsicherheit von reinem UDP sind von den in Kapitel 7.1 vorgestellten existierenden SOAP-Transportbindings daher lediglich die UDP-basierten Erweiterungen, wie beispielsweise PURE von Werner et al., als geeignet zu bewerten. Die genannten Bindings bilden eigenständige, proprietäre Ansätze und basieren nicht auf Standards. Eine Anlehnung bzw. Ähnlichkeit an existierende Technologien und Konzepte ist nicht vorzufinden, wodurch diese Lösungen nicht zustandslos in existierende Protokolle umgesetzt werden können. Aufgrund dieses grundlegenden Designs fügen sich die Lösungen nicht in die angestrebte Gesamtarchitektur ein (siehe Kapitel 4), bei der Anpassungen nur das 6LoWPAN selbst betreffen und eine nahtlose und transparente Umsetzung in die existierenden und verbreiteten Protokolle durch zustandslose Proxys erfolgt. Die aufgeführten Ansätze PURE und LTP sind somit hinsichtlich der Ressourcen geeignet, benötigen aber im schlechtesten Fall schwergewichtige Gateway-Architekturen zur Einbindung in höherwertige Dienstkompositionen.

In existierenden Anwendungen basiert die Beschränkung von SOAP auf die beiden verbreitetsten Bindings UDP und HTTP auf der erforderlichen Interoperabilität. HTTP und natives UDP sind so allgegenwärtig, dass sie fast immer die bevorzugte Wahl für den SOAP-Nachrichtentransport darstellen. Jede Erweiterung für 6LoWPAN-Netzwerke muss der Anforderung einer nahtlosen und zustandslosen Umsetzung in mindestens eines der beiden Protokolle genügen. Eine solche Lösung ist bislang nicht vorzufinden.

7.3 Konzeption und Spezifikation eines CoAP-basierten SOAP-Bindings

Bei der IETF wird derzeit ein Protokoll entwickelt, welches den beiden Kriterien Ressourceneffizienz und Einbindung in die angestrebte Gesamtarchitektur gleichermaßen genügt. Das Protokoll wird bei der IETF durch die Constrained RESTful Environments (CoRE) Arbeitsgruppe entwickelt und heißt Constrained Application Protocol (CoAP). Das in diesem Abschnitt konzipierte vollständig neuartige Binding nennt sich SOAP-over-CoAP. Im Rahmen der Standardisierung von CoAP wurde das hier vorgestellte SOAP-over-CoAP Binding vom Autor in der IETF Arbeitsgruppe CoRE als Spezifikationsentwurf eingereicht. Ergebnis der Einreichung des Entwurfes war die Einflussnahme auf CoAP in einer frühen Phase der Entstehung, um das angestrebte CoAP-Binding zu ermöglichen. Konkrete Änderungen an CoAP beziehen sich beispielsweise auf die Verwendung der Methode POST und damit einhergehend der Mechanismen zur Fragmentierung von Nutzdaten durch CoAP auf Anwendungsschicht. Die Ergebnisse dieser Arbeit sind somit direkt in die Spezifikation von CoAP eingeflossen.

7.3.1 Einführung in CoAP

CoAP ist ein Anwendungsschichtprotokoll (siehe Kapitel 3.1.5), welches für ressourcenlimitierte

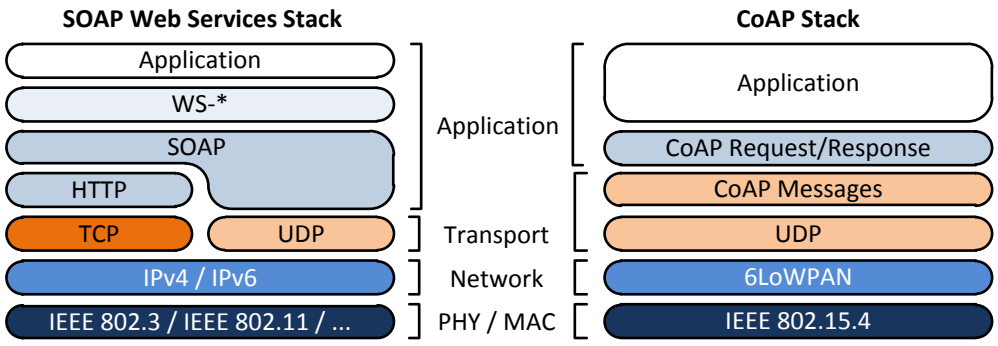

Abbildung 7.7 Gegenüberstellung SOAP Web Services und CoAP Protokollstapel

Umgebungen vorgesehen ist und den REST-Prinzipien in weiten Teilen entspricht [165]. Ein Vergleich mit einem typischen SOAP Web Services Protokollstapel ist in Abbildung 7.7 grafisch dargestellt. Grundsätzlich ist CoAP aber von den unteren Protokollen unabhängig und kann somit unabhängig von den 6LoWPAN-Protokollen eingesetzt werden.

CoAP lehnt sich stark an HTTP an und verfügt über einen ähnlichen grundlegenden Aufbau (siehe Tabelle 7.1). Um auch in ressourcenlimitierten Umgebungen eingesetzt werden zu können, wird

	HTTP	**CoAP**
Rollenmodell	Client/Server	Client/Server
Kommunikationsmuster	Request/Response	Request/Response
Transport	TCP	UDP Zuverlässigkeit optional durch CoAP Messages
Übertragung	Unicast, Point-to-Point	Unicast, Multicast, Point-to-Point, Point-to-Multipoint
Nachrichtenaustausch	vornehmlich synchron	synchron, asynchron
Dienst-/Datenmodellierung	Ressourcen	Ressourcen
Dienst-/Datenadressierung	URI	URI
Methoden	GET, PUT, POST, DELETE, TRACE, HEAD, OPTIONS, CONNECT	GET, PUT, POST, DELETE
Media-Types	Text, JSON, XML, EXI,…	Text, JSON, XML, EXI,…
Response-Codes	Informational 1xx Successful 2xx Redirection 3xx Client Error 4xx Server Error 5xx	Success 2.xx Client Error 4.xx Server Error 5.xx
Header-Kodierung	ASCII	Binär
Proxy und Caching	HTTP zu HTTP	CoAP zu CoAP HTTP zu CoAP CoAP zu HTTP
Formale-Beschreibung	WADL	WADL
Ressourcen-Beschreibung	HTTP Link Header Format	CoRE Link Format
Ressource-Discovery	n.a.	CoAP, CoAP Link Format
Eventing (vgl. Push)	Long Polling/Comet, Web Sockets, BOSH,…	CoAP Observe
Security	SSL/TLS	DTLS

Tabelle 7.1 Gegenüberstellung HTTP und CoAP

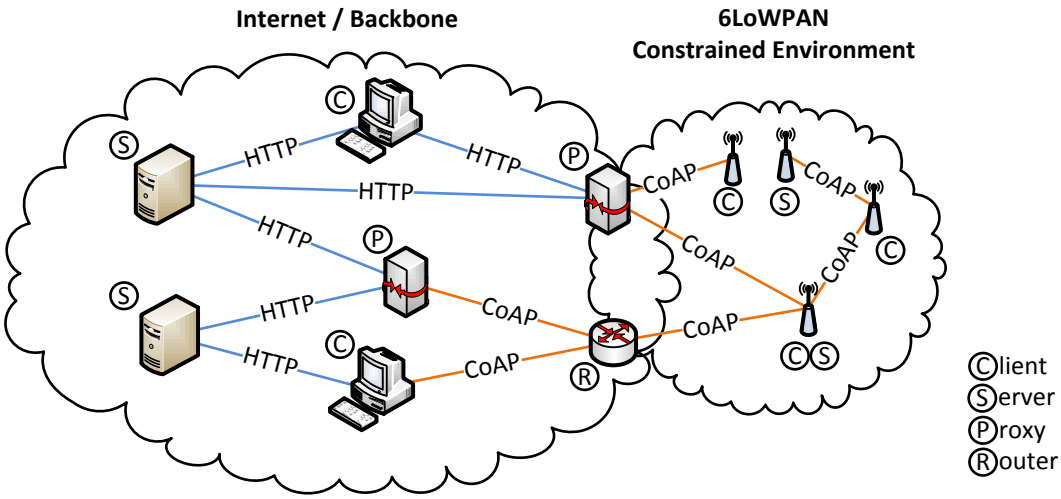

Abbildung 7.8 Umsetzung zwischen CoAP und HTTP mittels Proxy

der Protokollkopf in CoAP im Gegensatz zu HTTP binär kodiert, wodurch die benötigte Bandbreite deutlich sinkt. Ein typischer CoAP-Protokollkopf umfasst ca. 4-20 Byte. CoAP unterscheidet sich von HTTP auf Transportschicht ebenfalls in der Nutzung des leichtgewichtigeren UDP anstelle von TCP. Daher wurden in CoAP Mechanismen zur Herstellung der Transportverlässlichkeit zusätzlich mittels der *CoAP-Messages*-Schicht integriert. Diese Schicht dient neben dem Herstellen einer Transportzuverlässigkeit ebenfalls der Duplikatserkennung und Fragmentierung von Nutzdaten auf Anwendungsschicht. Bei HTTP werden diese Funktionen durch TCP und darunterliegende Schichten realisiert.

Abstrakt betrachtet kann CoAP somit als binäres HTTP angesehen werden, dass die fehlenden Funktionen von TCP auf Anwendungsschicht realisiert und zusätzlich zu HTTP rudimentäre Funktionen für Discovery und Eventing enthält.

Durch die große Überlappung bezüglich der Protokollmechanismen ist es möglich, CoAP zu großen Teilen direkt in natives HTTP umzusetzen. Hierfür wurden in CoAP spezielle Proxymechanismen definiert. Der Grundgedanke ähnelt dabei dem von 6LoWPAN. Innerhalb des 6LoWPANs können effizientere und optimierte Protokolle genutzt werden, die beim Übergang in ein anderes Netzwerk mittels Proxys in ihre nativen Pendants transparent und zustandslos überführt werden können. Eine mögliche Netzwerkinfrastruktur stellt Abbildung 7.8 vor.

7.3.2 Grundlegende Nutzung von CoAP als SOAP-Binding

Bei der Nutzung von CoAP als Transportmechanismus werden die beiden schwergewichtigeren Protokolle TCP und HTTP durch die leichtgewichtigeren Alternativen UDP und CoAP vollständig ersetzt (siehe Abbildung 7.9). Die meisten benötigen Protokollmechanismen für die Interaktion von Geräten und Diensten sind bereits durch SOAP, DPWS und die WS-* Spezifikationen abgedeckt. Daher werden viele der eigentlichen Protokollfunktionen von CoAP für die Nutzung als Transportprotokoll für SOAP nicht benötigt. Hierzu zählen Funktionen, wie beispielsweise das Suchen und Finden von Geräten bzw. asynchrone Push-Nachrichten, die bereits durch WS-Discovery und WS-Eventing ermöglicht werden. Dies verringert den notwendigen Protokollumfang und erleichtert dadurch die Implementierung merklich. Der Vorteil der Nutzung der WS-* Protokolle für die Abbildung solcher Funktionen liegt weiterhin darin, dass die Steuerinformationen in den Nachrichten durch effiziente Darstellungsformen (z.B. EXI) besonders effektiv kodiert werden können. Das Protokoll, welches für den Transport genutzt wird, ist davon in der Regel ausgenommen.

Dennoch macht das SOAP-over-CoAP Binding von CoAP nicht nur Gebrauch als Transportprotokoll, sondern nutzt es ebenfalls als Anwendungsschichtprotokoll. Beispielsweise können Erfolg und Fehler durch die zugehörigen CoAP Response Codes (vgl. HTTP Status Codes) in der Antwort angezeigt werden.

Die genannte grundlegende Nutzung von CoAP entspricht der Nutzung von HTTP im existierenden SOAP-over-HTTP Binding, welches ebenfalls nicht alle HTTP-Funktionen vollständig nutzt. Der Vorteil in diesem Ansatz liegt darin, dass die zustandslose Umsetzung von CoAP in HTTP in Proxys auch bei der Nutzung als SOAP-Transportmechanismus erhalten bleibt. Würden für das

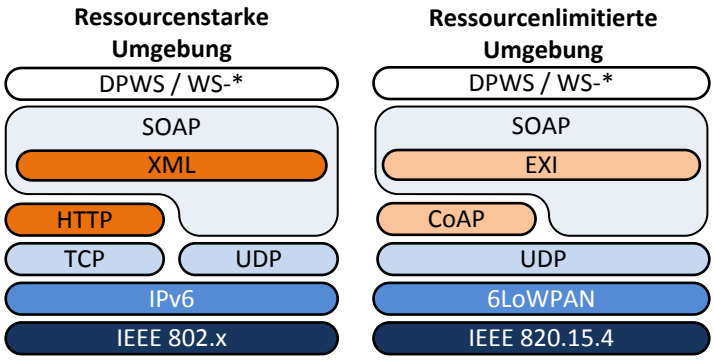

Abbildung 7.9 Ressourcenoptimierter DPWS-Protokollstapel

CoAP-Binding Funktionen und Mechanismen eingesetzt werden, die jeweils nur von einem der beiden Protokolle abgebildet werden können, wäre dieser Vorteil nicht mehr gegeben.

Dadurch fügt sich das CoAP-Binding nahtlos in die Gesamtarchitektur (siehe Kapitel 4) ein. CoAP kann entsprechend der Spezifikation zustandslos in HTTP überführt werden und somit ebenfalls das SOAP-over-CoAP Binding in das SOAP-over-HTTP Binding. Da beide Protokolle unabhängig gegenüber den Nutzdaten sind, kann SOAP in allen bereits aus vorangegangenen Kapiteln bekannten Formaten, wie beispielsweise Unicode XML, EXI und Fast Infoset, kodiert werden. Auch eine Kombination mit generischen Datenkompressoren ist möglich (z.B. gzip). Wie in Kapitel 6 dargestellt sind ebenfalls natives XML und EXI zustandslos und transparent ineinander überführbar. Für die Umsetzung der Kodierungen ineinander kann somit die Proxy-Infrastruktur von CoAP zur Umsetzung in HTTP wiederverwendet werden.

Die Anzeige der Kodierungsform erfolgt bei HTTP und bei CoAP in speziellen Header-Feldern. Seit der SOAP-Spezifikation in der Version 1.2 ist es möglich, in das gleiche Header-Feld von HTTP ebenfalls die aufzurufende Methode eines Dienstes einzubetten. Dadurch ist ein SOAP Antwort-Kommunikationsmuster möglich (siehe Kapitel 7.1.6), bei dem ein SOAP-Dokument nur in der Antwort, nicht aber in der Anfrage enthalten ist. Ein mögliches Szenario entspricht dem Abrufen einfacher Sensordaten, bei dem außer dem Aufruf der Methode des jeweiligen Dienstes keine Steuerinformationen notwendig sind. CoAP erbt von HTTP diesen Mechanismus und verfügt somit ebenfalls über das genannte SOAP-Antwort-Kommunikationsmuster.

Bei der Nutzung von HTTP kann ein anfragender Endpunkt durch ein spezielles HTTP-Headerfeld (vgl. HTTP Accept) für die Antwort eine spezifische Kodierung fordern, die von der Kodierung der Anfrage abweichen kann. Auch CoAP verfügt über diesen Mechanismus. Dies fügt sich für spezifische Anwendungen unter Umständen in die Implementierungsstrategie ein, da ausgehende Nachrichten statisch eingebunden werden können, zur Laufzeit unverändert bleiben und folglich nur ein dediziertes Darstellungsformat unterstützt wird. Somit müssen nur eingehende Nachrichten untersucht werden (siehe Implementierung von uDPWS in Kapitel 5.3). Der jeweils andere Kommunikationspartner muss mit der entstehenden Variabilität umgehen gehen.

7.3.3 Identifizierung und Zuordnung von Nachrichten und Anfragen

Da CoAP auf UDP aufsetzt, wird innerhalb von CoAP auf Anwendungsschicht zwischen Nachrichten und Anfragen bzw. Antworten unterschieden. CoAP-Nachrichten (vgl. Schicht

CoAP-Messages) transportieren CoAP-Anfragen bzw. CoAP-Antworten (vgl. Schicht CoAP-Request-Response) und stellen z.B. die Transportzuverlässigkeit her. Da diese beiden Schichten voneinander entkoppelt werden können, besitzen beide Schichten eigene Mechanismen, um zum einen die Nachrichten und zum anderen Anfragen und Antworten einander zuzuordnen.

Auf der Nachrichtenschicht werden hierfür *Message IDs* genutzt. Diese müssen in jeder Nachricht enthalten sein und dienen, neben der Transportzuverlässigkeit, der Erkennung von Duplikaten eingehender Nachrichten.

Um Anfragen und Antworten eindeutig identifizieren und einander zuordnen zu können und dennoch von der Nachrichtenschicht unabhängig zu sein, kann jede Anfrage einen *Token* enthalten. Dieser *Token* wird bei der Antwort wieder eingefügt und erlaubt so die Zuordnung zur Anfrage.

SOAP-basierte Web Services nutzen meist die WS-Addressing Spezifikation. Auch WS-Addressing beschreibt einen Mechanismus, bei dem eindeutige *Message IDs* genutzt werden, um SOAP-Anfragen und SOAP-Antworten der Dienste einander zuordnen zu können. Diese *Message IDs* werden ebenfalls vom Client generiert und in die Anfrage integriert. In der Folge integriert der Dienst die gleiche *Message ID* auch in der Antwort.

Im Unterschied zur datentypfreien CoAP *Token Option* ist das Format für die WS-Addressing *Message IDs* als XML-Schema Datentyp definiert. DPWS schreibt vor, dass die WS-Addressing *Message IDs* einem *urn:uuid IRI* Format [166] folgen müssen. Solche *urn:uuids* sind bis zu 128 Bit breite hexadezimale eindeutige Bezeichner. Diese werden normalerweise mit dem Datentyp *xs:anyURI* als Zeichenketten dargestellt, die eine typische Länge von 45 Byte haben. Diese Zeichenketten sind eindeutig und daher von automatenbasierten Datenformaten, wie beispielsweise EXI, kaum zu optimieren.

Die CoAP *Token Option* definiert somit einen zu großen Teilen identischen Mechanismus wie die WS-Addressing *Message ID*. Da für das SOAP-over-CoAP Binding SOAP-Anfragen in CoAP-Anfragen und SOAP-Antworten in CoAP-Antworten eingebettet werden können, sind die Mechanismen in vielen Szenarien redundant. Der Unterschied liegt lediglich in der optimaleren Darstellung der CoAP *Token Option*, da hierfür kein konkretes Format definiert ist. Daher kann im Kontext des SOAP-over-CoAP Bindings die Anforderung an die WS-Addressing *Message ID*

insoweit abgeändert werden, als dass diese nur dann im SOAP-Dokument enthalten sein muss, wenn keine CoAP *Token Option* vorhanden ist bzw. dies für das Szenario zwingend nötig ist.

Das HTTP-Binding verfügt nicht über einem Mechanismus, wie er mit der CoAP *Token Option* realisiert werden kann, da HTTP auf das verbindungsorientierte TCP aufsetzt. Werden die Anfragen bzw. die Antworten durch die Nutzung eines CoAP-HTTP-Proxys in das jeweils entgegengesetzte Binding umgesetzt, so muss der CoAP-HTTP-Proxy ggfs. die aufgetretene WS-Addressing *Message ID* für die Dauer der Anfrage zwischenpuffern. Intern im Proxy erfolgt in diesem Fall ebenfalls eine Zuordnung der durch ihn generierten und genutzten CoAP *Token Options* zu den WS-Addressing *Message IDs*.

7.3.4 Message Exchange Patterns

SOAP definiert verschiedene Kommunikationsmuster (engl. Message Exchange Patterns). Dies sind unter anderem das Anfrage-Antwort-Muster sowie das Anfrage-Muster. Weiterhin existiert das Antwort-Muster, bei dem lediglich eine Antwort auf SOAP-Ebene gesendet wird, die durch einen anderen Mechanismus ausgelöst wird. Das SOAP-over-CoAP Binding ist, wie auch das HTTP-Binding, in der Lage, alle diese Kommunikationsmuster zu realisieren. Die verschiedenen Kommunikationsmuster sind in Abbildung 7.10 für das jeweilige SOAP-Binding dargestellt.

Beim Anfrage-Muster enthält lediglich die CoAP-Anfrage ein SOAP-Dokument als Nutzlast. Die CoAP-Antwort enthält keine Nutzdaten und besteht auf Anwendungsschicht lediglich aus dem CoAP-Protokollkopf. Daher können beim CoAP-Binding, ähnlich wie beim HTTP-Binding und im Gegensatz zum UDP-Binding, auch beim Anfrage-Muster Rückmeldungen an den aufrufenden Endpunkt gegeben werden, soweit dies durch CoAP-Mechanismen (z.B. Response Codes) möglich

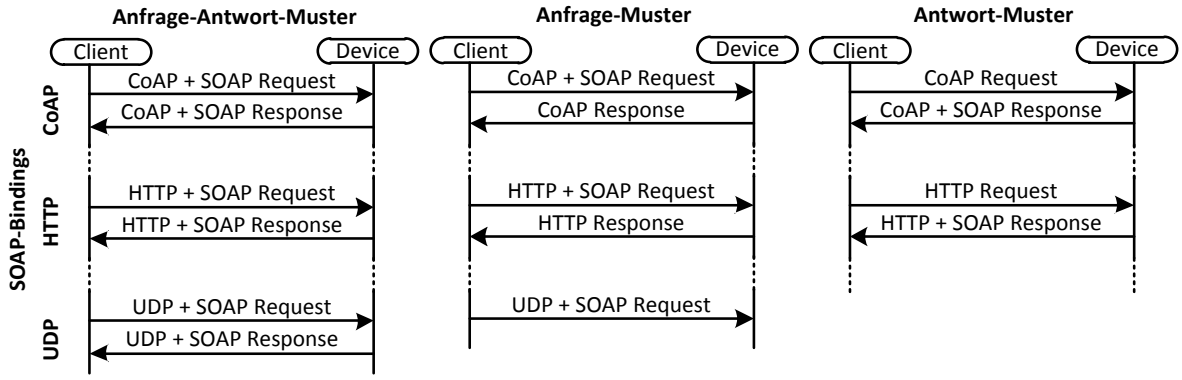

Abbildung 7.10 Message Exchange Patterns von SOAP-over-CoAP

ist. Beim Anfrage-Antwort-Muster enthält auch die CoAP-Antwort ein SOAP-Dokument als Nutzlast. Die Nutzung des mit dem HTTP-Binding umsetzbaren Antwort-Musters ist mittels CoAP ebenfalls möglich.

CoAP ermöglicht die Herstellung der Transportzuverlässigkeit mittels CoAP-Messages. Diese Schicht realisiert ein einfaches Bestätigungsverfahren über eingehende Nachrichten. Die Besonderheit besteht darin, dass das Bestätigungsverfahren von den Anfragen und Antworten von CoAP und somit auch von SOAP entkoppelt werden kann. Im einfachsten Fall wird die Empfangsbestätigung synchron gemeinsam mit der CoAP-Antwort gesendet. Benötigt das Gerät für das Erzeugen der Antwort einen vergleichsweise langen Zeitraum, so kann der Empfang der Nachricht mit der CoAP-Anfrage asynchron bereits vor dem Senden der CoAP-Antwort bestätigt werden. Die CoAP-Antwort wird in diesem Fall übermittelt, sobald diese zur Verfügung steht. In den beiden aufgeführten Fällen wird lediglich für die Anfrage ein zuverlässiger Transport erreicht. In den meisten Fällen genügt dies, da in einem solchen Fall beim Auftreten eines Fehlers bei der Übermittlung der Antwort eine vollständig neue Anfrage durch den ursprünglichen Client initiiert wird. Soll dies vermieden werden, so können im asynchronen Modus von CoAP getrennte Nachrichtenübermittlungen genutzt werden, die jede für sich gelten und jeweils nur die

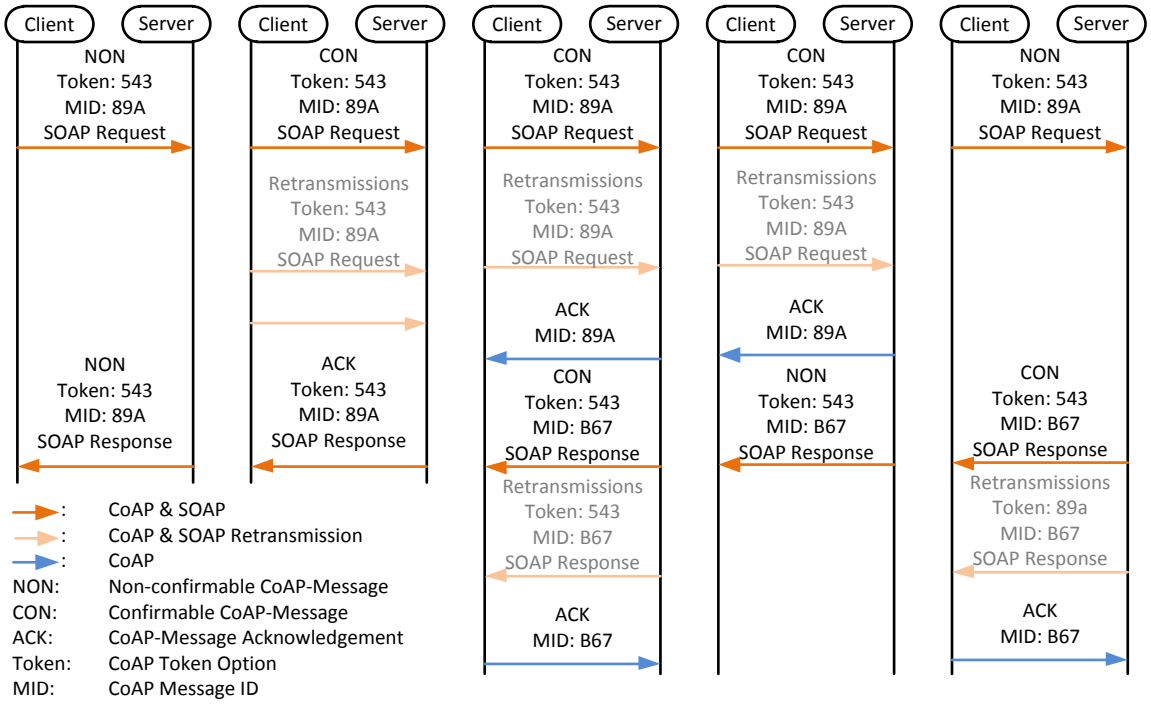

Abbildung 7.11 Transportmodi des SOAP-over-CoAP Bindings

CoAP-Anfrage bzw. die CoAP-Antwort übermitteln. Auf HTTP übertragen, würde dies bedeuten, dass zwei getrennte TCP-Verbindungen für das Senden der Anfrage und der Antwort genutzt werden.

Die aufgeführten Modi für das SOAP Anfrage-Antwort-Muster sind in Abbildung 7.11 dargestellt. Das in Abbildung 7.11 dargestellte Bestätigungsverfahren für die Nachrichtenübermittlung ist bei CoAP grundsätzlich optional. Im Kontext des SOAP-over-CoAP Bindings sollte das Bestätigungsverfahren allerdings genutzt werden, da dies einen wichtigen Unterschied zum existierenden SOAP-over-UDP Binding darstellt.

Durch die WS-Addressing Spezifikation besteht in DPWS die Möglichkeit, eine SOAP-Antwort zu einem Endpunkt zu senden, der sich von dem unterscheidet, welcher die ursprüngliche SOAP-Anfrage gesendet hat. Die entsprechende Adresse des dritten Endpunktes wird bei Bedarf im *ResponseTo*-Feld von WS-Addressing übermittelt. Eine solche Separierung wird von CoAP nativ nicht unterstützt. Daher stellt dies eine Erweiterung von möglichen CoAP-Implementierungen bei der Nutzung von CoAP als SOAP-Transportmechanismus dar. Die Implementierungen müssen ggfs. Empfangsbestätigungen über den Eingang der Nachricht mit der Anfrage an einen anderen Endpunkt senden als die dazugehörige Antwort.

7.4 Implementierung von SOAP-over-CoAP

Um Untersuchungen des Speicherbedarfes und der Leistungsfähigkeit des konzipierten CoAP-Bindings im Vergleich mit anderen existierenden SOAP-Bindings durchführen zu können, wurde das SOAP-over-CoAP Binding implementiert. Das CoAP-Binding wurde zum einen für einen DPWS-Client und zum anderen für uDPWS (siehe Kapitel 5.3) als DPWS-Gerät realisiert. Letztere Implementierung entstand in Teilen im Rahmen der Bachelorarbeit von Beichler [167].

Als DPWS-Client wurde die existierende DPWS-Implementierung WS4D-gSOAP [106] genutzt. Für die erforderlichen CoAP-Funktionen kam die existierende C-Bibliothek libcoap [168] zum Einsatz. Das CoAP-Binding wurde dazu als gSOAP-Plugin umgesetzt, wodurch es von DPWS unabhängig ist und auch für andere gSOAP-basierte Implementierungen eingesetzt werden kann.

Für die Implementierung des CoAP-Bindings für Zielplattformen von 6LoWPANs wurde uDPWS um den neuen Transportmechanismus erweitert. Als Hardwareplattform werden in den

nachfolgenden Unterabschnitten, wie bereits in Kapitel 5.3 beschrieben, ausschließlich Ergebnisse bezüglich der TelosB-Plattform dargestellt.

7.4.1 Implementierung

Der modulare Aufbau von uDPWS wurde für die Erweiterung um das CoAP-Binding beibehalten, um die drei Bindings SOAP-over-UDP, SOAP-over-HTTP und SOAP-over-CoAP unabhängig voneinander untersuchen zu können und nicht benötigte Module zur Entwicklungszeit auszuschließen. Bei der Erweiterung von uDPWS um das neue Binding war es unter anderem notwendig, dass das zusätzlich unterstützte SOAP-Binding bei der Geräte- bzw. Dienstbeschreibung im Rahmen des Discovery-Vorganges mit angezeigt werden kann, um zur Laufzeit die dynamische Aushandlung des optimalen Verfahrens zur Kommunikation zwischen zwei Endpunkten zu ermöglichen.

Für das SOAP-over-CoAP Transportmodul wurde die CoAP-Implementierung für Contiki mit der Bezeichnung Erbium von Kovatsch genutzt, die in [115] beschrieben ist. Erbium wurde nicht weiter optimiert, sondern in der existierenden Form verwendet.

7.4.2 Speicherbedarf

Tabelle 7.2 stellt den statischen Speicherbedarf des ROMs des Systems für drei verschiedene Konfigurationen dar. Das UDP-Binding für SOAP wird in allen Konfigurationen unterstützt, da es bei der Kommunikation mittels IP-Multicast für das Discovery obligatorisch ist. Die zweite Spalte der Tabelle zeigt den Speicherbedarf von uDPWS, welches zusätzlich zu UDP das SOAP-over-HTTP Binding unterstützt. Die dritte Spalte zeigt den Speicherbedarf von uDPWS, wenn zusätzlich zum UDP-Binding SOAP-over-CoAP unterstützt wird. Die letzte Spalte stellt den

Bindings	**HTTP [Byte]**	**CoAP [Byte]**	**CoAP + HTTP [Byte]**
System	1602	1302	1572
Contiki	26394	23254	26394
uDPWS	10206	12458	14216
Hosting Service	686	686	686
Air Conditioner	1978	2002	2467
Gesamt	**40866**	**39702**	**45335**

Tabelle 7.2 Speicherbedarf ROM von uDPWS mit SOAP-over-CoAP

Speicherbedarf einer Konfiguration dar, in der alle drei Transportmechanismen unterstützt werden. Bei den aufgezeigten Werten sind vorrangig die Komponenten Contiki, uDPWS und Air Conditioner sowie der Speicherbedarfs des Gesamtsystems getrennt zu bewerten.

Bezüglich der Komponente uDPWS ist die Konfiguration, in der nur das CoAP-Binding unterstützt wird, durch einen höheren Speicherbedarf als die Konfiguration gekennzeichnet, die nur das HTTP-Binding unterstützt. Dies liegt darin begründet, dass die verwendete CoAP-Bibliothek Erbium ohne weitere Anpassungen in uDPWS eingebunden wurde. Erbium selbst hat einen ROM-Bedarf von 3.036 Byte. Für das CoAP-Binding wird aber lediglich eine sehr geringe Untermenge der Funktionen von CoAP benötigt. Wenn der Standardisierungsprozess von CoAP die Entwurfsphase überschritten hat und somit die Implementierung von CoAP stabil bleibt, sind Optimierungen hinsichtlich des Speicherbedarfes möglich.

Bezüglich der Komponente Contiki ist die Konfiguration auffällig, in der nur das CoAP-Binding unterstützt wird. In dieser Konfiguration kann auf das Transportprotokoll TCP vollständig verzichtet werden. Die Implementierung von TCP ist Bestandteil des Betriebssystems Contiki.

Der Speicherbedarf für die Komponente des Referenzszenarios (Air Conditioner) ist in der Konfiguration die alle Bindings unterstützt am größten, da in dieser Konfiguration die meisten unterschiedlichen Nachrichtenfragmente für ausgehende SOAP-Nachrichten vorgehalten werden müssen.

Tabelle 7.2 stellt ebenfalls den Speicherbedarf für das gesamte System dar, welches aus uDPWS, den Diensten, Contiki und grundlegenden Systemfunktionen (z.B. Treibern) besteht. Die CoAP-Binding Konfiguration, die kein TCP benötigt, weist in der Summe den geringsten Gesamtspeicherbedarf auf. Das lässt darauf schließen, dass der TCP-Stack in Contiki einen höheren ROM-Bedarf hat als der CoAP-Stack Erbium. Da der TCP-Stack von Contiki bereits als eine der effizientesten und dennoch standardkonformen Implementierungen von TCP gilt kann geschlussfolgert werden, dass CoAP mit seinem größeren Funktionsumfang dennoch ressourcenschonender implementiert werden kann. Durch die bereits genannten Optimierungen der verwendeten CoAP-Bibliothek könnte eine noch größere Differenz erzielt werden. TCP stellt sich somit nicht nur in Bezug auf die benötigte Bandbreite, sondern darüber hinaus ebenfalls bezüglich des Speicherbedarfes als nachteilig heraus.

	HTTP [Byte]	CoAP [Byte]	CoAP + HTTP [Byte]
System	4	4	4
Contiki	7114	6946	7114
uDPWS	2859	2920	2957
Hosting Service	0	0	0
Air Conditioner	6	6	6
Gesamt	**9983**	**9876**	**10081**

Tabelle 7.3 Speicherbedarf RAM von uDPWS mit SOAP-over-CoAP

In der aktuellen Version benötigt das CoAP-Binding für uDPWS, inklusive der Anpassungen des Hosting Services und der einzelnen Module von uDPWS, somit etwa 4 kB ROM. Auch das von Glombitza et al. in [85] vorgestellte UDP-basierte Binding LTP, das bezüglich der Effizienz beim Transport ebenfalls geeignet wäre, benötigt etwa 4 kB ROM. Allerdings fügt sich LTP nicht so nahtlos wie das CoAP-Binding in die in Kapitel 4 beschriebene Gesamtarchitektur ein und benötigt konzeptionelle Erweiterungen, um zustandslos in natives HTTP umgesetzt werden zu können.

Tabelle 7.3 zeigt den dynamischen Speicherbedarf des gesamten Systems ebenfalls für die drei Konfigurationen. Durch die Nutzung von IPv6 und der damit einhergehenden Anforderung der Unterstützung einer MTU von mindesten 1280 Byte wird der dynamische Speicherbedarf des gesamten Systems zu großen Teilen durch die Nachrichtenpuffer verursacht. In der Summe sind keine nennenswerten Unterschiede zwischen den Konfigurationen festzustellen, da in allen Konfigurationen die gleichen Nachrichtenpuffer genutzt werden, die jeweils identische Größen aufweisen.

7.4.3 Laufzeitmessungen

Um den Gewinn bezüglich der Leistungsfähigkeit des CoAP-Bindings zu untersuchen, wurden Laufzeitmessungen mit den entstandenen Implementierungen durchgeführt. Die verwendete Netzwerkinfrastruktur entspricht einem Single-Hop Szenario, welches Identität zu den in Kapitel 5.3 und Kapitel 6.3 dargestellten Szenarien aufweist (siehe Abbildung 7.12). Als Client wurde WS4D-gSOAP genutzt, das ebenfalls die drei möglichen SOAP-Bindings unterstützt. Alle Nachrichten sind in XML mit Unicode kodiert, da WS4D-gSOAP das EXI-Format nicht unterstützt.

Nachricht	**Probe + Probe Match**			**Resolve + Resolve Match**			**Dienst**	
Binding	**HTTP**	**CoAP**	**UDP**	**HTTP**	**CoAP**	**UDP**	**HTTP**	**CoAP**
SOAP-Anfrage	645	645	645	774	774	774	547	547
SOAP-Antwort	1125	1125	1101	1015	1015	1015	638	638
Binding-Anfrage	76	8	0	76	8	0	76	8
Binding-Antwort	92	8	0	92	8	0	91	8
Summe [Byte]	**1938**	**1786**	**1746**	**1957**	**1805**	**1789**	**1352**	**1201**
Umlaufzeit [ms]	**582,19**	**409,21**	**399,2**	**579,78**	**419,28**	**409,67**	**450,24**	**289,71**
Verarbeitungs-zeit [ms]	**12,82**	**12,03**	**11,82**	**13,69**	**12,02**	**12,05**	**8,94**	**8,13**

Tabelle 7.4 Laufzeitmessungen uDPWS mit SOAP-over-CoAP

Die Ergebnisse der Laufzeitmessungen sind in Tabelle 7.4 für drei Aufrufe exemplarisch dargestellt. Für die Nachrichten Probe/Probe Match, Resolve/Resolve Match sowie einen Dienstaufruf sind in der Tabelle jeweils die versendeten Bytes für die SOAP-Anfrage und die SOAP-Antwort angegeben. Weiterhin beinhaltet die Tabelle die benötigten Größen der Protokollköpfe der Bindings. Für alle Nachrichten wurden die insgesamt versendeten Bytes sowie die Umlaufzeiten (engl. Round Trip Time) und die Verarbeitungszeiten auf dem Sensorknoten ermittelt, die in den unteren drei Zeilen der Tabelle dargestellt sind.

Die Laufzeitmessungen zeigen, dass die SOAP-Transportmechanismen CoAP und UDP ähnlich effizient sind. Das HTTP-Binding ist etwa 30-55 % ineffizienter als die anderen beiden Bindings. Der Unterschied zwischen den beiden UDP-basierten Bindings und SOAP-over-HTTP begründet sich zum einen in der Nutzung von TCP auf Transportschicht. Zum anderen nutzt das HTTP-Protokoll ASCII-kodierte Protokollköpfe, die eine typische Größe von mindestens 100-200 Byte besitzen und zusätzlich zu den SOAP-Dokumenten übertragen werden müssen. Weiterhin kann der Trend aus der allgemeinen Gegenüberstellung von UDP und TCP aus

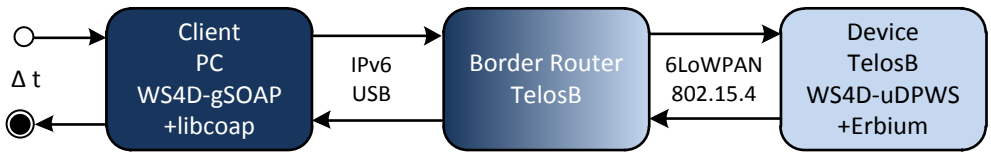

Abbildung 7.12 Schematischer Messaufbau Laufzeitmessungen uDPWS mit CoAP-Binding

Kapitel 7.2.3 bestätigt werden. Ein Grund hierfür ist unter anderem die geringe Verarbeitungszeit der Nachrichten. In Kombination mit den größenoptimierten Nachrichten und somit einer zu versendenden Nutzlast von weit unter 100 Byte sind noch deutlich größere Unterschiede zu erwarten.

Bei allen drei Bindings und bei allen drei exemplarischen Nachrichten ist zu beobachten, dass die Verarbeitungszeit mit anteilig unter 3 % der gesamten Umlaufzeit trotz der stark limitierten Rechenleistung der Zielplattform vernachlässigbar klein bleibt. Dies bestätigt die Feststellungen aus Kapitel 5.3.3, dass das Paradigma von drahtlosen Sensornetzwerken bezüglich der Minimierung der Sende- und Empfangszeit erhalten bleibt und die Verarbeitungszeit keinen primären Einfluss auf den Energiehaushalt hat.

7.5 Zwischenfazit

Das neu konzipierte CoAP-Binding für SOAP ermöglicht die Ausnutzung des verbindungslosen UDP-Protokolls hinsichtlich der hohen Effizienz, der geringen Laufzeiten/Latenz und der ressourcenschonenden Implementierung. Die Funktionen von reinem UDP werden aber zugleich um zusätzliche Mechanismen zur Herstellung der Transportsicherheit, Duplikatserkennung und Fragmentierung auf Anwendungsschicht erweitert. Für die Nutzung von CoAP als SOAP-Transportbinding waren konzeptionelle Erweiterungen und Änderungen am ursprünglichen Design von CoAP notwendig, die durch den Autor dieser Arbeit direkt in der entsprechenden IETF Arbeitsgruppe eingebracht wurden. Somit wurde gezielt Einfluss auf den Standardisierungsprozess von CoAP genommen.

Da für CoAP dedizierte Proxys zur zustandslosen und transparenten Umsetzung von CoAP in HTTP und umgekehrt definiert wurden, fügt sich das SOAP-over-CoAP Binding nahtlos in eine skalierbare IP-basierte Netzwerkinfrastruktur ein. Auf schwergewichtige Anwendungsschichtgateways muss nicht zurückgegriffen werden. Weiterhin kann CoAP mit verschiedenen Darstellungsformen der zu transportierenden Daten kombiniert werden. Hierzu zählen natives Unicode XML sowie effizientere Formate wie EXI. Somit konnte ein Binding konzipiert werden, dass sowohl bandbreiten- als auch ressourceneffizient ist und sich zugleich in einer WSA-basierten Dienstumgebung nutzen lässt.

Mit der Implementierung von uDPWS, welches durch das neuartige Binding bis zu drei verschiedene Transportmechanismen unterstützt, wurde bereits ohne weitere Optimierungen des Betriebssystems Contiki und der genutzten CoAP-Bibliothek der Nachweis erbracht, dass DPWS auch auf stark ressourcenlimitierten Systemen realisierbar ist. Der mittels der Implementierung nachgewiesene Vorteil der UDP-basierten Bindings SOAP-over-UDP und SOAP-over-CoAP gegenüber dem TCP-basierten SOAP-over-HTTP hinsichtlich der Gesamtlaufzeit liegt in der Verbindungslosigkeit von UDP und der effizienteren Darstellungsform der Protokollköpfe begründet.

Die Mechanismen von TCP zur Flusskontrolle und zur Transportsicherung sind in stark limitierten Umgebungen nicht effizient anwendbar. Das SOAP-over-HTTP Binding sollte folglich nur in ressourcenstarken Netzwerken genutzt werden. In 6LoWPAN Netzwerken sind für den Transport von SOAP-Dokumenten die Bindings UDP und CoAP zu bevorzugen. Das UDP-Binding ist bei der Kommunikation mittels UDP-Multicast zu bevorzugen, da in der Regel keine Empfangsbestätigungen gesendet werden sollten (vgl. Response Strom), sondern die Anwendung direkt auf die Anfragen reagiert und wenn nötig eine Antwort sendet. Diese Antwort wird vom ursprünglichen Sender als Empfangsbestätigung interpretiert. Für die Kommunikation mittels Unicast sollten die Vorteile von CoAP bezüglich der Transportsicherheit, Duplikatserkennung und Fragmentierung ausgenutzt werden.

Das konzeptionell erweiterte DPWS in Kombination mit dem in diesem Kapitel spezifizierten Binding SOAP-over-CoAP sowie dem im vorrangegangen Kapitel vorgestellten erweiterten EXI-Format stellt eine effiziente Lösung bezüglich eines Anwendungsschichtprotokolls für 6LoWPANs dar. Dabei genügt die benötigte Bandbreite den Anforderungen der stark limitierten, drahtlosen Umgebungen. Die Implementierungen der Protokolle sind durch einen ähnlichen Speicherverbrauch gekennzeichnet wie Lösungen, die speziell für diese Umgebungen entwickelt wurden. Durch die höhere Komplexität, Modularität und die starke Erweiterbarkeit ist diese WSA-konforme Lösung aber als deutlich leistungsstärker zu bewerten.

Nicht betrachtet wurden Multi-Hop Netzwerke sowie das Verhalten in Abhängigkeit von Paketverlusten entlang des Kommunikationskanals. Für die beiden SOAP-Transportmechanismen CoAP und HTTP sind jeweils Mechanismen und Algorithmen definiert, wie Paketverluste und Verbindungsabbrüche kompensiert werden und darauf reagiert werden kann. Wie effizient diese Mechanismen und Algorithmen im Kontext des Transports von komprimierten und auch nicht

komprimierten SOAP-Dokumenten sind, konnte aufgrund fehlender effizienter Routingverfahren für Multi-Hop 6LoWPANs im Rahmen dieser Arbeit nicht weiter untersucht werden.

8 Zusammenfassung, Fazit und Ausblick

Während in vielen existierenden Anwendungen und ganzen Domänen entweder Geräte oder höherwertige Dienste im Internet jeweils exklusiv miteinander verknüpft werden, sind zukünftige Lösungen durch ein hohes Maß an Verschränkung der bislang getrennten Bereiche geprägt. Bei der gerätenahen Kommunikation werden batteriebetriebene Kleinstgeräte als Sensoren oder einfache Aktoren einen essenziellen Anteil an der Datenerfassung und Interaktion mit der physikalischen Umgebung haben. Ein enorm hohes Maß an Heterogenität bezüglich der Dienste, Daten, Plattformen und Ressourcen ist somit die Folge.

Die batteriebetriebenen Kleinstgeräte sind bezüglich den zur Verfügung stehenden Hardwareressourcen und der Bandbreite drahtlosen Sensornetzwerken sehr ähnlich. Existierende Protokolle und Technologien für diese vernetzten Umgebungen erfüllen die Ressourcenanforderungen nur durch die Verschiebung sämtlicher Komplexität der Anwendung und der Protokolle auf ressourcenstärkere Stellvertreter. Dies resultiert meist in unflexiblen und schwergewichtigen Gateway-Infrastrukturen. Um dies zu vermeiden, ist auch in diesen ressourcenarmen Systemen ein starker Trend zu IP-basierten Lösungen zu beobachten. Mit den 6LoWPAN-Protokollen (IPv6 over Low power Wireless Personal Area Networks) der IETF wurden die Grundlagen für IPv6-basierte Kommunikation geschaffen. Bei der Konzeption einer Vernetzungslösung auf Anwendungsschicht müssen die durch IP implizierte Infrastruktur und die Architekturprinzipien berücksichtigt werden, um nicht auf höheren Schichten wieder eine Kapselung der Netzwerke und von den offenen Technologien zu bewirken.

In der vorliegenden Arbeit wurde für die Anwendungsschicht von 6LoWPANs zunächst eine skalierbare Architektur sowie Infrastruktur konzipiert, durch die diese Netzwerke in höherwertige Diensteumgebungen integriert werden können. Hierbei ist die Ende-zu-Ende Konnektivität eine der elementarsten Prinzipien. Um existierende Implementierungen nicht ändern müssen, können transparente Vermittler zur zustandslosen Umsetzung der existierenden und der neu konzipierten Protokolle eingesetzt werden.

Anschließend wurde eine Lösung spezifiziert, die sowohl den Prinzipien der konzipierten Architektur und Infrastruktur, als auch den Ressourcenanforderungen von 6LoWPANs genügt. Hierfür waren einerseits Neuentwicklungen, konzeptionelle Anpassungen und

Weiterentwicklungen von Ansätzen sowie der umsetzenden Protokolle und andererseits dedizierte Implementierungen nötig, um allen Aspekten gerecht zu werden. Die Spezifizierung der im Rahmen dieser Arbeit entstandenen Protokolle ging durch den Autor weiterhin unter anderem in die Standardisierung des Constrained Application Protocols (CoAP) bei der IETF ein. Die durch den Autor erwirkten Änderungen umfassen unter anderem, welche Methoden und Statuscodes von CoAP Nutzdaten enthalten dürfen. Diese Änderung zog weitere Anpassung der in CoAP integrierten Nachrichtenfragmentierung auf Anwendungsschicht nach sich.

Die vorgestellte Lösung basiert auf SOAP und dem W3C Web Services Framework und ist durch hohe Flexibilität und Modularität gekennzeichnet. Für gerätenahe Umgebungen existiert mit dem Devices Profile for Web Services (DPWS) bereits ein dediziertes Profil. In bisherigen Arbeiten bleibt aber bislang ungeklärt, ob allgemein SOAP-basierte Web Services und konkreter DPWS auch weit genug nach unten skalieren, um in drahtlosen, vermaschten Netzwerken aus batteriebetriebenen Kleinstgeräten eingesetzt werden zu können. Der Erhalt der Konformität zur WS-Architecture und somit zum WS-* Framework ist dabei eine nicht zu vernachlässigende Kernanforderung. Zum einen wird dadurch die existente Ergänzung um vorhandene und eigene Erweiterungen nicht eingeschränkt. Zum anderen wird trotz der Erweiterbarkeit die Interoperabilität gewährleistet, da diese durch die in der WS-Architecture spezifizierten Architekturprinzipien sichergestellt ist. Die Spezifizierung einer solchen dedizierten Architektur ist ein immenser Vorteil von SOAP Web Services gegenüber anderen Protokollen.

Die in dieser Arbeit erstmals konzipierte und auf Stärken sowie Schwächen untersuchte Lösung ist mit der bestehenden WS-Architecture des W3C und somit mit SOAP und DPWS kompatibel. Dies wird möglich, da bei SOAP-basierten Web Services die drei Bereiche der (1) Schnittstellenmechanismen, (2) der Datenrepräsentation und (3) des Transportes der Informationsträger nahezu vollständig entkoppelt voneinander betrachtet und für jeden einzelnen Bereich eine möglichst effiziente Lösung hervorgebracht werden kann. In existierenden Ansätzen verwandter Arbeiten führte eine zu starke Verschmelzung der drei Bereiche stets zu Inkompatibilität mit der WS-Architecture und somit dem Verlust des immensen Vorteils SOAP-basierter Web Services.

Bezüglich der konzeptionierten, allgemeingültigen Schnittstellenmechanismen für DPWS konnte mit der Neudefinition eines Template-Konzeptes sowie einer Umgestaltung der Discovery-Phase das Suchen und Finden von Geräten in einem Netzwerk merklich effizienter gestaltet werden. Die

Erweiterung des Eventing-Mechanismus um deutlich feingranulare Filter ermöglicht weiterhin eine ressourcenschonendere Form der asynchronen Push-Kommunikation. Das weiterentwickelte Eventing kommt vor allen in Szenarien zum Tragen, in denen Sensorwerte überwacht werden müssen (vgl. Schwellwertüberwachung). Das verbesserte Discovery trägt vorrangig in dynamischen Szenarien mit vielen unbekannten Geräten und Clients zur Erhöhung der Effizienz bei.

Bei der Konzeption einer geeigneten Form der Datenrepräsentation wurden in dieser Arbeit, nach detaillierter Analyse der Anforderungen und Voraussetzungen, feingranulare Untersuchungen existierender Lösungen vorgenommen. Unter den existierenden Kodierungen sind automatenbasierte Strategien mit bekanntem Vokabular und bekannter Grammatik deutlich zu bevorzugen. Alle anderen Lösungen erreichen bei der Anwendung auf die vergleichsweisen einfachen Dienste und Datenstrukturen lediglich Kompressionsraten von etwa 50-60 %. Durch dedizierte Weiterentwicklung des ausgewählten automatenbasierten EXI-Formates entstand eine effiziente Lösung, die SOAP-basierte Kommunikation in 6LoWPANs bezüglich der Bandbreite ermöglicht. Unter den diversen Ausprägungen des EXI-Formates konnte der Modus *schema-informed bit-aligned non-strict* als einzige geeignete Variante identifiziert werden. Nur in diesem Modus sind Nachrichtengrößen von unter 100 Byte bei akzeptabler Komplexität des Formates zu erreichen. Mit der Implementierung uEXI entstand die erste frei zugängliche Implementierung von EXI für Zielplattformen von 6LoWPANs, wodurch Untersuchungen bezüglich des Speicherbedarfs möglich wurden. Trotz der immensen Komplexität des EXI-Formates und des zur effizienten Kodierung von DPWS benötigten Automatenstruktur (vgl. EXI-Grammar) benötigt uEXI weniger als 20 kB ROM. Die Untersuchungen haben ebenfalls gezeigt, dass anwendungsspezifische Erweiterungen und Anpassungen der Grammatiken und des Vokabulars keinen nennenswerten Einfluss auf den Speicherbedarf haben, die Effizienz der Kodierung aber deutlich gesteigert wird.

Anders als zur Konzeption einer geeigneten Form der Datenrepräsentation wurde für den effizienten Transport von SOAP-Dokumenten mit dem SOAP-over-CoAP Binding ein vollständig neues Protokoll spezifiziert, das sich in der vorliegenden Form sowohl bezüglich der Ressourcen als auch zur Einbettung in die WS-Architecture optimal darstellt. Mit dem neuartigen Binding werden die Vorteile des zustandslosen UDP und die zusätzlichen Funktion von CoAP (Zuverlässigkeit, Duplikatserkennung, Fragmentierung, zustandslose Umsetzung in HTTP)

überzeugend in einem Binding kombiniert. Ein solches Transportbinding für SOAP war bislang nicht vorzufinden. Gegenüber dem existierenden SOAP-over-HTTP wurde eine Verringerung der Latenz um 30-55 % nachgewiesen. Dies entspricht, trotz der zusätzlichen Protokollmechanismen, der Effizienz des nativen UDP-Bindings. In der Kombination mit EXI entsteht eine noch weitaus größere Differenz zwischen den Bindings HTTP und CoAP.

Alle definierten Protokolle zielen jeweils auf die Erhöhung der Effizienz, sind aber dennoch unabhängig voneinander und somit in diversen Modi miteinander kombinierbar und austauschbar. Bei einer konkreten Umsetzung kann dadurch jeweils das beste Verhältnis aus den hier vorgestellten Erweiterungen und weiteren technischen oder nicht-technischen Anforderungen gewählt werden.

Die entstandenen Implementierungen uDPWS und uEXI sind beide als Open Source durch die Web Services for Devices (WS4D) Initiative für zukünftige Forschungs- und Entwicklungsarbeiten zugänglich. Die Implementierungen stellen die weltweit ersten frei zugänglichen Umsetzungen der jeweiligen Protokolle für konkrete Zielplattformen von 6LoWPANs dar und zeigen die grundsätzliche Machbarkeit auf. Die Implementierungen können für den produktiven Einsatz bezüglich des Speicherbedarfes und der benötigten Rechenleistung weiter optimiert werden.

Die Kernfragestellung dieser Arbeit ob SOAP bzw. DPWS skalierbar genug sind, um allgemein in stark ressourcenlimitierten Umgebungen bzw. zur Vernetzung dieser mit höherwertigen Umgebungen eingesetzt werden zu können, kann abschließend größtenteils positiv beantwortet werden. Die entstandene Architektur ist flexibel und erweiterbar genug, um auch auf konkrete Domänen angewendet werden zu können und ggfs. um domänenspezifische Erweiterungen ergänzt zu werden. Ein noch fehlender Mechanismus der genutzten Protokolle stellt das Caching dar, welches die Skalierbarkeit eines Systems in Abhängigkeit der Anzahl der Netzwerkteilnehmer erhöhen kann. Die Entwicklung einer allgemeingültigen und anwendungs- wie auch domänenübergreifenden Lösung zur Realisierung von Caching-Funktionen für SOAP-basierte Web Services bzw. für DPWS sollte Bestandteil zukünftiger Betrachtungen sein. Weiterhin ist bislang ungeklärt, wie eine Ende-zu-Ende Verschlüsslung trotz der Ressourcenbeschränkungen realisiert werden kann. Existierende zertifikatbasierte Verschlüsselungsverfahren können dies bislang nicht ermöglichen. Diese Fragestellung ist allerdings unabhängig von SOAP bzw. DPWS und wird derzeit unter anderem in der IETF bearbeitet, um eine Grundlage für eine Vielzahl von weiterführenden Protokollen bereitzustellen.

Die Frage ob SOAP bzw. DPWS ausreichend skaliert, um ganz konkret in 6LoWPAN-basierten Netzwerken eingesetzt werden können, kann ebenfalls positiv beantwortet werden. Bislang nicht geklärt ist lediglich die effiziente Umsetzung einer Gruppenkommunikation in 6LoWPANs. Für die Gruppenkommunikation in 6LoWPANs steht derzeit IP-Multicast zur Verfügung. Allerdings wird Multicast weder von der Zugriffschicht noch von den 6LoWPAN-Protokollen selbst noch von den existierenden Routingverfahren für 6LoWPANs optimal unterstützt. Wie auch die Problemstellung einer effizienten Ende-zu-Ende Verschlüsslung, bleibt die Fragestellung bezüglich optimaler Multicast-Kommunikation ein protokollübergreifendes Problem und bedarf ganzheitlicher Betrachtungen.

Weiterführende Untersuchungen, die auf den Grundlagen dieser Arbeit aufbauen, können somit sowohl anwendungsspezifisch als auch anwendungsunabhängig gestaltet werden. In die anwendungsunabhängigen Untersuchungen fließen konkrete Betrachtungen ein, zu denen die bereits genannten Sicherheitsaspekte und effiziente Gruppenkommunikation zählen. Aber auch allgemeinere Untersuchungen im Hinblick auf konkrete Netzwerkgrößen und –topologien können wichtige Erkenntnisse erbringen. Im Rahmen von anwendungsspezifischen Untersuchungen müssen die in dieser Arbeit vorgestellten Grundlagen im Kontext von domänenspezifischen Erweiterungen erneut evaluiert werden. Zu solchen domänenspezifischen Erweiterungen zählt beispielsweise Health Level Seven (HL7), welches SOAP-basierte Vernetzung im klinischen Umfeld ermöglich.

Erst wenn die oben genannten Aspekte hinreichend untersucht wurden, kann der gesamte Ansatz mit anderen Technologien und Vernetzungsstrategien, wie beispielsweise Bluetooth und ZigBee, umfassend und objektiv verglichen werden.

Referenzen

[1] L. Kleinrock, “Information Flow in Large Communication Nets,” Massachusetts Institute of Technology, 1961.

[2] S. J. Lukasik, “Why the Arpanet Was Built,” *IEEE Annals Of The History Of Computing*, vol. 33, no. 3, pp. 4-21, 2010.

[3] V. G. Cerf, Y. K. Dalal, and C. A. Sunshine, “Specification of Internet Transmission Control Program,” *RFC 675*. 1974.

[4] V. Cerf and R. Kahn, “A Protocol for Packet Network Intercommunication,” *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 637-648, 1974.

[5] M. Lawson, “Berners-Lee on the read / write web,” *BBC News*. 2005.

[6] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities,” *Scientific American*, vol. 284, no. 5, pp. 34-43, 2001.

[7] J. Markoff, “Entrepreneurs See a Web Guided by Common Sense,” *The New York Times*, pp. 11-14, 2006.

[8] Z. Shelby and C. Bormann, *6LoWPAN*. Chichester, UK: John Wiley & Sons, Ltd, 2009.

[9] M. Weiser, “The world is not a desktop,” *Interactions Magazine*, vol. 1, no. 1, pp. 7-8, Jan. 1994.

[10] M. Weiser, “Hot topics-ubiquitous computing,” *IEEE Computer*, vol. 26, no. 10, pp. 71-72, 1993.

[11] F. Jammes and H. Smit, “Service-oriented architectures for devices - the SIRENA view,” in *3rd IEEE International Conference on Industrial Informatics*, 2005, pp. 140-147.

[12] I. Akyildiz, “Wireless sensor networks: a survey,” *Computer Networks*, vol. 38, no. 4, pp. 393-422, 2002.

[13] W. Dargie and C. Poellabauer, *Fundamentals of Wireless Sensor Networks*. John Wiley and Sons, Ltd., 2010, pp. 168–183.

[14] B. Raman and K. Chebrolu, “Censor networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 75-78, Jul. 2008.

[15] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787-2805, 2010.

[16] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 33-35, 1965.

[17] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82-85, Jan. 1998.

[18] K. Roy, B. Jung, and A. R. Than, "Integrated Systems in the More-than-Moore Era: Designing Low-Cost Energy-Efficient Systems Using Heterogeneous Components," in *23rd International Conference on VLSI Design*, 2010, pp. 464-469.

[19] G. Bell, "Bell's law for the birth and death of computer classes," *Communications of the ACM*, vol. 51, no. 1, pp. 86-94, 2008.

[20] C. Shapiro and H. R. Varian, *Information Rules*, vol. 32, no. 2. Harvard Business School Press, 1999.

[21] B. Metcalfe, "Metcalfe's law: A network becomes more valuable as it reaches more users," *InfoWorld*, vol. 17, no. 40, pp. 53-54, 1995.

[22] T. Nixon, A. Regnier, D. Driscoll, and A. Mensch, "Devices Profile for Web Services Version 1.1," *OASIS Standard*. OASIS, pp. 1-43, 2009.

[23] Oasis, "OASIS: Advancing open standards for the information society," 2012. [Online]. Available: http://www.oasis-open.org/.

[24] Microsoft, "Windows Rally," 2012. [Online]. Available: http://www.microsoft.com/whdc/connect/rally/default.mspx.

[25] D. Booth et al., "Web Services Architecture," *W3C Working Group Note*. World Wide Web Consortium, 2004.

[26] P. Spiess et al., "SOA-Based Integration of the Internet of Things in Enterprise Services," in *IEEE International Conference on Web Services*, 2009, pp. 968-975.

[27] D. Savio and S. Karnouskos, "Web-service enabled wireless sensors in SOA environments," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 952-958.

[28] H. Bohn, F. Golatowski, and D. Timmermann, "Dynamic device and service discovery extensions for WS-BPEL," in *International Conference on Service Systems and Service Management*, 2008, pp. 1-6.

[29] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," *W3C Recommendation*. World Wide Web Consortium, 2007.

[30] J. D. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334-1340, 1983.

[31] J. Postel, "Internet Protocol," *RFC 791*. Internet Engineering Task Force, 1981.

[32] S. Frankel and D. Green, "Internet Protocol Version 6," *RFC 2460*. Internet Engineering Task Force, 1998.

[33] G. Mulligan, "The 6LoWPAN architecture," in *4th Workshop on Embedded Networked Sensors*, 2007, p. 78.

[34] G. Tolle, "A 6LoWPAN application environment," in *5th International ACM Conference on Embedded Networked Sensor Systems*, 2007, pp. 375-376.

[35] T. Winter et al., "RPL: IPv6 Routing Protocol for Low power and Lossy Networks," *RFC6550*. Internet Engineering Task Force, 2012.

[36] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis, "Evaluating the Performance of RPL and 6LoWPAN in TinyOS," in *Workshop on Extending the Internet to Low Power and Lossy Networks*, 2011.

[37] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP)." Internet Engineering Task Force, 2011.

[38] L. Richardson and S. Ruby, *RESTful Web Services*, vol. 12, no. 6. O'Reilly, 2007, pp. 1-448.

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

[40] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall, 2005.

[41] I. Melzer, *Service-orientierte Architekturen mit Web Services*, vol. 31, no. 9. Spektrum Akademischer Verlag, 2010, pp. 666-670.

[42] "Web Services Resource Access (WSRA)," 2008. [Online]. Available: http://www.w3.org/2008/08/ws/charter.html.

[43] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. 'big' web services: making the right architectural decision," in *17th International Conference on World Wide Web*, 2008.

[44] W3C, “Efficient XML Interchange Working Group.” [Online]. Available: http://www.w3.org/XML/EXI/.

[45] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*, 1st ed. Morgan Kaufmann, 2009, p. 432.

[46] Statistische Ämter Des Bundes Und Der Länder, Ed., “Demografischer Wandel in Deutschland,” Statistisches Bundesamt, 2011.

[47] S. Hanke et al., “universAAL - eine offene und konsolidierte AAL-Plattform,” in *Ambient Assited Living - 4. Deutscher AALKongress*, 2011, pp. 127-140.

[48] P. Wolf et al., “openAAL - the open source middleware for ambient-assisted living AAL,” in *AALIANCE conference*, 2010, pp. 1-5.

[49] G. Fagerberg et al., “Platforms for AAL Applications,” in *5th European Conference on Smart Sensing and Context*, 2010, pp. 177-201.

[50] H. Nakashima et al., *Handbook of Ambient Intelligence and Smart Environments*. Boston, MA: Springer US, 2010, pp. 1171-1199.

[51] M. Lipprandt et al., “OSAMI-D: An open service platform for healthcare monitoring applications,” in *2nd Conference on Human System Interactions*, 2009, pp. 139-145.

[52] K. Kok et al., “Smart houses for a smart grid,” in *20th International Conference on Electricity Distribution*, 2009, pp. 8-11.

[53] A. Weidlich and S. Karnouskos, “Integrating Smart Houses with the Smart Grid Through Web Services for Increasing Energy Efficiency,” in *10th IAEE European Conference*, 2009.

[54] “EEBus - Strom intelligent nutzen,” 2012. [Online]. Available: http://www.eebus.de/.

[55] J. M. Gilbert and F. Balouchi, “Comparison of energy harvesting systems for wireless sensor networks,” *International Journal of Automation and Computing*, vol. 5, no. 4, pp. 334-347, 2008.

[56] M. Handley, “Why The Internet Only Just Works,” *BT Technology Journal*, vol. 24, no. 3, pp. 119-129, 2007.

[57] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-end arguments in system design,” *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277-288, Nov. 1984.

[58] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, “Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services,” *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223-235, Jul. 2010.

[59] G. Candido, A. W. Colombo, J. Barata, and F. Jammes, "Service-Oriented Infrastructure to Support the Deployment of Evolvable Production Systems," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 4, pp. 759-767, Nov. 2011.

[60] J. W. Hui and D. E. Culler, "IPv6 in Low-Power Wireless Networks," *Proceedings of the IEEE*, vol. 98, no. 11, pp. 1865-1878, Nov. 2010.

[61] J. Hui, "An Extended Internet Architecture for Low-Power Wireless Networks - Design and Implementation," University of California at Berkeley, 2008.

[62] G. Anastasi, M. Conti, and M. Di Francesco, "A Comprehensive Analysis of the MAC Unreliability Problem in IEEE 802.15.4 Wireless Sensor Networks," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 52-65, Feb. 2011.

[63] L. Lo Bello and E. Toscano, "Coexistence Issues of Multiple Co-Located IEEE 802.15.4/ZigBee Networks Running on Adjacent Radio Channels in Industrial Environments," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 2, pp. 157-167, May 2009.

[64] C. Gomez and J. Paradells, "Wireless home automation networks: A survey of architectures and technologies," *IEEE Communications Magazine*, vol. 48, no. 6, pp. 92-101, Jun. 2010.

[65] J. Davies, *Understanding IPv6*. Microsoft Press, 2002.

[66] C. Bouras, A. Karaliotas, and P. Ganos, "The deployment of IPv6 in an IPv4 world and transition strategies," *Internet Research*, vol. 13, no. 2, pp. 86-93, 2003.

[67] J. G. Jayanthi and S. A. Rabara, "Transition and mobility management in the integrated IPv4 and IPv6 network - A systematic review," in *International Conference on Electronics and Information Engineering*, 2010, pp. V1-162 - V1-166.

[68] M. Durvy et al., "Making sensor networks IPv6 ready," in *6th ACM Conference on Embedded Network Sensor Systems*, 2008, pp. 421-422.

[69] J. Ko et al., "ContikiRPL and TinyRPL: Happy Together," in *Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, 2011.

[70] J. Postel, "User Datagram Protocol," *RFC 768*. Internet Engineering Task Force, 1980.

[71] A. L. Caro, J. R. Iyengar, P. D. Amer, S. Ladha, G. J. Heinz, and K. C. Shah, "Research feature - SCTP: a proposed standard for robust internet data transport," *Computer*, vol. 36, no. 11, pp. 56-63, Nov. 2003.

[72] N. R. Ranga, "Beyond TCP: stream control trasmission protocol," *IEEE Potentials*, vol. 25, no. 1, pp. 16-18, Jan. 2006.

[73] J. Hui and P. Thubert, “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” *RFC 6282*. Internet Engineering Task Force, 2011.

[74] R. Fielding et al., “Hypertext Transfer Protocol-HTTP/1.1.,” *RFC 2616*. Internet Engineering Task Force, 1999.

[75] E. J. Whitehead and Y. Y. Goland, “The WebDAV property design,” *Software: Practice and Experience*, vol. 34, no. 2, pp. 135-161, Feb. 2004.

[76] A. Donoho, J. Costa-requena, T. Mcgee, A. Messer, A. Fiddian-green, and J. Fuller, “UPnP™ Device Architecture 1.1.” UPnP-Forum, pp. 1-136, 2008.

[77] J. Cowan and R. Tobin, “XML Information Set (Second Edition),” *W3C Recommendation*. World Wide Web Consortium, 2004.

[78] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, “XML Schema Part 1: Structures Second Edition,” *W3C Recommendation*. World Wide Web Consortium, 2004.

[79] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition),” *W3C Recommendation*. World Wide Web Consortium, 2008.

[80] E. van der Vlist, *XML Schema*. O’Reilly, 2002.

[81] H. Balzert, *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Heidelberg: Spektrum Akademischer Verlag, 2011.

[82] “Health Level Seven.” [Online]. Available: http://www.hl7.org.

[83] M. Yuksel and A. Dogac, “Interoperability of medical device information and the clinical applications: an HL7 RMIM based on the ISO/IEEE 11073 DIM,” *IEEE Transactions on Information Technology in Biomedicine: a publication of the IEEE Engineering in Medicine and Biology Society*, vol. 15, no. 4, pp. 557-566, Jul. 2011.

[84] I. K. Samaras, G. D. Hassapis, and J. V. Gialelis, “A Modified DPWS Protocol Stack for 6LoWPAN-Based Wireless Sensor Networks,” *IEEE Transactions on Industrial Informatics*, no. 99, 2012.

[85] N. Glombitza, D. Pfisterer, and S. Fischer, “LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices,” in *7th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2010, pp. 1-9.

[86] N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, “Tiny web services,” in *6th ACM Conference on Embedded Network Sensor Systems*, 2008, pp. 253-266.

[87] J. Janecek, "Efficient SOAP processing in embedded systems," in *11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2004, pp. 128-134.

[88] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic wireless sensor networks," in *33rd IEEE Conference on Local Computer Networks*, 2008, pp. 740-747.

[89] I. K. Samaras, J. V. Gialelis, and G. D. Hassapis, "A service oriented-based system for real time industrial applications," in *15th IEEE Conference on Emerging Technologies & Factory Automation*, 2010, pp. 1-8.

[90] "WS4D (Web Services for Devices)," 2012. [Online]. Available: http://www.ws4d.org.

[91] R. Jeyaraman, "DPWS Interop Scenarios." OASIS WS-DD TC, 2008.

[92] C. Bormann, "Guidance for Light-Weight Implementations of the Internet Protocol Suite." IETF, 2012.

[93] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Fourth International Symposium on Information Processing in Sensor Networks*, 2005, pp. 364-369.

[94] F. Aslam, C. Schindelhauer, G. Ernst, D. Spyra, J. Meyer, and M. Zalloom, "Introducing TakaTuka: A Java Virtual Machine for Motes," in *6th ACM Conference on Embedded Network Sensor Systems*, 2008, pp. 399-400.

[95] J. Ko et al., "Low Power or High Performance? A Tradeoff Whose Time Has Come (and Nearly Gone)," in *European Conference on Wireless Sensor Networks*, 2012, vol. 7158, pp. 98-114.

[96] F. Aslam et al., "Optimized Java Binary and Virtual Machine for Tiny Motes," in *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, 2010, pp. 15-30.

[97] P. Levis and D. Gay, *TinyOS Programming*. Cambridge: Cambridge University Press, 2009.

[98] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *29th Annual IEEE International Conference on Local Computer Networks*, 2004, vol. 2004, pp. 455-462.

[99] D. Déharbe, S. Galvão, and A. Moreira, "Formalizing FreeRTOS: First Steps," in *Formal Methods Foundations and Applications*, vol. 5902, M. V. M. Oliveira and J. Woodcock, Eds. Springer Verlag, 2009, pp. 101-117.

[100] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC language," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003, pp. 1 - 11.

[101] S. Pöhlsen, C. Buschmann, and C. Werner, "Integrating a Decentralized Web Service Discovery System into the Internet Infrastructure," in *6th European Conference on Web Services*, 2008, pp. 13-20.

[102] S. Pöhlsen, "Entwicklung einer Service-orientierten Architektur zur vernetzten Kommunikation zwischen medizinischen Geräten, Systemen und Applikationen," Universität Lübeck, 2010.

[103] A. Bobek, E. Zeeb, H. Bohn, F. Golatowski, and D. Timmermann, "Device and service templates for the Devices Profile for Web Services," in *6th IEEE International Conference on Industrial Informatics*, 2008, pp. 797-801.

[104] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks," *1st Conference on Symposium on Networked Systems Design and Implementation*, pp. 15-28, 2004.

[105] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," *RFC 3376*. Internet Engineering Task Force, 2002.

[106] E. Zeeb, G. Moritz, D. Timmermann, and F. Golatowski, "WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services," in *39th International Conference on Parallel Processing Workshops*, 2010, pp. 1-8.

[107] "SOA4D (Service-Oriented Architecture for Devices)," 2012. [Online]. Available: https://forge.soa4d.org/.

[108] R. A. Van Engelen and K. A. Gallivan, "The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks," in *2nd IEEEACM International Symposium on Cluster Computing and the Grid*, 2002, p. 128.

[109] S. Prüter, G. Moritz, E. Zeeb, R. Salomon, F. Golatowski, and D. Timmermann, "Applicability of Web Service Technologies to Reach Real Time Capabilities," in *11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2008, pp. 229-233.

[110] G. Moritz, S. Prüter, D. Timmermann, and F. Golatowski, "Web services on deeply embedded devices with real-time processing," in *IEEE International Conference on Emerging Technologies and Factory Automation*, 2008, pp. 432-435.

[111] C. Lerche, "Implementierung Web Services basierter Kommunikation für Geräte mit starken Ressourcenbeschränkungen," Universität Rostock, 2010.

[112] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security,” *RFC 4347*. Internet Engineering Task Force, 2006.

[113] S. Unger, S. Pfeiffer, and D. Timmermann, “How much Security for Switching a Light Bulb -The SOA Way,” in *Security, Trust and Privacy Symposium 2012*, 2012.

[114] Free Software Foundation Inc, “GCC, the GNU Compiler Collection,” 2010. [Online]. Available: http://gcc.gnu.org/.

[115] M. Kovatsch, S. Duquennoy, and A. Dunkels, “A Low-Power CoAP for Contiki,” in *IEEE Workshop on Internet of Things Technology and Architectures*, 2011.

[116] D. Yazar and A. Dunkels, “Efficient application integration in IP-based sensor networks,” in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for EnergyEfficiency in Buildings*, 2009, pp. 43-48.

[117] A. Hornsby and E. Bail, “uXMPP: Lightweight implementation for low power operating system Contiki,” in *International Conference on Ultra Modern Telecommunications Workshops*, 2009, pp. 1-5.

[118] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core,” *RFC 3920*. Internet Engineering Task Force, 2004.

[119] F. Osterlind and A. Dunkels, “Approaching the Maximum 802.15.4 Multi-hop Throughput,” *5th ACM Workshop on Embedded Networked Sensors*, 2008.

[120] C. Werner, C. Buschmann, Y. Brandt, and S. Fischer, “Compressing SOAP Messages by using Pushdown Automata,” *IEEE International Conference on Web Services*, pp. 19-28, 2006.

[121] C. Werner, *Optimierte Protokolle für Web Services mit begrenzten Datenraten*. Logos Berlin, 2007.

[122] K. Sayood, *Introduction to data compression*. Academic Press, 2000.

[123] P. Deutsch, “GZIP file format specification version 4.3,” *RFC 1952*. Internet Engineering Task Force, 1996.

[124] M.-C. Roşu, “A-SOAP: Adaptive SOAP Message Processing and Compression,” *IEEE International Conference on Web Services*, pp. 200-207, 2007.

[125] B. Martin and B. Jano, “WAP Binary XML Content Format,” *W3C Note*. World Wide Web Consortium, 1999.

[126] “Wireless Markup Language.” WAP Forum, 1998.

[127] S. Lewontin, "Nokia Position Paper: W3C Workshop on Binary Interchange of XML Information Item Sets," *W3C Workshop on Binary Interchange of XML Information Item Sets*. World Wide Web Consortium, 2003.

[128] M. Girardot and N. Sundaresan, "Millau: an encoding format for efficient representation and exchange of XML over the Web," *Computer Networks*, vol. 33, no. 1–6, pp. 747-765, 2000.

[129] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *ACM SIGMOD International Conference on Management of Data*, 2000, vol. 29, no. 2, pp. 153-164.

[130] Iso International Organization For Standardization, "Abstract Syntax Notation One (ASN.1): Specification of basic notation," *ITUT Recommendation X680*. 2002.

[131] ITU-T, "Information technology - ASN.1 encoding rules: XML Encoding Rules (XER)," *ITUT Recommendation X693*. International Telecommunication Union (ITU), 2004.

[132] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E. Pelegri-Llopart, "Fast Web Services." 2003.

[133] J. Cheney, "Compressing XML with multiplexed hierarchical PPM models," *IEEE Data Compression Conference*, pp. 163-172, 2002.

[134] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, no. 4, pp. 396-402, 1984.

[135] V. Toman, "Syntactical compression of XML data," in *Conference on Advanced Information Systems Engineering*, 2004.

[136] S. Hariharan and P. Shankar, "Compressing XML documents with finite state automata," *Tenth International Conference on Implementation and Application of Automata*, pp. 285-296, 2005.

[137] J. Kangasharju, S. Tarkoma, and T. Lindholm, "Xebu: A binary format with schema-based optimizations for XML data," *Lecture Notes in Computer Science*, no. Web Information Systems Engineering, pp. 528-535, 2005.

[138] J. Clark and M. Murata, "RELAX NG Specification," *OASIS Committee Specification*, vol. 3. Oasis, 2001.

[139] "Information technology – Generic applications of ASN.1: Fast Infoset." International Telecommunication Union (ITU), 2005.

[140] P. M. Tolani and J. R. Haritsa, "XGRIND: A Query-friendly XML Compressor," in *18th International Conference on Data Engineering*, 2002, p. 225--234.

[141] "XML Binary Characterization Working Group Public Page," *World Wide Web Consortium*. [Online]. Available: http://www.w3.org/XML/Binary/.

[142] J. Schneider and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0," *W3C Recommendation*. World Wide Web Consortium, 2011.

[143] P. Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," *RFC 1951*. Internet Engineering Task Force.

[144] D. A. Huffman, "A method for the construction of minimum redundancy codes," *Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098-1101., 1952.

[145] C. Bournez, "Efficient XML Interchange Evaluation," *W3C Working Draft*. World Wide Web Consortium, 2009.

[146] D. Peintner, H. Kosch, and J. Heuer, "Efficient XML interchange for rich internet applications," in *IEEE International Conference on Multimedia and Expo*, 2009, pp. 149–152.

[147] M. Gotzmann, "Optimierte Datenkodierungsalgorithmen für ressourcenlimitierte Umgebungen," Universität Rostock, 2012.

[148] M. Rethfeldt, "Optimierte Datenkodierungsverfahren für ressourcenlimitierte Umgebungen," Universität Rostock, 2012.

[149] S. Käbisch, D. Peintner, J. Heuer, and H. Kosch, "Optimized XML-based Web service generation for service communication in restricted embedded environments," in *16th IEEE Conference on Emerging Technologies & Factory Automation*, 2011, pp. 1-8.

[150] M. Gudgin et al., "SOAP Version 1.2 Part 2: Adjuncts," *W3C Recommendation*. World Wide Web Consortium, 2007.

[151] K. Iwasa, J. Durand, T. Rutt, M. Peel, S. Kunisetty, and D. Bunting, "Web Services Reliable Messaging (WS-ReliableMessaging)," *OASIS Standard*. OASIS, 2004.

[152] D. Box et al., "Web Services Addressing (WS-Addressing)," *W3C Member Submission*. World Wide Web Consortium, 2004.

[153] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," *RFC 959*. Internet Engineering Task Force, 1985.

[154] J. Klensin, "Simple Mail Transfer Protocol," *RFC 2821*. Internet Engineering Task Force, 2008.

[155] H. M. Mountain, J. Kopecky, S. Williams, G. Daniels, and N. Mendelsohn, "SOAP Version 1.2 Email Binding," *W3C Recommendation*. World Wide Web Consortium, 2002.

[156] F. Forno and P. Saint-Andre, “XEP-0072: SOAP Over XMPP,” *Draft Standard*. XMPP Standards Foundation, 2005.

[157] Apache Software Foundation, “Apache Software Foundation.” [Online]. Available: http://www.apache.org/.

[158] N. Glombitza, D. Pfisterer, and S. Fischer, “LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices,” in *7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2010, pp. 1-9.

[159] C. Werner, C. Buschmann, T. Jacker, and S. Fischer, “Enhanced transport bindings for efficient SOAP messaging,” in *IEEE International Conference on Web Services*, 2005, pp. 193-200.

[160] G. Gehlen, F. Aijaz, and B. Walke, “Mobile Web Service Communication Over UDP,” in *IEEE Vehicular Technology Conference*, 2006, pp. 1-5.

[161] IBM Corp, “IBM WebSphere MQ,” *International Business*, 2011. [Online]. Available: http://www.ibm.com/software/integration/wmq/.

[162] Microsoft Corp, “Microsoft Message Queuing (MSMQ),” *MSDN Library*, 2011. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms711472.aspx.

[163] J. Wetherill, “Messaging Systems and the Java Message Service (JMS),” *Oracle Corporation*, 2002. [Online]. Available: http://java.sun.com/developer/technicalArticles/Networking/messaging/.

[164] M. Hidell and P. Sjodin, “Performance of NACK-oriented reliable multicast in distributed routers,” in *Workshop on High Performance Switching and Routing*, 2006, p. 7.

[165] C. Bormann, A. P. Castellani, and Z. Shelby, “CoAP: An Application Protocol for Billions of Tiny Internet Nodes,” *IEEE Internet Computing*, vol. 16, no. 2, pp. 62-67, Mar. 2012.

[166] M. Duerst and M. Suignard, “Internationalized Resource Identifiers (IRIs),” *RFC 3987*. 2005.

[167] B. Beichler, “Implementierung eines SOAP over CoAP Bindings für drahtlose Sensorknoten,” Universität Rostock, 2011.

[168] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, “Implementation of CoAP and its Application in Transport Logistics,” in *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, 2011.

A Anhang - Protokollspezifische Erweiterungen der EXI-Grammatik

Devices Profile for Web Service

Das Element *Relationship* beinhaltet das Attribut vom Typ *DeviceRelationshipTypes*, welches auf *xs:union* abgebildet wird. Elemente vom Typ *xs:union* bzw. Elemente, die sich von diesem Typ ableiten, müssen laut EXI-Spezifikation immer als Zeichenkette im EXI-Stream eingebettet werden und können nicht binär kodiert sein. Für die Optimierungen kann der Typ *DeviceRelationshipTypes* entsprechend angepasst und zugleich durch den *xs:enumeration* Mechanismus erweitert werden, um bekannte URIs vorab zu definieren.

WS-Addressing

Der *xs:complexType AttributedURIType* wird unter anderem für die Elemente *To* und *Action* genutzt. Diese Elemente können der DPWS-Spezifikation entsprechend diverse vordefinierte Inhalte enthalten, die mit Hilfe des *xs:enumeration* Mechanismus im Schema definiert werden können.

Der *xs:complexType AttributedURIType* wird weiterhin vom Element *Address* aus dem *xs:complexType EndpointReferenceType* genutzt. DPWS schreibt vor, dass das *Address* Element einem *urn:uuid IRI (Internationalized Resource Identifier)* Format [166] folgen muss. Solche *urn:uuids* sind 128 Bit breite hexadezimale eindeutige Werte. Diese werden normalerweise mit dem Datentyp *xs:anyURI* als Zeichenketten dargestellt, die in kodierter Form eine typische Länge von 45 Byte haben. Diese Werte sind einmalig für jeden Aufruf und daher von automatenbasierten Kompressoren kaum zu optimieren. Da DPWS das *urn:uuid* Format vorschreibt, kann eine kompaktere Darstellung für das Element *Address* vom Typ *xs:unsignedLong* gewählt werden, die lediglich 16 Byte breit ist und dennoch den Informationsgehalt nicht verändert.

Da auch für das *MessageID* Element eine solche *urn:uuid IRI* genutzt werden kann, kann hier die gleiche Optimierung wie für das *Address* Element vorgenommen werden. Im *non-strict* Modus des EXI-Formates sind dennoch Inhalte vom Typ *xs:anyURI* im *urn:uuid IRI* Format zulässig.

WS-MetadataExchange

Die Metadaten eines Gerätes, die während des Discovery-Vorganges übermittelt werden, sind in die *MetadataSections Relationshiop*, *ThisModel* und *ThisDevice* unterteilt. Clients sind dadurch in der Lage, nur einzelne Sections des Gerätes abzufragen. Die Unterteilung der Sections im XML-Dokument wird durch das Attribut *Dialect* angezeigt. Auch für dieses Attribut können mit Hilfe des *xs:enumerations* Werte vordefiniert werden.

WS-Eventing

Der Mechanismus für die Zustellung von Events (vgl. *Event-Notifications*) wird durch das Attribut *Mode* im Element *DeliveryType* angezeigt. Da DPWS mindestens den Push-Mechanismus von der Quelle zur Senke verlangt, kann der Wert, der diesen Modus im Attribut *Mode* anzeigt, vordefiniert werden. Durch den *non-strict* Modus des EXI-Formates können in dem Element zusätzlich weiterhin andere Mechanismen angezeigt werden.

Als Event-Filter muss mindestens der Dialekt *Action* unterstützt werden, der ebenfalls durch einen spezifischen Wert angezeigt wird. Auch dieser Wert ist in der DPWS-Spezifikation vorgegeben und kann in die Automatenstruktur integriert werden.

Für die eindeutige Identifizierung von *Subscriptions* werden im Allgemeinen *urn:uuid IRIs* verwendet. Diese können, wie auch die zuvor in WS-Addressing genannten Elemente, durch kompaktere Datentypen repräsentiert werden.

B Anhang - Anwendungsspezifische Erweiterungen der EXI-Grammatik

Devices Profile for Web Services

Für ein und dasselbe Gerät sind diverse Metadaten über den Lebenszyklus des Gerätes meist konstant. Diese Werte können in den Schemas bereits vorab eingebettet werden. Die betreffenden Elemente sind unter anderem *ServiceID*, *FirmwareVersion*, *SerialNumber*, *Manufacturer*, *ManufacturerUrl* usw.

WS-Addressing und WS-Eventing

Für die protokollspezifischen Optimierungen wurde der Wertebereich, z.B. des WS-Addressing *Action* Elementes, eingeschränkt. Zu den *xs:enumerations* können ebenfalls spezifische Werte der Dienste des Gerätes hinzugefügt werden. Die *Actions* werden von WS-Eventing ebenfalls als Event-Filter genutzt. Auch hier können die spezifischen Methodenaufrufe der Dienste in die Grammatik eingebracht werden.

WS-Discovery

Während des Discovery-Vorganges werden unter anderem die Transportadressen des Gerätes und der Dienste übermittelt. In Abhängigkeit davon, wie die Transportadressen im 6LoWPAN vergeben werden, können auch diese vorab definiert werden und müssen so während des Discovery-Vorganges nicht als Zeichenkette übertragen werden. Es ist zum Beispiel denkbar, dass die Transportadressen aus den Adressen der Sicherungsschicht (engl. Link Layer, vgl. MAC-Adresse) generiert werden und sich damit über den Lebenszyklus des Gerätes nicht ändern. Sind die verwendeten Sensoren und deren Sicherungsschicht-Adressen zuvor bekannt, können diese Informationen über die daraus resultierenden Transportadressen ebenfalls in die Automatenstruktur eingebracht werden. Eine andere Möglichkeit stellt die statische Adressvergabe über DHCP, wobei auf einen definierten Adressraum zurückgegriffen wird. Der *non-strict* Modus des EXI-Formates ermöglicht zur Laufzeit auch hier wieder die Abweichung von den zuvor definierten Adressen.

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Dissertation selbstständig und ohne fremde Hilfe verfasst, die von mir genutzten Quellen und Hilfsmittel angegeben und den benutzten Werken wörtlich oder inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort, Datum: Rostock, 01.11.2012 Guido Moritz

Thesen

1. In Bereichen wie der Gebäudeautomatisierung, Industrieautomatisierung und Ambient Assisted Living wird zukünftig eine starke Verschränkung von Geräteinfrastrukturen mit höherwertigen Diensten vorzufinden sein. Dies beinhaltet teilweise ebenfalls eine Überlappung von bislang entkoppelter Domänen.
2. Innerhalb der Geräteinfrastruktur werden batteriebetriebene Kleinstgeräte als Sensoren und Aktoren eingesetzt werden. Diese Klasse von Geräten konnte unter dem Begriff drahtlose Sensornetzwerke bislang keine nennenswerte Durchdringung erreichen.
3. Drahtlose Sensornetzwerke bilden die ultimative Benchmark für alle Arten von Protokollen, da in diesen Umgebungen sowohl die Bandbreite als auch die Energie und die zur Verfügung stehenden Speicher- und Rechenkapazitäten besonders stark limitiert sind. In einigen Szenarien kommt zusätzlich die Mobilität der Endpunkte erschwerend hinzu.
4. Während zur Vernetzung von ressourcenstarken Geräteensembles mit höherwertigen Diensten z.B. im Internet bereits geeignete Technologien und Protokolle zur Verfügung stehen, sind die drahtlos kommunizierenden und stark ressourcenlimitierten Kleinstgeräte bislang nur durch unflexible und schwergewichtige Stellvertreter, wie beispielsweise Gateways, in Dienstkompositionen integrierbar.
5. Zur Vermeidung solcher unflexiblen Infrastrukturen wird ein starker Trend hin zu IP-basierten Lösungen erkennbar. Durch IP wird bereits auf sehr niedriger Schicht eine Homogenisierung und somit Abstraktion gegenüber dem konkreten Übertragungsmedium und der Netzwerktopologie erreicht. Zwingender Bestandteil der Infrastruktur müssen zustandslose Protokollumsetzer für die Schichten oberhalb von IP sein, die eine transparente Umsetzung zwischen existierenden und neuartigen, ressourceneffizienteren Protokollen ermöglichen.
6. Die 6LoWPAN-Protokolle der IETF bilden die derzeit optimale Lösung hinsichtlich des Einsatzes von IPv6 in drahtlosen vermaschten Sensornetzwerken. IPv4 kann in solchen Umgebungen nicht effizient eingesetzt werden.
7. Bislang existieren für 6LoWPAN-basierte Netzwerke keine Protokolle, die effiziente Gruppenkommunikation mittels IP-Multicast oder Ende-zu-Ende Verschlüsselung (inkl. Authentifizierung und Integritätsprüfung) ermöglichen.

8. Auf der Anwendungsschicht haben sich SOAP-basierte Web Services und das Geräteprofil DPWS bereits als machbar und sinnvoll zur Vernetzung von ressourcenstarken Geräten untereinander bzw. zur Vernetzung der Geräte mit höherwertigen Diensten herausgestellt. Ungeklärt war bislang, ob diese Protokolle genug skalieren, um allgemein in stark drahtlosen Sensornetzwerken und konkret in 6LoWPANs eingesetzt zu werden. Der Erhalt der Konformität zur WS-Architecture und dem WS-* Protokollframework des W3C ist dabei eine nicht zu vernachlässigende Kernanforderung.
9. Bei REST-basierten Protokollen wie HTTP entsteht die Interoperabilität lediglich durch Best Practices und Erfahrungen. Im Gegensatz dazu ist die Interoperabilität bei SOAP und dem WS-* Framework integraler Bestandteil des Designs und mittels dedizierte Architekturprinzipien in einer verbindlichen Spezifikation sichergestellt.
10. SOAP Web Services sind gegenüber Basisarchitekturen wie Ressourcen-orientierten und Service-orientierten Architekturen sowie Architekturstilen wie Representational State Transfer unabhängig zu betrachten, da durch die zugrundeliegende WS-Architecture das atomare Element die Nachricht ist, die sowohl Dienste als auch Ressourcen und deren Manipulation abbilden kann.
11. Im Mittelpunkt einer Gegenüberstellung von SOAs und ROAs steht daher der Vergleich der Architekturprinzipien, nicht aber der Vergleich der Protokolle oder deren Implementierungen. Letztere haben einen erheblichen Einfluss auf die Leistungsfähigkeit der Gesamtlösung. Beim Vergleich der umsetzenden Protokolle muss beachtet werden, dass beide Stile nicht auf ein einzelnes Protokoll beschränkt sind, wie auch ein einzelnes Protokoll verschiedene Stile abbilden kann.
12. Die aufgezeigten konzeptionellen Erweiterungen und Anpassungen sind mit SOAP, DPWS und allgemeiner mit der WS-Architecture des W3C kompatibel, um den Nutzen von IP auf den unteren Schichten nicht durch eine Kapselung auf höheren Schichten abzuschwächen.
13. Grundlage für die in dieser Arbeit aufgezeigten Ansätze bildet die generelle Architektur von SOAP Web Services, bei der Schnittstellenmechanismen, Datenrepräsentation und Transport der Informationsträger entkoppelt voneinander betrachtet werden können.
14. Vor dem Hintergrund des Einsatzes in 6LoWPANs mussten vor allem für DWSP existente Mechanismen des Discovery (Suchen und Finden von Diensten) und des Eventings (asynchrone Benachrichtigungen, Push) konzeptionell weiterentwickelt werden.

15. Zur Verringerung der Nachrichtengröße hat sich eine erweiterte Form des EXI-Formates als optimal herausgestellt. Von den Modi von EXI ist nur der schema-informed uncompressed bit-aligned non-strict Modus bei der Nutzung von SOAP in 6LoWPANs einsetzbar.
16. Mit der Implementierung uEXI, die im Rahmen dieser Arbeit entstanden ist und als Open Source veröffentlicht wurde, entstand die weltweit erste Implementierung von EXI für Zielplattformen von 6LoWPANs. Mittels der in der Arbeit dargestellten Untersuchung und durch uEXI konnte somit erstmals der Nachweis der Machbarkeit für den Einsatz im Kontext von SOAP in 6LoWPANs erbracht werden.
17. Bei der Nutzung von DPWS können durch protokoll- und anwendungsspezifische Erweiterungen der genutzten Automatenstruktur zur Kodierung mittels EXI immense Verbesserungen erreicht werden. Die Erhöhung der Komplexität der Automatenstruktur durch die Einbringung anwendungsspezifischer Elemente hat dabei keinen nennenswerten Einfluss auf den Speicherbedarf der Implementierung.
18. Zum Transport der Informationsträger stehen TCP und UDP-basierte Verfahren zur Auswahl. Bei nativem Unicode XML SOAP-Dokumenten leisten beide Verfahren ähnliche Effizienz. Bei größenoptimierten bzw. komprimierten Kodierungsformen der Nutzdaten sind UDP-basierte Verfahren um ein Vielfaches effizienter.
19. Für einen optimalen Transport von SOAP-Dokumenten wurde mit dem neuartigen UDP-basierten SOAP-over-CoAP Binding eine Grundlage geschaffen, die sowohl effizient und zugleich mit dem existierenden SOAP-over-HTTP Binding und der WS-Architecture kompatibel ist. Zur Erarbeitung des CoAP-Bindings nahm der Autor nennenswerten Einfluss auf den Standardisierungsprozess von CoAP bei der IETF.
20. Die entstandene Open Source Implementierung uDPWS stellt die weltweit erste Implementierung von DPWS für 6LoWPANs dar. Die Implementierung enthält das spezifizierte SOAP-over-CoAP Binding. Somit erbringen uDPWS und diese Arbeit erstmals den Nachweis der Machbarkeit von DPWS für 6LoWPAN-Netzwerke.
21. Durch die Implementierungen konnte nachgewiesen werden, dass sich eine Erhöhung des Rechenaufwandes bei gleichzeitiger Verringerung der benötigen Sende- bzw. Empfangszeit effektiv positiv auf den Energiehaushalt auswirkt. Dies ist ein existierendes Paradigma von drahtlosen Sensornetzwerken und gilt ebenfalls im Kontext des Einsatzes von SOAP in 6LoWPANs.

Kurzfassung

In zukünftigen intelligenten, vernetzten Umgebungen dienen stark ressourcenlimitierte Kleinstgeräte beispielsweise als Sensoren oder Aktoren. Charakteristisch für diese Plattformen sind Parameter wie die drahtlose Kommunikation mit geringer Bandbreite, energiesparender Batteriebetrieb, eine miniaturisierte Bauform, geringer Speicher und niedrige Rechenkapazität. Eine große Hürde bei der Nutzung solcher Kleinstgeräte besteht in der Integration in höherwertige Netzwerke und der nahtlosen Konnektivität mit externen Diensten. Aufgrund der starken Ressourcenbeschränkungen werden für die Kommunikation bislang teilweise proprietäre Protokolle eingesetzt. Zur Kommunikation mit externen Diensten sind daher unflexible und schwergewichtige Protokollumsetzer nötig. Eine direkte Ende-zu-Ende Konnektivität ist nicht möglich.

Um die Infrastrukturen flexibler und skalierbarer zu gestalten, gibt es in aktuellen Forschungs- und Entwicklungsarbeiten einen starken Trend zur Nutzung von Internettechnologien in drahtlosen Sensornetzwerken. Die Grundlage bilden die IETF 6LoWPAN-Protokolle (IPv6 over Low power Wireless Personal Area Network), die eine optimierte Lösung von IPv6 für ressourcenlimitierte Geräte darstellen. Die zur Verfügung stehenden Anwendungsschichtprotokolle aus dem Internet sind allerdings nicht ohne weitere Anpassungen in 6LoWPAN-basierten Netzwerken einsetzbar.

In dieser Arbeit wird eine Vernetzungslösung für 6LoWPANs auf Grundlage von W3C SOAP Web Services vorgestellt. Dazu werden drei Aspekte getrennt betrachtet. Zum einen werden in dieser Arbeit erweiterte, generelle Schnittstellenmechanismen und -funktionen definiert. Hierzu zählen beispielsweise effizientes Discovery (Suchen und Finden von Geräten) und feingranulares Eventing (asynchrone Statusbenachrichtigungen, Push). Zum Zweiten wird die Nachrichtengröße durch Nutzung spezifischer Datenkodierungen minimiert. Zum Dritten wird ein Ansatz vorgestellt, mit dem die größenoptimierten Nachrichten effizient im 6LoWPAN übermittelt werden können. Die Gemeinsamkeit aller Anpassungen und Erweiterungen besteht in dem Erhalt der Konformität zur Web Services Architecture (WSA) des W3C, wodurch sich die drahtlosen Sensornetzwerke nahtlos in WSA-basierte Dienstumgebungen integrieren lassen. Alle im Rahmen dieser Arbeit entstandenen Ergebnisse sind durch vollständig funktionsfähige Implementierungen validiert. Diese Implementierungen sind die ersten ihrer Art und als Open Source frei zugänglich.

Abstract

In future smart environments, sensor and actuator functionalities will be composed of highly resource constrained devices. It is typical for these constrained platforms to communicate wirelessly and with little bandwidth, to have a small form factor, and to operate with scarce resources in terms of memory, computing power and energy to enable battery-driven power supply. A major challenge is a seamless integration of these constrained devices into more powerful and service based networking environments. Due to the aforementioned resource constraints, existing technologies often employ proprietary communication protocols. To enable interoperability with external services, intermediaries are required as converters that are often inflexible and complex to operate and maintain. Direct end-to-end communication between devices is impossible.

To overcome these problems, more lightweight and scalable solutions for wireless sensor networks are required and the usage of technologies, which are already widely used in the Internet, form a promising approach. The basis of this trend are the IETF 6LoWPAN protocols (IPv6 over Low power Wireless Personal Area Network), which provide efficient IPv6 support for wireless sensor networks. However, existing application layer protocols cannot be applied in 6LoWPAN networks without further research.

This thesis presents a protocol to close this gap on application layer of 6LoWPANs. It is based on W3C SOAP Web Services. Therefore, three separated key aspects are examined. First, enhanced and improved general interface mechanisms and functionalities are defined. This for example includes efficient Discovery (finding devices present in a network) and fine grained Eventing (asynchronous notifications, Push). Second, message sizes are reduced by developing a highly compact encoding for XML. Third, a novel approach for efficiently transmitting SOAP messages is presented. The extensions and adaptations described in thesis all have a vital property in common. They ensure conformance to the Web Services Architecture (WSA) of the W3C, providing wireless sensor networks with the capability to be seamlessly integrated in WSA based service infrastructures. All originated results are validated and evaluated by fully functional implementations. These implementations are the first of their kind and are published open source and thus are freely available solutions of the discussed protocols.

curriculum vitae

Persönliche Daten

	Guido Moritz geb. am 23.02.1983 in Kühlungsborn ledig, deutsch
Anschrift:	Neptunallee 7a 18057 Rostock
Telefon:	+49 (0) 176 23 70 79 86
E-Mail	info@guido-moritz.de

Bildung

10/2002 – 10/2007	Studium an der Universität Rostock; Studienfach: Elektrotechnik; Vertiefungsrichtung: Technische Informatik Thema der Diplomarbeit: „Evaluierung und Integration von Echtzeiterweiterungen für Service-orientierte Architekturen“
07/1995 – 07/2002	Fritz-Reuter-Gymnasium Kühlungsborn; Abschluss: Abitur

Berufstätigkeit

seit 09/2011	wissenschaftlicher Projektmitarbeiter an der Universität Rostock Institut für Angewandte Mikroelektronik und Datentechnik Nationales Verbundprojekt „Roadmap Ambient Assisted Living Interoperabilität (RAALI)“ des BMBF
02/2008 – 08/2011	Promotionsstudent (Stipendium) Interdisziplinäre Fakultät der Universität Rostock
11/2007 - 02/2008	wissenschaftlicher Projektmitarbeiter an der Universität Rostock Institut für Angewandte Mikroelektronik und Datentechnik

Forschungsaktivitäten, Projekte und ehrenamtliche Tätigkeiten

IETF	Beteiligung an Standardisierung von Kommunikationsprotokollen bei der Internet Engineering Task Force in den Arbeitsgruppen ROLL, 6LoWPAN und CoRE
SIEMENS	Kooperation mit Industriepartner Siemens ReCoDeNets: IT Infrastructure for resource constrained device networks
WS4D	Forschung im Bereich Web Services for Devices (Open Source Projekte) http://www.ws4d.org
Lehre	Durchführung/Betreuung von Seminaren der Veranstaltungen: Echtzeitbetriebssysteme Prozessrechentechnik Betreuung von Praktika Xenomai Betreuung zahlreicher Beleg-, Studien-, Bachelor-, Master- und Diplomarbeiten
SOCNE	Publicity Chair International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE)
Review	Gutachter bei zahlreichen internationalen Journals, Konferenzen und Workshops