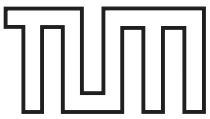


Fakultät für Elektrotechnik und Informationstechnik

Fachgebiet für Betriebssysteme (F13)

Konzeption eines universellen Kommunikationsprotokolls zur Realisierung von Anwendungen im Smart Home

Wolfgang Christoph Hermann Söllner

21. November 2013

Fakultät für Elektrotechnik und Informationstechnik

Fachgebiet für Betriebssysteme (F13), Fakultät für Informatik

Konzeption eines universellen Kommunikationsprotokolls zur Realisierung von Anwendungen im Smart Home

Wolfgang Christoph Hermann Söllner

Vollständiger Abdruck der von der Fakultät für **Elektrotechnik und Informationstechnik** der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: **Univ.-Prof. Dr.-Ing. Eckehard Steinbach**

Prüfer der Dissertation:

1. **Univ.-Prof. Dr. rer. nat. habil. Uwe Baumgarten**
2. **Univ.-Prof. Dr.-Ing. Klaus Diepold**

Die Dissertation wurde am **21. November 2013** bei der Technischen Universität München eingereicht und durch die Fakultät für **Elektrotechnik und Informationstechnik** am 29. September 2014 angenommen.

Danksagung

Mit erfolgreicher Beendigung dieser Dissertation ist es an der Zeit, mich bei denjenigen zu bedanken, die mich während meiner kurzweiligen und lehrreichen Jahre an der Technischen Universität München begleitet haben. Neben meinen ehemaligen Kolleginnen und Kollegen an den Einrichtungen und Lehrstühlen der TUM, mit denen ich enger zusammenarbeiten konnte, möchte ich besonders drei Personen hervorheben.

Herrn Dr. Finkbein verdanke ich zum einen eine Reihe von Kontakten innerhalb und außerhalb der Hochschule, die mir während der Zeit der Dissertation zur Seite standen und mir neben wertvollen Tips vor allem Motivation vermittelten. Zum anderen war mir seine einvernehmende und aufmerksame Art zu diskutieren in unseren Gesprächen stets eine große Hilfe und Anregung, die mir auch über meine Zeit an der TUM hinaus in Erinnerung bleiben wird.

Danken möchte ich auch meinem Zweitgutachter, Herrn Prof. Diepold. Trotz seines unermüdlichen Einsatzes für die Fakultät Elektro- und Informationstechnik und seiner weiteren Verpflichtungen für die Hochschule nahm er sich die Zeit, sich mit dem Thema dieser Dissertation zu auseinanderzusetzen und hat besonders in der Endphase durch seine treffenden Anregungen und seine Unterstützung sehr dazu beigetragen, die Arbeit zu einem erfolgreichen Abschluß zu führen.

Zuletzt möchte ich meinem Erstgutachter und Doktorvater, Herrn Prof. Baumgarten, meinen besonders großen Dank ausdrücken. Nicht nur stand er mir als fachlicher Gesprächs- und kritischer Diskussionspartner stets helfend zur Seite, sondern sorgte auch durch sein freundschaftliches und herzliches Auftreten und sein offenes Ohr für die Wünsche aller Mitarbeiterinnen und Mitarbeiter am Lehrstuhl für genau die positive Atmosphäre, in der das Arbeiten richtig Freude bereitet. Nicht zuletzt wegen seines Engagements, auch in der Lehre, blicke ich auf meine Zeit an der TUM äußerst wohlwollend und sehr gerne zurück.

Abstract

Seit einiger Zeit schon ist aus regenerativen Quellen erzeugte elektrische Energie eine Alternative zu Strom, der aus Kern- und fossilen Brennstoffen erzeugt wird. Mit zu den größten Schwierigkeiten zählt hierbei das unstete Verhalten der Erzeugung, die großteils von Umwelt- und Witterungsbedingungen abhängig ist. Um diesem Nachteil zu begegnen, werden derzeit verschiedene Konzepte zur kurzfristigen und überbrückenden Speicherung von elektrischer Energie entworfen und implementiert.

Da eine Speicherung aufgrund von Energietransport und -umwandlung stets verlustbehaftet stattfindet, beschäftigt sich die vorliegende Arbeit mit Möglichkeiten, elektrische Energie unmittelbar nach Erzeugung zu verbrauchen und so zumindest diese Verluste zu minimieren. Neben planbarem Grundlastbedarf von Großabnehmern bieten dabei insbesondere die Privathaushalte mit tageszeitabhängigem, schwankenden Spitzenlastverbrauchsprofil einen Forschungsansatz, um Energie über den Tag verteilt, dezentral und auf Anfrage abzunehmen.

Es werden Fragestellungen betreffend der Schwierigkeiten und Hindernisse in einem intelligent gesteuerten Haus im übergeordneten Bereich des Smart Grid untersucht. Ziel ist, die Steuerung von elektrischen Verbrauchern möglichst aller Preisklassen und unterschiedlichen Funktionsumfangs zentral innerhalb des Haushalts zu ermöglichen und Letztere auf Wunsch des Benutzers über geeignete, zuverlässige und sichere Methoden und Schnittstellen auch an externe Parteien delegieren zu können.

Neben organisatorischen Herausforderungen werden insbesondere technische Schwierigkeiten (unter anderem derzeitiger Heimsteuerungsprodukte) diskutiert und Lösungsvorschläge schrittweise erarbeitet. Großer Wert liegt dabei auf Verwendung offener Protokolle und Standards, um eine breite und möglichst reibungslose Integration in bestehende Heimnetze und Konzepte zu ermöglichen.

Zur Demonstration wird das gesamte Konzept in eigens entwickelte Hard- und Software umgesetzt und damit eine beispielhafte Systemlösung vorgestellt.

Inhaltsverzeichnis

1 Einleitung

Immer mehr Bereiche unseres täglichen Lebens werden durch Informations- und Kommunikationstechnik positiv beeinflußt. Durch Unterstützung bei alltäglichen Dingen, wie z.B. mit der eMail geschehen, ermöglichen uns technische Geräte und Konzepte, uns auf wesentlichere und für uns wichtigere Dinge zu konzentrieren.

Auch im sogenannten Smart Home, zu deutsch etwa „intelligent gesteuertes Heim“, wird seit einigen Jahren an der Erleichterung von alltäglichen Aufgaben durch technische Helfer gearbeitet. Wie später näher erläutert, beschränkt sich die Unterstützung nicht nur auf die Erhöhung des Komforts der Bewohner, auch andere Anwendungsfelder, wie die Unterstützung der Stromerzeugungsunternehmen, vernetzte medizinische Projekte oder auch Hilfe für ältere Mitglieder unserer Gesellschaft („Ambient Assisted Living“), sind inzwischen denkbar und fallen in den Bereich des Smart Home.

Die vorliegende Arbeit beschäftigt sich mit der „Letzten Meile“ innerhalb eines Smart Home: sie versucht, möglichst vollständige und einfache Antworten auf die Frage der Vernetzung und Steuerung von elektrischen Geräten, gleich welcher Bauart, welchen Typs, welcher Funktion und welchen Herstellers, zu geben.

Dazu wird im ersten Teil ein Überblick über die Motivation und die zugrundeliegenden Techniken vermittelt: nach einem kurzen Einschub über Hintergründe von Stromerzeugung und Lastprofil-Anpassung im sogenannten Smart Grid erfolgt eine Erörterung der technischen Bestandteile eines Smart Home. Anschließend dient eine Diskussion der beteiligten Personen- und Interessensgruppen, die im Verlauf der Arbeit berücksichtigt werden müssen, dem Aufbau der Motivation. Das Kapitel schließt mit der Aufgabenstellung, den Zielen und einem Ausblick auf den weiteren Aufbau der Arbeit.

1.1 Künftige Energieerzeugung im Smart Grid

Eine große Motivation für dezentral flexibel steuerbaren Stromverbrauch geht aus dem Gedanken des Smart Grid hervor: Mit zukünftig fortschreitendem Ausbau von regenerativ betriebenen, dezentralen und dynamischen Stromerzeugungsanlagen erscheint ein verlustbehaftetes Speichern oder Transportieren von Energie immer aufwändiger und weniger wirtschaftlich.

Stattdessen ist Energie möglichst zeit- und ortsnah am Erzeugungsort im Rahmen der sogenannten Laststeuerung zu verbrauchen, gerade um Transport und Zwischenspeichern nach Möglichkeit zu vermeiden. Neben industriellen Großabnehmern, die weitgehend

konstante Lasten („Grundlast“) im Stromnetz darstellen, sind insbesondere private Haushalte mit ihrer steigender Zahl an elektrischen Verbrauchern ein mögliches Ziel für eine intelligente Laststeuerung durch Netzbetreiber.

Im Rahmen dieser Steuerung fallen insbesondere zwei Anwendungsfälle ins Auge: die dezentrale Anpassung des Lastprofils durch Angleichen an ein bestimmtes dynamisches Erzeugungsprofil sowie die Behandlung von Notfällen, beispielsweise bei unvorhergesehenem Ausfall von Stromerzeugungsanlagen oder Leitungsschäden.

1.1.1 Regelung des Lastprofils

Bei industriellen Großabnehmern ist eine Steuerung der Last durch das Stromversorgungsunternehmen bereits möglich. Stromkunden erhalten vergünstigte Konditionen für ihre Einwilligung, unangekündigt für gewisse maximale Zeiträume (Spitzenzeiten) im Zuge der Lastregelung vom Netz getrennt zu werden. Andererseits werden solche Verbraucher gerade in Zeiten geringer Last auch ausgleichend zugeschaltet. Am besten geeignet hierfür sind kurzfristig unterbrechbare Anlagen (z.B. Kieswerke) oder solche, die Energie selbst, möglicherweise in anderen Formen, speichern können (z.B. große Kühlhäuser).

Naheliegend ist, diese Lastregelung auch auf private Haushalte und die darin enthaltenen elektrischen (Groß- oder Dauer-)Verbraucher, wie Kühlschränke oder Herde, auszudehnen. So könnte beispielsweise eine von Privathaushalten verursachte Spitzenlast über den Tag verteilt oder auch dynamisch an die Stromerzeugung angepaßt werden.

Vertraglich könnte eine solche Regelung über eine Obergrenze des Strompreises pro Kilowattstunde gestaltet werden. Mit dem Endkunden wird ein maximaler Preis, z.B. 22 ct/kWh, vereinbart. Sollten nun regional, z.B. durch günstige Witterungseinflüsse bedingt, Überkapazitäten in der Erzeugung entstehen, kann der Stromversorger diesen Preis dynamisch für seine Kunden absenken. In jedem teilnehmenden, *smarten* Haushalt wird eine Steuereinheit, z.B. das Internet-Gateway, über den aktuellen Preis benachrichtigt und kann nun ihrerseits anhand vom Endbenutzer festgelegten Regeln entsprechende elektrische Verbraucher aktivieren.

Sollte der Netzbetreiber nach einer gewissen Zeit keine ausreichende Last im Netz feststellen, kann er durch weiteres Senken des Strompreises erneut einen Anreiz für die Abnahme schaffen. Für das Stromerzeugungsunternehmen hat das den Vorteil, daß die Energie nicht zu negativen Preisen „verkauft“ werden muss. Für den Endkunden bedeutet es eine geringere Stromrechnung.

1.1.2 Netzseitiger Lastabwurf im Störfall

Muß aufgrund eines Störfalles (Unterfrequenz, Unterspannung, oder thermische Überlastung) zwingend Last vom Netz getrennt werden, so besteht bislang für die Stromerzeugungsunternehmen und Netzbetreiber keine Möglichkeit einer differenzierten Auswahl der abzuwerfenden Lasten. Erschwerend kommt hinzu, daß eine solche Trennung in Sekunden

geschehen muß, um den möglichen Zusammenbruch des Gesamtnetzes zu verhindern.

Wie im vorigen Kapitel erläutert, können industrielle Großabnehmer und -lasten bereits geregelt werden. Könnte dieselbe Regelung nun auch in Privathaushalten umgesetzt werden, so würde sich eventuell ein vollständiges Abschalten ganzer Blöcke und Städte erübrigen. Bewohner legen in ihrem lokalen Heimnetz fest, welche Geräte in einem Störfall nicht benötigt werden und entweder abgeschaltet oder pausiert werden können (beispielsweise der Kühlschrank, alle bis auf eine Lampe, etc.).

Stellt der Netzbetreiber einen Störfall fest, der durchaus regional begrenzt sein kann, so werden die Steuerungen in den betroffenen Haushalten innerhalb von Sekunden benachrichtigt und schalten die vorab konfigurierten Geräte ab. Unter Umständen reicht dieser (aktive) Lastabwurf durch die Stromkunden selbst bereits aus, damit essentielle Gerätschaften dennoch aktiv bleiben können und ein erzwungener Lastabwurf etwa durch Abschalten ganzer Blöcke noch nicht notwendig wird.

Der beschriebene Vorgang bietet dem derzeitigen Stand gegenüber keine Nachteile. Ein Netzbetreiber kann nach einer festgelegten Zeit noch immer prüfen, ob der Lastabfall ausreicht und gegebenenfalls durch netzseitig erzwungene Abschaltungen für weitere Reduktion sorgen. Gleichzeitig aber würde den Bürgern eine Möglichkeit eröffnet, sich aktiv und konstruktiv an einer Problemlösung oder zumindest -minderung zu beteiligen.

1.2 Das Smart Home

Eine Studie des VDI/VDE definiert ein Smart Home als „ein privat genutztes Heim (z. B. Eigenheim, Mietwohnung), in dem die zahlreichen Geräte der Hausautomation (wie Heizung, Beleuchtung, Belüftung), Haushaltstechnik (wie z. B. Kühlschrank, Waschmaschine), Konsumelektronik und Kommunikationseinrichtungen zu intelligenten Gegenständen werden, die sich an den Bedürfnissen der Bewohner orientieren. Durch Vernetzung dieser Gegenstände untereinander können neue Assistenzfunktionen und Dienste zum Nutzen des Bewohners bereitgestellt werden und einen Mehrwert generieren, der über den einzelnen Nutzen der im Haus vorhandenen Anwendungen hinausgeht“ [1]. Nach Ansicht des Autors der vorliegenden Arbeit ist diese Definition zutreffend und gilt daher auch als Grundlage der folgenden Ausführungen.

In diesem Teil der Einleitung wird ein Überblick größtenteils über die technischen Gegebenheiten des Smart Homes erarbeitet. Zunächst werden dafür Basis-Komponenten beschrieben, Domänen von Verbrauchern eingeführt, eine Überleitung zu intelligenten Geräten geschaffen und anschließend deren Vernetzung konzeptionell umrissen.

1.2.1 Basis-Komponenten im Smart Home

Innerhalb eines Smart Homes lassen sich aus funktionaler Sicht drei Typen von Komponenten identifizieren:

- *Sensor*-Komponenten haben die Aufgabe, bestimmte Umgebungszustände zu ermitteln und diese in einer geeigneten digitalen Repräsentation im gesamten Smart Home zur Verfügung zu stellen.
- *Aktor*-Komponenten dienen als Befehlsempfänger und bewirken eine Änderung eines oder mehrerer Umgebungszustände.
- Der Verbindung der beiden vorgenannten Bestandteile dient die *Steuerungs- oder Anwendungslogik*, die Sensorwerte aufnimmt und diese nach Regeln und Transformationsvorschriften – auch unter Berücksichtigung der vom Benutzer vorgegebenen Rahmenbedingungen – in Steuerbefehle für Aktoren umsetzt.

1.2.1.1 Sensoren

Unabhängig von jeweiligen Geräten werden in einem Smart Home technische Einrichtungen verwendet, die bestimmte Umgebungszustände aufnehmen (im weitesten Sinne des Wortes *messen*) und diese im Netzwerk in geeigneter Repräsentation bekanntgeben. Diese Komponenten werden als *Sensoren* definiert.

Es muß sich nicht nur um Meßfühler für physikalische Größen handeln:

- So ist beispielsweise auch ein Lichtschalter als „Sensor“ modellierbar, der seine Betätigung und möglicherweise auch die Dauer dieser Betätigung an andere Teilnehmer im Smart Home weiterleitet.
- Gleichzeitig lassen sich auch virtuelle Quellen, etwa ein von außerhalb des Smart Homes übermittelter Strompreis in ct/kWh, als Sensoren betrachten und deren „Meßwerte“ in programmgesteuerte Abläufe integrieren.

Sensoren arbeiten zum einen *ereignisbasiert*. Sie melden selbständig eine Zustandsänderung, worunter auch ein regelmäßiges Verbreiten ihres Meßwerts fällt („Push“-Betrieb). Gleichzeitig können sie zum anderen so konstruiert sein, daß eine Abfrage ihres Zustandes auch direkt ermöglicht ist („Pull“-Betrieb).

Jedem Sensor ist eine innerhalb des Smart Home Netzwerks eineindeutige Identifikation zugewiesen, die dem Bestimmen der Herkunft einer Sensornachricht dient. Besagte Identifikation kann sich aus unterschiedlichen einzelnen Adress-Bestandteilen der Schichten des Open Systems Interconnection (OSI)-Modells zusammensetzen, z.B. der Internet Protokoll (IP)-Adresse eines Gateways, dem jeweiligen User Datagram Protocol (UDP)- oder Transmission Control Protocol (TCP)-Port und einer Adresse des Anwendungsprotokolls.

1.2.1.2 Aktoren

Der zweite Typ von Komponenten empfängt Befehle und setzt sie in eine Zustandsänderung um. Analog zu Sensoren sei hier angemerkt, daß neben Geräten, die physikalisch auf ihre Umwelt einwirken, auch virtuelle Informationssenken, z.B. eine Anzeige auf einem Display oder ein Senden des Wertes per eMail an einen bestimmten Empfänger, als *Aktoren*

bezeichnet und modelliert werden.

Befehle an Aktoren erfolgen in einer geeigneten Repräsentation, die neben der eindeutigen Adresse (siehe voriges Unterkapitel) des oder der Ziel-Aktoren auch den neuen, umzusetzenden einfachen Wert beinhalten. In Kapitel 1.2.4 wird untersucht, welche Netzwerktopologien für diese Form der Kommunikation erforderlich sind.

1.2.1.3 Anwendungslogik

Sensoren und Aktoren können prinzipiell, sofern eine Kommunikationsverbindung zwischen ihnen eingerichtet und ihre Datentypen bzw. deren Repräsentation kompatibel ist, ohne zwischenliegende vermittelnde Instanz miteinander interagieren. In einfachen Geräten und Anwendungsfällen ist das bereits realisiert:

- Ein Lichtschalter bedient eine Lampe.
- Der Drehschalter am Herd steuert die Wärmeabgabe der zugeordneten Herdplatte.
- Ein Wählschalter am Handmixer bestimmt die Geschwindigkeit des Rührwerks.

Diesen Anwendungsfällen ist eine direkte, nicht änderbare Zuordnung von Sensor und Aktor sowie eine statische Transformation des Meßwerts in einen Stellbefehl gemein.

Sollen jedoch komplexere Szenarien mit mehreren Sensoren und Aktoren –möglicherweise über unterschiedliche Geräte hinweg– abgebildet werden, so genügt die statische Zuordnung nicht mehr. Stattdessen sorgt eine *Anwendungslogik*-Schicht für eine dynamische Zuordnung ($1:1$, $m:1$, $1:n$, $m:n$) von Sensoren (m) und Aktoren (n) und transformiert gegebenenfalls die Meßwerte in für Aktoren verständliche Steuerbefehle.

Einige Beispiele einer solchen Logik seien angegeben:

- Ein gekipptes *oder* geöffnetes Fenster senkt die Soll-Temperatur des betroffenen Raumes ab, um so effektiv die Heizung für die Dauer der Öffnung auszuschalten und somit Energie zu sparen.
- Die Flurbeleuchtung wird bei Öffnen der Tür eingeschaltet, aber nur, sofern ein Helligkeitssensor außen einen bestimmten Meßwert unterschritten hat („Dunkelheit“).
- Windempfindliche Lamellenjalousien werden eingezogen, sobald ein Windsensor einen definierten Maximalwert überschritten hat. Gleichzeitig verlieren die Bedientaster in den betroffenen Räumen für die Dauer der Maßnahme ihre Funktion.

1.2.2 Domänen elektrischer Verbraucher

Im zukünftigen Smart Home kommen Geräte mit unterschiedlichen Aufgaben und Funktionen vor, die den Zweck verfolgen, das Leben der Bewohner zu vereinfachen und den Wohnkomfort zu erhöhen. Besagte Geräte lassen sich dazu in Domänen (im Fachjargon „Gewerke“) einordnen, in welchen ähnliche Aufgaben zu erfüllen sind und die auch Geräte leichter klassifizierbar machen.

Durch Anwendungslogik kann nun zwischen Geräten einer Domäne und auch zwischen Geräten unterschiedlicher Domänen eine Interaktion eingerichtet werden, die auf bestimmte Ereignisse hin ausgelöst wird.

1.2.2.1 Beleuchtung

In diese Domäne fallen alle Geräte, die die elektrische Beleuchtung des Smart Home steuern bzw. bewirken. Nicht nur die Helligkeit, sondern auch die Farbe läßt sich inzwischen dank LED-Technik in weiten Bereichen einstellen. Typische Geräte umfassen Lichtschalter, Drehdimmer, abgesetzte Dimmaktoren und letztlich eine Reihe von Lichtquellen.

1.2.2.2 Klima & Heizung

Geräte dieser Kategorie regeln durch Messung und Steuerung das Raumklima. Nicht nur fallen hierunter Temperatur- und Luftfeuchtesensoren, auch Motoren zum Öffnen und Schließen von Fenstern, Vorrichtungen zum Regeln der Heizleistung mittels Ventil- oder Stromsteuerung sowie der Beschattung von Räumen durch elektrische Jalousiemotoren sind eingeschlossen.

Die am häufigsten implementierte Anwendungslogik – oft auch in einem einzigen preiswerten Gerät vereint – stellt die Temperaturregelung von Räumlichkeiten dar, die in komplexeren Installationen auch Fensteröffnungszustände einbezieht und durch Fensterabschattung neben der Heizung zusätzliche Stellglieder bedienen kann.

1.2.2.3 Sicherheit

Unter diese Domäne fallen Geräte und Sensoren, die dem Überwachen des Sicherheitszustandes eines Objektes dienen. Beispiele hierfür sind Drehgriffsensoren für Fenster, Reed-Kontakte für Fenster und Türen, elektrische Türschlösser mit definierten Zuständen, Bewegungsmelder, Rauchsensoren, und viele andere. Je nach Implementierung ist ihre Verbindung mit dem Smart Home Netzwerk durch zusätzliche Maßnahmen (feuerfeste Kabel, stärkere Funksender, etc.) stärker gegen Ausfall gesichert.

1.2.2.4 Umwelt

In diesem Gewerk werden vorrangig Sensoren etabliert, die der Aufnahme von Umweltmeßdaten (Helligkeit, Luftfeuchte, Außentemperatur, etc.) dienen. Eine Klassifizierung der Sensoren gestaltet sich einfach, da eine physikalische Größe stets in eine elektrische Spannung gewandelt und diese mit einer bestimmten Frequenz gemessen wird.

Die Interpretation des Meßwerts findet in Anzeigeeinheiten in Verbindung mit einer physikalischen Einheit oder in anderen Geräten zur Steuerung deren Verhaltens statt.

1.2.2.5 Haushaltsgeräte und weitere Verbraucher

Geräte aus vorangegangenen Domänen lassen sich in der Regel jeweils gut mit gleichen Funktionen und Anforderungen herstellerübergreifend klassifizieren: ein Lichtschalter beispielsweise wird stets eine vergleichbare technische Funktion bieten, egal von wem er produziert wird.

Vorweggenommen hat sich der Vorteil einer herstellerübergreifenden Kompatibilität für die Vernetzung von Heimsteuerungsgeräten, wie später in Kapitel 2 gezeigt werden wird, nicht durchsetzen können. Stattdessen existiert eine Reihe von Insellösungen, in welchen Hersteller Geräte mit vergleichbaren Funktionen für ihre eigene Produktpalette jeweils neu auflegen.

Schon deswegen gestaltet sich daher eine herstellerübergreifende Integration von Haushaltsgeräten, auch als „weiße Ware“ bezeichnet, in ein gemeinsam vernetztes Smart Home ungleich schwieriger. Wegen unzähliger Modelle und Modellvarianten läßt sich in der Regel eine gemeinsame schematische Beschreibung nicht mit vertretbarem Aufwand erstellen.

Haushaltsgeräte beschränken sich dabei nicht nur auf Küchengeräte (Herd, Kühlschrank, Kaffeemaschine, etc.), sondern können auch aus so kleinen preiswerten Verbrauchern wie Tischventilatoren, Staubsaugern oder Luftentfeuchtern bestehen.

Per definitionem fallen für diese Arbeit alle Geräte in die Kategorie „Haushaltsgeräte“, die sich in keine andere Domäne einordnen lassen.

1.2.3 Smart Devices

Aus vorangegangenen Ausführungen wird die Bezeichnung „Smart Device“ abgeleitet und jede Basis-Komponente innerhalb eines Haushalts als intelligent oder „smart“ klassifiziert, die ihre Funktionen ganz oder teilweise über eine maschinen-bedienbare, digitale Schnittstelle zur Verfügung stellt. Der verwendete Kommunikationsweg sei hierfür zunächst unberücksichtigt.

1.2.3.1 Komponenten und logischer Aufbau

In Kapitel 1.2.1 wurde eine getrennte Sicht auf Komponenten eingeführt. In einem Smart Device ist diese Trennung fortgeführt: es besteht aus einer beliebigen Kombination von Sensoren, Aktoren und Anwendungslogik, wie in Abbildung 1.1 dargestellt und erläutert.

Zusätzlich beinhaltet es mindestens ein Kommunikationsmodul zur Verbindung mit dem Smart Home-Netzwerk sowie optional eine Bedienerschnittstelle zur Konfiguration (z.B. „Anlernknopf“, „grafisches Display“, usw.). Die genannten Komponenten sind auf das technische Verständnis des Endbenutzers zugeschnitten.

Die Kommunikationsmodule der Smart Devices ermöglichen nun eine freiere Koppelung von Komponenten aus Kapitel 1.2.1. Es ergeben sich Vorteile:

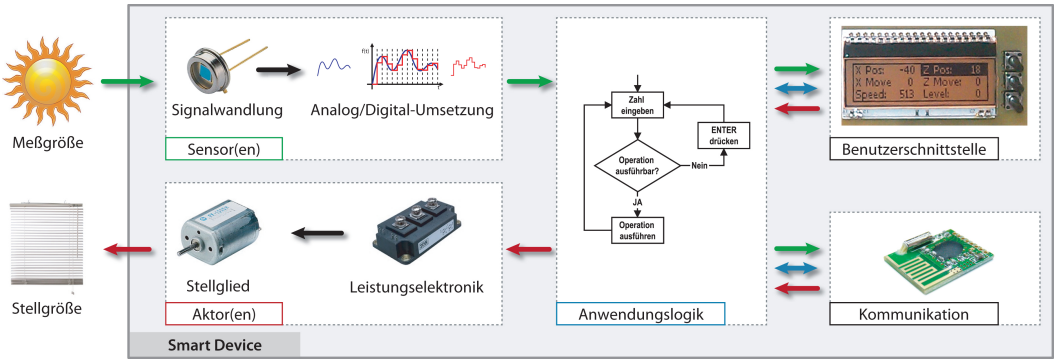


Abbildung 1.1: Komponenten und Bestandteile eines Smart Devices. Es können jeweils ein oder mehrere Sensoren (grün), ein oder mehrere Aktoren (rot) sowie eine Instanz von Anwendungslogik (blau) enthalten sein. Über die Benutzerschnittstelle und das Kommunikationsmodul wird der Zugriff auf Sensoren, Aktoren und Parameter der Anwendungslogik ermöglicht. Besagte Schnittstellen können dabei einem unterschiedlichen logischen Aufbau folgen; beispielsweise steht der Benutzerschnittstelle nur ein interpretierter Zahlenwert (Helligkeit) zur Verfügung, wohingegen die Kommunikationsschnittstelle auch die unverarbeiteten Sensormeßwerte übermittelt.

- *Kostenersparnis durch gemeinsam genutzte Komponenten.* Insbesondere Sensoren, die ihre Meßwerte im gesamten Heimnetz zur Verfügung stellen, können von mehreren Geräten gleichzeitig genutzt werden; es muß nicht mehr jedes Gerät mit einem eigenen Sensor für eine bestimmte Meßgröße ausgestattet werden.
- *Sicherheit durch Redundanz.* Bedingt bei direkt verbundenen Systemen der Ausfall eines Sensors einen Ausfall der gesamten Anlage, so kann dies im Smart Home durch Trennung von Funktionalität und Gerät vermieden werden. Für die zu sichernde Meßgröße sind lediglich mehrere Sensoren zu integrieren. Ein Client wählt aus den vorhandenen Quellen und bleibt bei Ausfall nur einer solchen dennoch funktionsfähig.
- *Erhöhen des Benutzerkomforts durch gesteigerte Automatisierung.* Durch flexibles Festlegen von Regeln auf der Anwendungsschicht können Sensoren beliebig mit Aktoren in unterschiedlichen Geräten verknüpft und somit neue Funktionalitäten im gesamten Smart Home realisiert werden.

1.2.4 Vernetzung

Innerhalb eines Smart Home-Szenarios gibt es unterschiedliche logische und physikalische Netze, die auf teilweise herstellerabhängigen Nachrichtentransportlösungen aufsetzen und sowohl Smart Devices mit dem Heimnetz verbinden wie auch externen Parteien einen Zugriff auf Smart Home-Funktionalitäten ermöglichen (siehe Abb. 1.2 mit den dortigen Festlegungen).

Ausgehend von der physikalischen Verbindung gelangen innerhalb des Smart Home Informationen über eine Reihe von proprietären oder auch standardisierten physikalischen

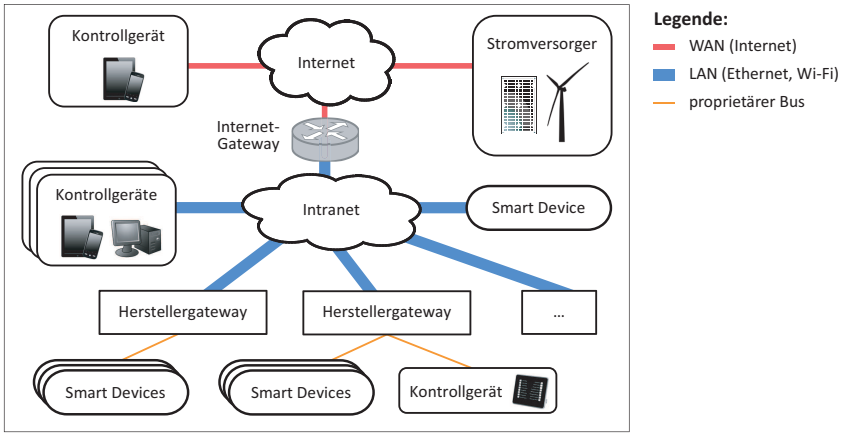


Abbildung 1.2: Komponenten und Bestandteiles einer Smart-Home Umgebung, die mittels IP-basiertem Netzwerk (Intranet) verbunden sind. Erstere sind unterteilt in Benutzerschnittstellen (Kontrollgeräte), Sensoren und Aktoren (Smart Devices), sowie Protokollwandler (Herstellergateways) zur Verbindung zwischen den Smart Devices und dem Intranet. Hochpreisige Geräte können auch direkt IP-Konnektivität aufweisen. Die Liniendicke repräsentiert die verfügbare Kapazität der entsprechenden Verbindung.

Verbindungen, sowie unterschiedlichen Netzwerk- und Anwendungsprotokollen zu Smart Devices.

Im Folgenden werden diese drei notwendigen Teile der Netzwerksicht genauer beschrieben, wobei das ein vereinfachtes OSI-Schichtenmodell als Orientierung dient: Die oberen drei Schichten (*Sitzung*, *Darstellung* und *Anwendung*) werden für den Verlauf dieser Arbeit bei Betrachtung nur der Netzwerkkommunikation zusammenfassend wie im TCP-Referenzmodell der *Anwendungsschicht* zugeordnet, da eine Berücksichtigung von Sitzung und Darstellung im Smart Device selbst geschieht und für die reine Kommunikation keine Bedeutung hat.

1.2.4.1 Informationseigenschaften von Smart Devices

Bevor auf die unterschiedlichen Übertragungsstandards und -Protokolle eingegangen wird, ist ein Blick auf die zu erwartenden Nachrichtengrößen, -Inhalte und deren Auftrittshäufigkeit innerhalb des Smart Home erforderlich. Erst mit zumindest einer groben Abschätzung dieser Parameter ist eine treffende Auswahl und Einschätzung vorhandener Übertragungssysteme oder eine Spezifikation für proprietäre Lösungen möglich. In Tabelle 1.1 werden einige mögliche Meß- und Stellgrößen aufgelistet.

Zu erkennen ist, daß typische Größen mit einem digitalen Informationsgehalt von maximal vier Byte[1] abgebildet werden können; diese Angabe ist insbesondere für die spätere Analyse

1 Anmerkung: Für den weiteren Verlauf der Arbeit dient das *Byte*, definiert als kleinste adressierbare Speichereinheit eines Rechensystems, synonym für den Begriff „Oktett“. Es besitzt einen Informationsgehalt von 8 Bit. In Abbildungen wird bei englischen Beschriftungen der Begriff „octet“ verwendet.

	Einheit	Minimum	Maximum	Bit	Byte
Temperatur	°C, °F	−100,00	240,00	16	2
Rel. Luftfeuchte	%	0	100	7	1
Luftgüte (CO^2-Gehalt)	p.p.m.	0	10 000	14	2
Helligkeit	lx	0	120 000	17	3
Drehzahl	$Umin^{-1}$	−20 000	20 000	16	2
Schalter	*ohne*	0	1	1	1
Dimmer	%	0	100	7	1
UNIX-Zeitstempel	*ohne*	0	0xFFFFFFFF	32	4

Tabelle 1.1: Abschätzung des Informationsgehalts von numerischen Größen im Smart Home in Bit und Byte in jeweils einer geeigneten binären ganzzahligen Repräsentation und gegebenenfalls einer platzsparenden Festkomma-Codierung des Zahlenwertes.

bisheriger Protokolle sowie für den eigenen Ansatz von Bedeutung.

1.2.4.2 Physikalische Verbindungen und Latenz

Smart Devices können für den Informationsaustausch über unterschiedliche kabelgebundene oder kabellose Übertragungstechniken an das interne Netzwerk angebunden sein. In Tabelle 1.2 ist ein Überblick über gebräuchliche Standards und eine Bad-Case-Abschätzung einiger ihrer Eigenschaften gegeben.

Von besonderem Interesse ist die Reaktionszeit eines Smart Devices im Hinblick auf Benutzerkomfort; würde eine Lampe erst eine Sekunde nach Betätigung des Schalters reagieren, wäre das Konzept unbrauchbar. Wie aus der Tabelle hervorgeht, hängt die Reaktionszeit außer bei GSM/EDGE im Wesentlichen vom Smart Device selbst ab. Die aufgelisteten Nachrichtentransportsysteme erscheinen ihrer vernachlässigbar kleinen Latenz wegen geeignet. Anmerkung: Nachdem eine Bedienung über GSM/EDGE voraussichtlich nur in Abwesenheit des Benutzers von außerhalb des Smart Homes stattfindet, ist eine Latenz im Sekundenbereich hier akzeptabel.

Sollten andere proprietäre Kommunikationsmodule und -Techniken eingesetzt werden,

	802.3	802.11	802.15.4	PLC	EDGE
Datenrate (brutto)	>100 Mbit s^{-1}	>2 Mbit s^{-1}	>160 kbit s^{-1}	>1 Mbit s^{-1}	>59,2 kbit s^{-1}
Latenzzeit	<1 ms	>30 ms	>40 ms	*k.I.*	>1000 ms
Reichweite	100 m	100 m	10 m	100 m	3000 m
Reaktionszeit	~20 ms	50,5 ms	110 ms	>21 ms	1155 ms

Tabelle 1.2: Bad-Case-Abschätzung verschiedener Parameter sowie der Reaktionszeit eines Smart Devices unter Verwendung unterschiedlicher Nachrichtenübertragungssysteme bei 1 kB Nachrichtengröße, 20 ms Verarbeitungszeit und einem einzigen Hop in der Verbindung. Eine eventuelle Antwort wird vernachlässigt.

müssen sich diese bezüglich Latenz und Datenrate an obigen standardisierten Techniken orientieren.

1.2.4.3 Zweischichten-Modell der Kommunikation

Bislang verfügbare Heimsteuerungsprodukte sind einfach konstruierte Geräte mit wenigen Funktionen, deren Kommunikationsprotokolle im Allgemeinen keine vollwertige Implementierung aller im OSI-Modell enthaltenen Schichten bieten. Je nach Ausführung verzichten sie beispielsweise auf Sicherheit (Verschlüsselung, Integrität, Vertraulichkeit), Zuverlässigkeit und Routing.

Die Schichten des OSI-Modells können daher für bisherige Produkte zu einem einfacheren Zweischichten-Modell zusammengefaßt werden, in Abbildung 1.3 rechts neben dem Gateway dargestellt. Die untere Schicht ist mit Aufgaben der Sicherung und Integrität betraut, während die obere Schicht ein Byte-orientiertes, proprietäres Anwendungsprotokoll zur Kontrolle angeschlossener Geräte implementiert.

Anders ausgedrückt sorgt die untere Schicht, im rechten Teil von Abbildung 1.2 grau gefärbt, für einen Broadcast-Transport eines einzelnen Byte zu allen Geräten im jeweiligen Netzwerksegment. Aufgaben des Anwendungsprotokolls, rot gefärbt, bestehen in der Adressierung der Geräte sowie der Übermittlung eines Befehls und seiner Argumente.

Die Anbindung an das Heimnetz erfolgt über ein Gateway, das über Standardanschlüsse wie Ethernet oder Wireless Local Area Network (WLAN) verfügt und IP-Fähigkeiten aufweist. Es transformiert Nachrichten zwischen dem Heimnetz und dem Smart Device-Segment sowohl auf Anwendungsebene wie auch auf den unteren OSI-Schichten. Vor- und Nachteile dieses Konzepts werden in Kapitel 2.1.3 ausführlich behandelt.

1.2.5 Abgrenzung zur Gebäudeautomation

Abschließend wird die Steuerung von Geräten im Smart Home von der der Gebäude-

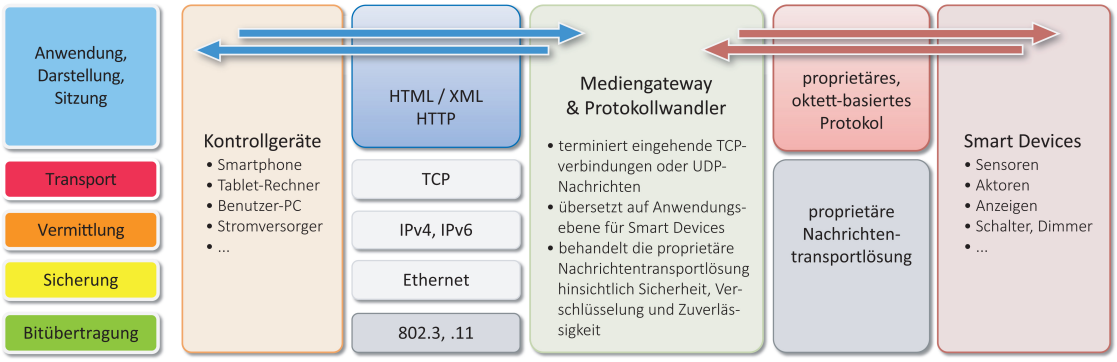


Abbildung 1.3: Vereinfachtes Zweischichten-Modell der Kommunikation zwischen Smart Devices und anderen Heimnetzgeräten.

automation abgegrenzt, da im Smart Home nur eine Teilmenge der Anforderungen der Gebäudeautomation umzusetzen ist. Es ergeben sich die folgenden Unterschiede:

- *Geringere Ausdehnung eines Smart Homes.* Während in einem Bürogebäude oder gar einem großen Industriekomplex große Entfernungen, gut 100 m und mehr, zwischen Geräten überwunden werden müssen, so sind es in einem typischen Smart Home, z.B. einer Doppelhaushälfte, gerade einmal um die 20 m.
- *Geringere Anzahl der zu steuernden Geräte.* Zudem befinden sich in einem Smart Home eine Größenordnung weniger zu steuernde Geräte. Schätzt man pro Raum eine Zahl von 15 bis 20 Geräten (z.B. Heizung/Klima: 3, Licht-Aktoren: 4, Schalter: 4, Jalousien: 5), so läßt sich damit eine Obergrenze der Anzahl angeben. Sie liegt für eine Vierzimmerwohnung mit Küche, Bad, Flur und Gästetoilette bei etwa 150 Geräten ohne „Weiße Güter“. Im Gegensatz dazu ist diese Zahl unter Umständen schon auf einem einzigen Stockwerk eines Bürogebäudes erreicht.
- *Unterschiedliche Gerätetypen.* Die bereits in Kapitel 1.2.2 eingeführten Domänen gewinnen in der Gebäudesteuerung neue Gerätetypen hinzu. Im Bereich der Raumklimatisierung etwa kommen Elemente für Zu- und Abluftsteuerung, bei den Umweltsensoren neben Windgeschwindigkeit und Luftdruck auch gerichtete Helligkeitssensoren zur Markisen- oder Jalousie-Steuerung, und im Sicherheitsbereich weitere Sensoren (Kartenleser) und Aktoren (Türöffner, Parkhausschranken, etc.) hinzu.
- *Geringere Anzahl der logischen Geräte-Netze.* In einem Smart Home kommt eine Installation – ob drahtgebunden oder drahtlos – typischerweise mit einem einzigen Hop und einem einzigen Smart Device-Segment aus. In der Gebäudesteuerung ist die Segmentanzahl bedingt durch die maximal möglichen Teilnehmer an einem Strang größer. Eine Koppelung verschiedener Segmente bedingt damit eine Anbindung an höherkapazitive Netzwerktechniken, wie etwa Ethernet.
- *Unterschiede der verfügbaren Produkte.* Für die Gebäudesteuerung besteht wegen hoher Anforderungen an die Zuverlässigkeit, Betriebssicherheit und Herstellerunabhängigkeit großer Standardisierungswillen, umgesetzt etwa mit Konnex Protokoll-Suite (KNX) und dem älteren European Installation Bus (EIB). Für die Heimsteuerung existieren dagegen ungleich mehr Produkte und Lösungen. Die meisten arbeiten funkbasiert in kostenfrei nutzbaren Bändern (in Deutschland: 433 MHz, 868 MHz und 2,4 GHz). Die Anzahl an verfügbaren Geräten unterscheidet sich von Produkt zu Produkt deutlich, angefangen von simplen Funksteckdosen bis hin zu vollständiger Abdeckung der bereits eingeführten Domänen. Die verwendeten Protokolle der einzelnen Produkte sind zumeist nicht kompatibel.

1.3 Interessensgruppen

Im einzelnen Smart Home treffen unterschiedliche Anforderungen und Erwartungen von

drei Interessensgruppen, *Energieerzeugungsunternehmen*, *Gerätehersteller* und *Endverbraucher*, aufeinander. In diesem Kapitel erfolgt eine Diskussion ihrer für die technische Seite wichtigen Gemeinsamkeiten und Unterschiede, sowie eine Einschätzung der Wichtigkeit der Punkte.

1.3.1 Energieerzeugungsunternehmen

Als Energieerzeugungsunternehmen gilt im Rahmen dieser Arbeit eine Firma, die durch eigene Kraftwerke beliebige Energieformen in elektrischen Strom umwandelt und diesen an Abnehmer verkauft. Der Verkauf muß dabei nicht an Endkunden erfolgen, auch eine Veräußerung an Zwischenhändler, das Energieversorgungsunternehmen (EVU), ist denkbar.

Insbesondere mit der anhaltenden Förderung von regenerativ erzeugtem Strom in verschiedenen Kraftwerkstypen und auch vor dem Hintergrund der absehbar versiegenden fossilen Energieträger (Kohle, Erdöl, Erdgas, etc.) haben die Energieerzeugungsunternehmen zukünftig mit Herausforderungen zu rechnen:

- *Wirtschaftlichkeit und Return-of-Invest.* Regenerativ Strom erzeugende Anlagen sind vergleichsweise teuer und müssen, wie jedes Kraftwerk, durch möglichst hohe Auslastung finanziert werden.
- *Schwankende Stromerzeugung.* Über Wind-, und Solarenergie regenerativ erzeugter Strom ist von seinem Erzeugungsprofil her unstet. Sonnenscheindauer und Windgeschwindigkeit können weder sicher vorhergesagt noch mit heutiger Technik beeinflußt werden.
- *Zeitlich unsteter Stromverkauf.* Auf der anderen Seite stehen Verbraucher, die mit einer konstanten Stromlieferung rechnen. Selbst wenn sich zu bestimmten Tages- und Jahreszeiten Spitzenlasten voraussagen lassen, deckt sich der Bedarf in der Regel kaum mit der Erzeugung.
- *Keine Speicherung, kurze Transportwege.* Transport und Zwischenspeicherung bedingen in jedem Falle Verluste, die der Energieerzeuger – zumindest bei Privatkunden – zu tragen hat. Ein großes Interesse besteht daher in möglichst kurzen Transportwegen und unverzüglichem Verbrauch erzeugter Energie.
- *Dezentrale Erzeugung.* Mit immer preisgünstigeren Stromerzeugungsanlagen (Photovoltaik, kleine Hausstromerzeuger im Kilowattbereich) gewinnt die dezentrale dynamische Erzeugung von elektrischer Energie an Bedeutung. Neben organisatorischen Fragen sind insbesondere die technischen Auswirkungen auf das Energieversorgungsnetz und der Steuerung zu berücksichtigen.
- *Notfallbehandlung und Lastabwurf.* Zur kurzfristigen Regelung des Lastprofils eines Netzes können bereits heute Großverbraucher innerhalb einer kurzen Zeit vom Netz getrennt werden. Der Ausgleich für diese Bereitschaft erfolgt über vergünstigte Strompreise. Könnten diese Kontrollmechanismen bei allen Abnehmern, insbesondere

kleineren Privathaushalten, installiert werden, wären Energieerzeuger zu einer weitaus feinfühligeren Steuerung des Lastprofils des Stromnetzes fähig.

Die genannten Herausforderungen werden hauptsächlich innerhalb des Smart-Grid Konzepts und des Kraftwerkmanagements behandelt. Für den Rahmen dieser Arbeit ist daher der Aspekt der Steuerung von Geräten in Privathaushalten durch Energieerzeugungsunternehmen von Interesse, die im Rahmen der Lastregelung und der Notfallbehandlung stattfindet.

1.3.2 Gerätehersteller

Aus Sicht von Geräteherstellern ist ein Smart Home-Szenario wenig komplex. Die folgenden Annahmen werden über ihre Sicht auf Funktionen und Produkte im Smart Home für ein mögliches Konzept bzw. eine technische Umsetzung im Rahmen der sogenannten *Open Innovation* getroffen [2, Kapitel 3.4 u. 3.5]:

- *Preiswerte Entwicklung (u.a. „Cost-to-Market").* Oberste Anforderung eines Unternehmens in Bezug auf ein eigenes Produkt ist dessen Wirtschaftlichkeit, wozu auch eine möglichst preisgünstige Entwicklung beiträgt. Neben Outsourcing bieten sich auch auf technischer Ebene Möglichkeiten, z.B. durch Wiederverwendung von Hard- und Softwareteilen neue Produkte preiswerter zu realisieren.

- *Möglichst geringe Entwicklungs- und Produktionszeit („Time-to-Market").* Neben einer preiswerten Entwicklung ist auch die Zeit, in welcher ein Produkt auf dem Markt verfügbar ist, relevant. Sie muß so gering als möglich ausfallen.

- *Flexibilität und Hoheit bei Produkt-Design („New-to-Market", „Fit-to-Market").* Hersteller müssen bei Produktdesign und Funktionsumfang Entscheidungsfreiheit besitzen. Keinesfalls darf die Wahl einer technischen Implementierung von Steuerungsplattform oder auch Kommunikationsform eine Einschränkung bedeuten, ansonsten wird sie nicht getroffen und der Hersteller implementiert eine eigene, proprietäre Lösung.

- *Freiheit in der Nutzung von Standards.* Vorhandene Standards werden eingesetzt, wenn sie einen wirtschaftlichen Nutzen, sprich Kostenersparnis, versprechen.

- *Kundenbindung.* Wünschenswert, insbesondere in lukrativen Produktsegmenten, ist für jeden Hersteller eine Kundenbindung. Die einfachste Form in einem Smart Home Szenario ist das Konzipieren einer abgeschlossenen, proprietären Gesamtlösung, in der der Kunde sämtliche Geräte vom Hersteller bezieht. Mittel- und langfristig allerdings besteht eine größere Chance auf Kundenbindung durch Konzepte der Open Innovation: Kostengünstige Werkzeuge für Endbenutzer, eine Internet-unterstützte Community und Innovationswettbewerbe der ein- oder anderen Form.

1.3.3 Endkunden

Wie bereits im letzten Punkt angedeutet, ist der wichtigste Teilnehmer im Smart Home der Endkunde selbst [2, Kapitel 1]. Er muß die technischen Lösungen zum einen verstehen und zum anderen auch die finanzielle Investition in Kauf, Montage, Konfiguration und Wartung tätigen. Daraus leiten sich folgende Anforderungen oder auch Erwartungshaltungen ab:

- *Erhöhen des Komforts und der Sicherheit.* Zunächst einmal sollen die technischen Neuerungen einen (subjektiven) Nutzen bieten. Neben einer Erhöhung des Komforts durch Ersparnis mechanischer Arbeit kann unter diesen Punkt auch die Erhöhung der subjektiven und objektiven Sicherheit dienen, indem Bewegungsmelder bei Abwesenheit Alarm auslösen oder im Rahmen des „Ambient Assisted Living“ ältere Menschen in täglichen Aufgaben durch Automatismus und Monitoring ihres Gesundheitszustandes unterstützt werden.
- *Einfache Konfiguration.* Weiterhin muß eine technische Umsetzung für den Endbenutzer leicht und verständlich zu konfigurieren sein. Neue Geräte, gleich welchen Herstellers, müssen sich nahtlos in eine vorhandene Installation einbinden lassen. Im Idealfall ist die Heimsteuerung offen realisiert, so daß technisch versierte Benutzer eigene Geräte erstellen und einbinden können; insbesondere die Kreativität von Hobby-Ingenieuren hat in der Vergangenheit stets der gesamten Gerätelandschaft genützt. Der bereits eingeführte Oberbegriff lautet „Open Innovation“.
- *Einfache Bedienung.* Hersteller von Geräten und Lösungen müssen mit technisch nicht versierten Bedienern und Endanwendern rechnen. Jedwedes Produkt ist daher in sich stimmig und intuitiv bedienbar zu konzipieren, angefangen von der Kommunikationseinrichtung bis hin zur Benutzerschnittstelle.
- *Übersichtliche Darstellung.* Das Smart Home muß sich dem Benutzer gegenüber transparent und übersichtlich präsentieren. Nicht nur betrifft das Benutzeroberflächen und Geräte; sondern insbesondere sind auch abrechnungsrelevante Details prominent und einfach verständlich aufzulisten.
- *Keine Insellösungen.* Im direkten Widerspruch zur Kundenbindung durch einen Hersteller an seine Produkte steht die Anforderung des Endkunden, möglichst keine Insellösungen erwerben zu müssen. Insbesondere das Nicht-Vorhandensein eines bestimmten, gewünschten Gerätes für ein bestimmtes Heimsteuerungssystem stellt einen Kaufhinderungsgrund dar.
- *Geringer Preis.* Die besten Lösungen und umfangreichsten Produkte nützen wenig, wenn die Mehrheit der Bevölkerung außerstande ist, die initiale finanzielle Belastung zu tragen. Eine geringe Verbreitung von Smart Homes bringt dem Gesamtkonzept „Smart Grid“ keinen Nutzen.
- *Keine Einschränkungen.* Durch den Einsatz von Smart Home-Technik darf der Benutzer nicht eingeschränkt werden. Insbesondere unter Beteiligung Dritter, wie dem Energieversorger, muß eine durchgängige und von besagten Dritten unabhängige Funktion aller Komponenten des Smart Home gewährleistet sein.

- *Finanzieller Nutzen.* Im Idealfall bietet eine Heimsteuerung den Bewohnern auch finanzielle Einsparungen bezüglich Strom- und Heizkosten. Ob dies für alle Haushalte gleichförmig zutrifft, hängt von vielen Faktoren ab und wird nicht untersucht.
- *Wahrung der Privatsphäre.* Bei allen technischen Möglichkeiten müssen dem Endanwender technische und konzeptionelle Garantien vermittelt werden können, daß sein vernetztes Heim zu keiner Zeit unbefugtem Zugriff ausgesetzt ist. Insbesondere ein Übermitteln aller Verbrauchsdaten oder einer Geräteliste eines Haushalts an Dritte (z.B. dem Energieversorger) darf in einem System keine Notwendigkeit sein. Möglichkeiten der dezentralen Steuerung müssen gegeben sein.

Aus dem einfachen Argument, daß der Endkunde letztlich die Finanzierung für das Smart Home-Konzept zu tragen hat, folgt, daß seine Interessen vorrangig zu behandeln sind. Die beste technische Lösung nützt dem gesamten Konzept wenig, wenn sie zu teuer oder zu kompliziert zu bedienen ist und letztlich vom Kunden nicht akzeptiert wird.

1.4 Motivation

Trotz verschiedener Ansätze und Forschungen auch zusammen mit Endbenutzern – wie dem „T-COM Haus“ aus dem Jahr 2005 [3] oder dem übergreifenden Projekt „E-Energy“ [4] – existiert bislang kein zukunftsorientiertes Gesamtkonzept zur Steuerung eines Haushalts mit allen verfügbaren elektrischen Verbrauchern. Lediglich einzelne Bereiche und Domänen werden von einer großen Anzahl von Herstellern bedient. Die Domäne der Haushaltsgeräte ist bisher nur vereinzelt in teuren Endgeräten berücksichtigt und nicht standardisiert.

Auch mit Inkrafttreten des Paragraphen 21b des Gesetzes über die Elektrizitäts- und Gasversorgung [5] im Jahre 2010, durch welches Neubauten und Komplett-Sanierungen zukünftig mit intelligenten Stromzählern („Smart Metern“) auszustatten sind, ist noch nicht absehbar, wie sich diese Technik mit den Anforderungen der Endkunden (Steuerung, Ersparnis, Privatsphäre) vereinbaren läßt. Denn auch im Konzept der Smart Meter existiert bislang kein allumfassender technischer Standard, der eine Steuerung beliebiger Geräte ermöglichte. Nach Meinung des Autors ist das auch nicht wünschenswert: ein Smart Meter sollte einzig dem Zweck der sicheren und zuverlässigen Stromverbrauchsdatenerfassung dienen, mit welcher dynamische Tarife möglich werden. Insbesondere die Notwendigkeit einer für den Endkunden intransparenten Verbindung zwischen Smart Devices und Smart Meter widerspräche dessen Anforderungen.

Die unterschiedlichen Lösungsansätze und Produkte, die dagegen bereits verfügbar sind, stellen zum großen Teil in sich geschlossene und deswegen zueinander nicht kompatible Insellösungen dar, was zwar den jeweiligen Herstellern Vorteile verschafft, für den Endverbraucher aus verschiedenen Gründen aber Nachteile bedeutet.

Da der Status Quo schon seit Jahren anhält und bislang keine allumfassende Lösung in Aussicht steht, besteht insbesondere auf technischer Seite Forschungs- und Handlungsbedarf, um die drei genannten Interessensgruppen sinnvoll zu „verbinden“ und

gewinnbringende Anwendungen überhaupt erst zu ermöglichen.

1.5 Ziele

Um die vorgenannten Punkte umsetzen zu können, wird in dieser Arbeit eine für ein Smart Home aus Kapitel 1.2 passende technische Lösung zur Vernetzung beliebiger elektrischer Verbraucher geschaffen unter Berücksichtigung der Interessen und Anforderungen der drei Interessensgruppen aus Kapitel 1.3.

Aufbauend auf einer Untersuchung vorhandener Produkte ist dafür zunächst zu bestimmen, welche Anforderungen überhaupt innerhalb des Smart Home an die Technik bestehen, und ob und ggf. wie diese Anforderungen bereits mit diesen Produkten erfüllt werden. Sofern dies nicht der Fall ist, ist in weiterer Diskussion eine Eignung von anderen, bereits bestehenden Techniken zu prüfen.

Kann auch hier keine zufriedenstellende Lösung gefunden werden, so ist danach, möglichst aufbauend auf offenen und etablierten Standards und Protokollen, ein Kommunikationsprotokoll insbesondere für den Anwendungsfall „Smart Home“ zu entwickeln:

- Berücksichtigung unterschiedlicher physikalischer Kommunikationstechniken mit verschiedenen externen Parametern (Datenrate, Latenz, Reichweite, etc.)
- Möglichst einfache Integration in bestehende Gerätekonzepte, d.h. Stellen von nur minimalen Anforderungen an die verwendete Hardware (z.B. Kühlschranksteuerung) bei gleichzeitig möglicher Integration in bestehende hoch entwickelte Geräte (z.B. Smart TV)
- Bieten von notwendigen Funktionen, wie beispielsweise das Erkennen und Beschreiben von „Geräten“ und abstrakter ihren „Diensten“

Zur Demonstration und Evaluation ist die implementierte Software anschließend beispielhaft auf Hardware-Demonstratoren zu implementieren. Hierbei liegt der Fokus auf einer möglichst minimalistischen Lösung, deren Eckpunkte durch Hersteller schnell und einfach nachvollzogen werden können:

- Möglichst geringe Hardware-Kosten und eine geringe Time-to-Market durch Wahl geeigneter Mikrocontroller-Plattformen.
- Modularisierung der daran angeschlossenen Peripherie, z.B. Sensoren oder Leistungselektronik.
- Ausreichend Kapazität für *Smart Functionality*, was Kommunikation und Benutzerschnittstelle einschließt.

Außerdem soll eine Software zur Vernetzung von unterschiedlichen Geräten konzipiert und demonstriert werden, die sehr einfach auf Standard-Hardware, z.B. dem Internet-Gateway eines Endbenutzers, implementiert und konfiguriert werden kann. Besagte Software soll neben dem eigens entwickelten Protokoll möglichst auch bereits bestehende Protokolle gängiger Produkte unterstützen, um auch mit diesen Produkten kompatibel zu sein.

In diesem Zusammenhang ist auch ein beispielhaftes Szenario aufzuzeigen, in welchem ein – im Gegensatz zum Smart Meter offener – Internet-Gateway die Kommunikation mit dem Energieversorger über eine bestehende Internetverbindung übernimmt und Befehle zur Steuerung von Geräten nach bestimmten Kriterien und Bedingungen an Geräte im Heimnetz umsetzt.

Abschließend sind sowohl das erstellte Protokoll wie auch die beispielhaften Hardware-Implementierungen nach noch zu definierenden Kriterien zu evaluieren.

1.6 Aufbau

Zu diesem Zweck werden in Kapitel 2 zunächst nach Meinung des Autors vorhandene Lösungen und Produkte von Bedeutung beschrieben und ihre Schwächen, sowohl technischer Natur wie auch hinsichtlich funktionaler Anforderungen im Smart Home, aufgezeigt.

Ein weiterer Teil des Kapitels beschäftigt sich mit gängigen Anwendungsschichtprotokollen. Auch hierbei findet eine Prüfung gegen funktionale Anforderungen statt. Letztlich wird in diesem Kapitel ermittelt, welche der vorhandenen Techniken sich zur Steuerung beliebiger Geräte, auch vor dem Hintergrund der unterschiedlichen Erwartungen der Interessensgruppen, eignet.

Aus den ermittelten Schwächen wird danach in Kapitel 3 ein Konzept und eine Beschreibung für ein universelles Anwendungsprotokoll entworfen, das auf möglichst etablierten und vorhandenen Protokollen aufsetzt und so eine Kompatibilität mit bisheriger Software erreicht. Insbesondere Flexibilität und Erweiterbarkeit spielen beim Entwurf Hauptrolle.

In Kapitel 4 wird das erstellte Protokoll beispielhaft auf einer eigens erstellten Hardware implementiert und seine Tauglichkeit unter Beweis gestellt. Es liegt dabei der Fokus auf einer möglichst einfachen und wirtschaftlichen Lösung zur Konstruktion beliebiger Geräte und deren Integrierbarkeit in ein einziges logisches Smart Home Netzwerk. Die Interessen potentieller Herstellers werden hierbei berücksichtigt und Wert auf eine große Portabilität und Wiederverwendbarkeit der Hard- und Software gelegt.

Bevor der Verlauf der Arbeit zusammengefaßt wird, werden in Kapitel 5 die Ergebnisse bewertet und gegen die bereits vorhandenen Lösungen und Protokolle aus Kapitel 2 verglichen.

Ein Fazit und ein Ausblick auf zukünftige Möglichkeiten und Erweiterungen schließen diese Arbeit im Kapitel 6 ab.

2 Heimsteuerung im Überblick

Ausgehend von den in der Einleitung bereits beschriebenen Eigenschaften von Smart Devices gibt dieses Kapitel im ersten Teil einen Überblick über die derzeit nach Meinung des Autors wichtigen verfügbaren Lösungsansätze, die zur Steuerung von Geräten eines Smart Home eingesetzt werden oder werden können. Es werden ihre Eigenschaften, sowohl des Nachrichtentransports wie auch auf der Anwendungsprotokollebene, beschrieben und verglichen.

Der zweite Teil beschäftigt sich mit reinen Anwendungsschichtprotokollen, die – unabhängig von konkreter Hardware – auf ihre Eignung zur Steuerung von Geräten geprüft werden. Insbesondere ist zum einen von Interesse, ob die höheren Abstraktionsmöglichkeiten dieser Protokolle nennenswerte Vorteile bieten und zum anderen, ob überhaupt eine realistische Möglichkeit der Implementierung auf typischer Smart Hardware besteht.

2.1 Derzeitige Heimsteuerungsprodukte

Einleitend werden in diesem Punkt die der Meinung des Autors nach relevanten Produkte vorgestellt. Kriterien für die Auswahl liegen in einer möglichst großen Anzahl an verfügbaren Geräten, der technischen Umsetzung von Nachrichtentransport, Anwendungsprotokoll und sicherheitsrelevanten Aspekten sowie die öffentliche Verfügbarkeit von technischer Dokumentation.

2.1.1 Generelles technisches Konzept

Allen in diesem Kapitel vorgestellten Produkten und Lösung ist eine vergleichbare Architektur, die bereits in Kapitel 1.2.4.3 eingeführt wurde, gemein. Durch ein gemeinsam genutztes Broadcast-Medium werden die jeweiligen Geräte und optional ein Gateway für andere Netzwerkschichten verbunden, siehe Abbildung 2.1.

Die Kommunikation findet im Halbduplex-Verfahren statt, es kann daher entweder gesendet oder empfangen werden. Außerdem kann stets nur ein Gerät senden, dabei aber – sofern implementiert – mehrere andere Geräte einschließlich des Gateways für den Empfang adressieren.

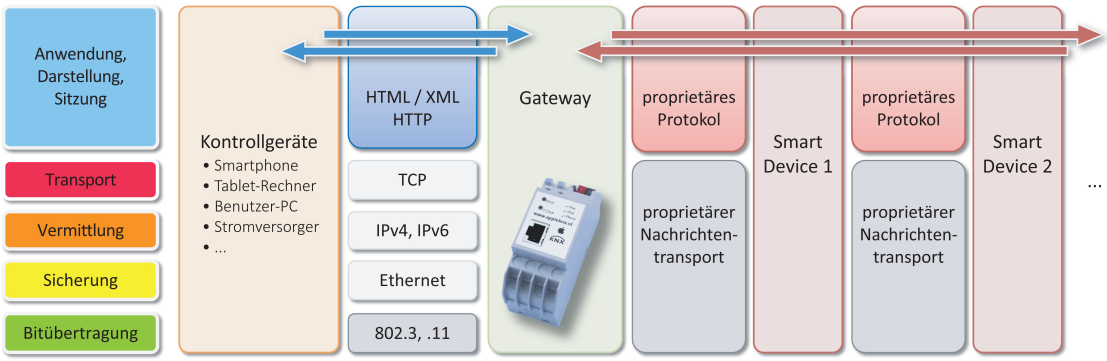


Abbildung 2.1: Anbindung von mehreren Smart Devices über ein gemeinsam genutztes Medium an ein IP-Netz. Die Kommunikation erfolgt im Halbduplex-Verfahren, alle angeschlossenen Teilnehmer können innerhalb eines Segmentes mit Bustopologie alle Nachrichten empfangen.

2.1.1.1 Nachrichtentransport

Geht man von einer Bottom-Up-Ansicht aus, so ist zunächst die Möglichkeit der Kommunikation zwischen Smart Devices zu schaffen. Dazu implementieren Produkte den physikalischen Nachrichtentransport auf OSI-Schicht 1 über drahtlose und drahtgebundene Medien mit unterschiedlichen äußeren Parametern und Eigenschaften.

Insbesondere bei drahtlosen Transportsystemen erfolgt eine Abnahme des gesamten Gerätes auf seine elektromagnetische Verträglichkeit (EMV) hin. Zum einen wird dabei sichergestellt, daß das Gerät und seine Funkschnittstelle nur im zulässigen Frequenzbereich arbeitet und weitestgehend nur dort elektromagnetische Strahlung emittiert. Zum anderen wird die Einhaltung regulatorischer Beschränkungen (z.B. „1 % Duty Cycle“) zertifiziert.

Weil eine solche Abnahme für jeden einzelnen Gerätetyp getrennt zu erfolgen hätte, sind Hersteller zur Verwendung zertifizierter Kommunikationsmodule, beispielhaft gezeigt in Abbildung 2.2, übergegangen. Häufig anzutreffen sind Lösungen, die die Nutzdaten mit Amplituden- oder Frequenzmodulation auf ein Trägersignal aufprägen und dieses auf das jeweilige Medium umsetzen. Im Entwicklungsprozeß entfällt somit der zeit- und kostenaufwändige Schritt der Einzelzertifizierung. Gesamtzulassungszeit und Time-to-Market werden günstig beeinflußt.

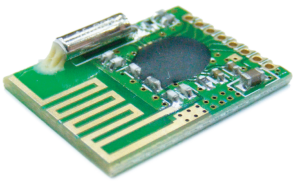


Abbildung 2.2: RFM70 Funkmodul für das 2,4 GHz-Band [6].

Optional werden in den Modulen Dienste aus den OSI-Schichten *Sicherung*, *Vermittlung* und *Transport* direkt implementiert. Mittels Carrier Sense Multiple Access (CSMA) und Collision Avoidance (CA) wird versucht, den durch Kollisionen auf dem Medium verursachten Nachrichtenverlust zu minimieren.

2.1.1.2 Proprietäre, binäre Anwendungsprotokolle

In der Regel ist wegen fehlender Ressourcen keine IP-Konnektivität in Endgeräten vorgesehen. Ein Gateway übernimmt die Aufgabe der Übersetzung, sowohl für die Transport-orientierten OSI-Schichten 1 mit 4, wie auch auf der Nachrichten-orientierten Anwendungsebene. Es besitzt entsprechend umfangreiche Konfigurationsmöglichkeiten, um Nachrichten des proprietären Busses auf IP-Ebene umzusetzen und an bestimmte Empfänger oder auch per IP-Broadcast, z.B. mittels UDP, weiterzuleiten. Außerdem können Nachrichten, die über IP an das Gateway gerichtet werden, auf den proprietären Bus umgesetzt werden und angeschlossene Geräte steuern.

Die verwendeten Protokolle sind in der Regel einfach gehalten, damit der Energieverbrauch, Übertragungs- und Verarbeitungszeit so gering wie möglich ausfallen. Nachrichten, Steuerbefehle und Zustandsänderungen werden in Protocol Data Units (PDU), auch *Telegrammen*, versendet. Neben der Zieladresse und optional der Absenderadresse beinhaltet ein Telegramm die Nutzdaten und optionale Redundanz zur Fehlererkennung mittels Cyclic Redundancy Check (CRC). In den Nutzdaten werden Befehl oder Zustand sowie optionale Argumente entsprechend binär codiert übermittelt. In der Regel wird pro Telegramm nur eine einzige Größe übertragen.

Einige Protokolle implementieren neben einer Fehlererkennung auch eine Fehlerbehebung durch Bestätigung und Neuübertragung eines Telegramms. Wird ein Telegramm am Empfänger nicht erkannt, so übermittelt der Absender dieses nach einer festgelegten Zeit erneut. Ein Zähler überwacht die Anzahl der Versuche und bricht den Vorgang nach einem erreichten Maximum ab. Die Anwendung wird dann über den nicht behebbaren Fehler unterrichtet.

Anzumerken ist, daß in diesem Falle letztlich immer der Benutzer über das weitere Vorgehen zu entscheiden hat: so könnte er z.B. den Lichtschalter einfach ein zweites Mal betätigen, wenn alle Übertragungen der ersten Betätigung verlorengegangen waren.

2.1.1.3 Sicherung für Multi- und Broadcast-Kommunikation

Falls der Nachrichtentransport bzw. das Kommunikationsmodul fehlerhafte Telegramme (z.B. unpassende Länge, falsche Prüfsumme) identifiziert und diese nicht automatisch verwirft, kann ein Non-Acknowledge (NACK) basierter Bestätigungsmechanismus verwendet werden. Dabei fordert die Anwendung des Smart Devices nur im Fehlerfalle ein bestimmtes Telegramm erneut an, bestätigt aber nicht den korrekten Empfang der übrigen Telegramme.

Verwirft dagegen der Nachrichtentransport fehlerhafte Telegramme automatisch ohne Benachrichtigung der Anwendung, so bieten sich Protokolle an, die von jedem (vorab bekannten) Empfänger eine Bestätigung – Acknowledge (ACK) basiert – jedes Telegramms erwarten und dieses bei Ausbleiben der Bestätigung erneut übertragen.

2.1.2 Gängige Implementierungen und Produkte

Die folgenden Abschnitte behandeln jeweils ein gängiges Heimsteuerungsprodukt. Neben technischen Eigenschaften wird auch auf die Arten und die Anzahl der verfügbaren Geräte eingegangen. Falls möglich, wird zusätzlich eine Einschätzung abgegeben, inwieweit sich die behandelten Protokolle erweitern lassen bzw. wie offen sie sind für eigene Entwicklungen. Insbesondere vor dem Hintergrund, daß ein Hersteller eine bestimmte Funktion nicht von sich aus in einem Gerät anbietet, gewinnt der Aspekt der Open Innovation an Bedeutung.

2.1.2.1 Z-Wave

Die aus inzwischen (2013) über 160 Mitgliederfirmen bestehende Z-Wave-Alliance [8] bietet im Umfeld des Smart Homes Produkte und Lösungen basierend auf dem eigens entwickelten Media Access Control (MAC)/Physical Layer (PHY) Protokoll *Z-Wave*. Seit 2012 ist es von der International Telecommunication Union (ITU) unter Empfehlung *G.9959* [9] standardisiert. Die Anwendungsschicht ist allerdings nicht offengelegt. Eine Mitgliedschaft in der Z-Wave-Alliance ist zur offiziellen Einsichtnahme erforderlich.

Die physische Schicht moduliert Nutzdaten über Frequency Shift Keying (FSK) und Manchester- oder Non-Return to Zero (NRZ)-Codierung auf den Funkkanal auf. Sie unterstützt Bitraten von 9,6 kbit s^{-1}, 40 kbit s^{-1} und 100 kbit s^{-1}, wobei nicht alle Datenraten auf allen Frequenzen verfügbar sind [10].

Tabelle 2.1 gibt einen Überblick über die verwendeten Funkfrequenzen je nach Land oder Ländergruppe. Es kann auf drei Kanälen innerhalb des jeweiligen Frequenzbandes gesendet werden, wobei als Kanalabstand 40 kHz definiert sind. Als Vorteil führt Z-Wave an, nicht im 2,4 GHz-Band zu operieren und Störungen durch WLAN nicht zu unterliegen.

Abbildung 2.3 zeigt eine typische, beispielhafte Protocol Data Unit (PDU) (Telegramm) auf PHY- und MAC-Ebene. Nachrichten werden bei Z-Wave innerhalb eines Netzwerks, bestehend aus maximal 238 Knoten, ausgetauscht. Sind mehr Geräte einzubinden, so müssen designierte routingfähige Knoten die Aufgabe der Weiterleitung zwischen den Netzen übernehmen. Essentielle Nachrichtentypen umfassen eine Schreibanforderung, eine Leseanforderung und das Ergebnis einer Leseanforderung an Geräte.

Geräte und Befehle sind in jeder Protokollversion statisch nach Funktion und Gerätetyp klassifiziert und werden binär in den Nutzdaten codiert. Damit ein Kontrollgerät einen Aktor bedienen kann, müssen sie mindestens die gleiche Protokollversion implementieren, wobei Rückwärtskompatibilität besteht. Nicht unterstützte Befehle werden ignoriert.

AU	NZ	BR	EU	HK	IN	JP	MY	RU	SG	USA
921,42	921,42	921,42	868,42	919,82	865,22	922-926	868,10	869	868,42	908,42

Tabelle 2.1: Verwendete Funkfrequenzbänder im Z-Wave-Protokoll in MHz [7].

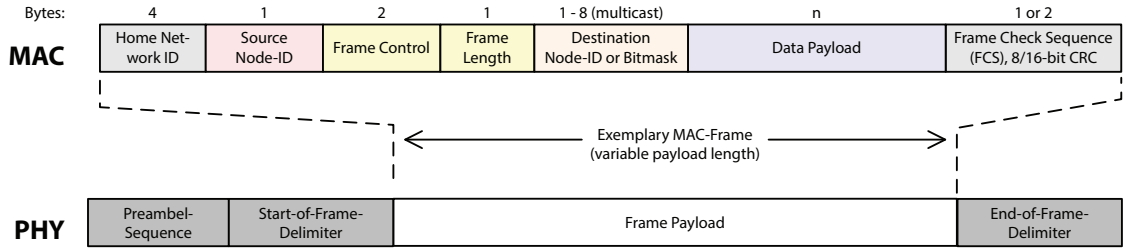


Abbildung 2.3: Frameformat im Z-Wave-Protokoll. Die erste Byte-Gruppe gibt die Netzwerk-ID an. Es können nur Geräte (Smart Devices, Gateways) mit gleicher Netzwerk-ID direkt miteinander kommunizieren. Jedes Gerät erhält eine 8 bit lange Adresse. Verschiedene Rahmenoptionen (Kanal, Bestätigung erwartet, etc.) werden in darauffolgenden Bytes festgelegt. Nach der Gesamtlänge des Rahmens folgt die ID des Zielteilnehmers oder, bei Multicast-Betrieb, eine Bitmaske der angesprochenen Teilnehmer. Das gesamte Telegramm ist abschließend mit einer CRC8 oder CRC16 geschützt.

Eine Verschlüsselung ist derzeit nur bei sicherheitskritischen Anwendungen (Türschlössern) in Gebrauch. Durch einmaligen Schlüsselaustausch während des Anlernens zwischen Aktor und Kontrollgerät (elektronischer Schlüssel) ist diese Verknüpfung statisch. Eine Verschlüsselung sämtlicher Kommunikation ist nicht vorgesehen.

Laut der Z-Wave-Alliance sind derzeit um die 700 Produkte verfügbar, keines allerdings aus der Domäne der Haushaltsgeräte. In Deutschland spielt Z-Wave eher eine untergeordnete Rolle, wobei keine Daten über die Zahl der Installationen vorliegen.

2.1.2.2 ZigBee, ZigBee Pro

Die ZigBee-Alliance [11] spezifiziert das gleichnamige Protokoll aufsetzend auf dem Funkstandard 802.15.4 [12] der Institute of Electrical and Electronics Engineers (IEEE). Sie besteht im Jahr 2012 aus über 400 Mitgliedsunternehmen, die insgesamt über 600 Produkte aus den bereits genannten Domänen entwickelt haben und vertreiben.

ZigBee bietet neben einem standardisierten Funkprotokoll auch eine Reihe von Spezifikationen für Geräte und Nachrichteninhalte nicht nur für das Smart Home, sondern auch für andere Bereiche [13]:

- *building automation.* Definiert Profile und Nachrichteninhalte für Gebäudeautomation, geeignet insbesondere für eine große Zahl (>1000) an möglichen Geräten in einem Netzwerksegment. Verfügbare Produkte: <10.
- *health care.* Definiert Profile für Gesundheitssensoren (Blutdruckmessung, Gewicht, etc.), besonderer Fokus liegt auf batteriebetriebenen Geräten. Verfügbare Produkte: <10.
- *home automation.* Geeignet, um typische Heimgeräte zu vernetzen (Schalter, Dimmer, Jalousien, etc.); batteriebetriebene Geräte werden unterstützt. Verfügbare Produkte: ~50.

- *input device.* Dieser Standard ist speziell für batteriebetriebene Eingabegeräte (Tastaturen, Mäuse, etc.) vorgesehen. Verfügbare Produkte: <5.
- *light link.* Als Unterkategorie von Gebäude- und Heimsteuerung dient der Standard für die Beleuchtung. Neben Profilen für reguläre und LED-Lampen werden auf Helligkeit, Farbtemperatur und unterschiedliche Farbkanäle berücksichtigt. ~30 Produkte verfügbar.
- *network devices.* Netzwerkgeräte stellen im ZigBee-Verbund die Gateways zu anderen Netzwerktechniken (Ethernet, etc.) und -Protokollen dar. Neben eventuell notwendiger Umsetzung auf dem physikalischen Medium transformieren sie insbesondere die binären ZigBee-Inhalte in andere Protokolle. Verfügbare Produkte: ~20.
- *remote control.* Zielt darauf ab, Infrarot-basierte Fernbedienungen abzulösen. Die definierten Profile können aber auch in Home-Entertainment-Geräte integriert werden, um ihnen eine Fernbedienung anderer Produkte zu ermöglichen. Verfügbare Produkte: <5.
- *retail services.* Ein eher noch im Entwurfsstadium steckender Standard für die Warenwirtschaft. Keine Produkte verfügbar.
- *smart energy.* Hierin sind Profile für intelligente Zähler (Strom, Gas, Wasser), Stromkosten-Displays, Pumpen und andere aufgeführt. Verfügbare Geräte: >100.
- *telecom services.* Breit gefaßt ist die Definition dieses Standards, die neben Spielkonsolen alle Geräte zur Informationsübertragung erfaßt. Verfügbare Geräte: <20.

Die einzelnen Nachrichtenformate und Eigenheiten der jeweiligen Spezifikationen aufzuführen oder das ZigBee-Protokoll insgesamt vorzustellen, ist im Rahmen der Arbeit nicht möglich. Es sei an dieser Stelle auf einen Vortrag von D. Stevanovic [14] mit detaillierteren Informationen über ZigBee und den umfangreichen Standard (offiziell nur auf Anfrage verfügbar) verwiesen.

ZigBee bietet eine integrierte Authentifizierung und Verschlüsselung von Nachrichteninhalten (nicht allerdings Metadaten wie MAC-Adressen oder Netzwerk-IDs) auf dem Funkkanal. Vertrauenszentren stellen eine automatisierte Schlüsselbehandlung (Austausch, Erneuerung, etc.) sicher. Auch ein Bootstrapping, d.h. einen nicht konfigurierten Client in ein verschlüsseltes Netz zu integrieren, wird unterstützt. Da ein Angreifer Nachrichteninhalte nicht mitlesen kann, sind derzeit nur die Metadaten (Quelle, Ziel, Häufigkeit) einer Kommunikation offen, wenn die Verschlüsselung aktiviert ist.

Wichtig für die Arbeit ist, die folgenden Schwachstellen von ZigBee zu erläutern:

- Geräte unterschiedlicher Spezifikationen können nicht miteinander kommunizieren. Ein Gateway ist erforderlich, um mit Hilfe von programmierter Funktionslogik auf Anwendungsebene zwischen ihnen zu übersetzen.

- Um Geräte, die dem ZigBee-Standard insgesamt entsprechen, vertreiben zu dürfen, müssen sie zertifiziert werden. Eine kostenpflichtige Mitgliedschaft in der Allianz ist dafür Voraussetzung [15].

- Auch ZigBee definiert in seinen Profilen statische Geräteklassen und -Attribute. Im Falle von Beleuchtung oder Klimageräten ist das vertretbar. Für die Domäne der weißen Ware allerdings ist dieses Konzept aufgrund der Vielzahl der möglichen Geräte nicht geeignet.
- Zur Übertragung von Nachrichten ist genau ein einziger Standard, 802.15.4, vorgesehen. Es bleibt dabei unberücksichtigt, welche Features (Netzwerktopologie, Full-Features-Devices, etc.) überhaupt aus dem Standard benötigt werden. Mit Kosten von knapp 19 € [16] für ein einzelnes ZigBee-Modul ist das Protokoll für Geräte, deren Gesamtpreis im Baumarkt 10 € nicht erreicht („Toaster"), ungeeignet.
- Da ein Netzsegment bei 802.15.4 über einen einzigen Knoten, den Koordinator, verwaltet wird, kann bei Ausfall dieses Koordinators nicht mehr mit und zwischen den Geräten dieses Segments kommuniziert werden.

2.1.2.3 EnOcean

Das besondere Alleinstellungsmerkmal des EnOcean-Protokolls [17] liegt in Geräten, die keine permanente Stromversorgung durch Batterien oder das Stromnetz besitzen und stattdessen den zum Betrieb erforderlichen Strom aus der Umwelt durch Umwandlung aus thermischer oder mechanischer Energie gewinnen.

Die ersten Geräte waren Schalter, bei deren Betätigung ein Piezo-Element den für das Funktelegramm notwendigen Strom erzeugte. Inzwischen ist dieses Konzept um eine Energiegewinnung basierend auf Peltiertechnik oder kleinen Photovoltaikzellen ergänzt. Das gesamte Portfolio umfaßt über 1100 Produkte [18] von knapp 100 Herstellern.

Seit März 2012 ist das Protokoll von der International Electrotechnical Commission (IEC) bzw. der International Organisation for Standardisation (ISO) unter der Bezeichnung ISO/IEC 14543-3-10 standardisiert und spezifiziert die OSI-Schichten 1 mit 3. Der Standard an sich ist allerdings nicht kostenfrei öffentlich zugänglich. Ein Entwurf aus dem Jahr 2011 ist allerdings unter [19] verfügbar.

Eine Nachricht bzw. ein *Frame* des Protokolls wird in drei Schritten konstruiert:

1. *Telegramm.* Beinhaltet die Information der Vermittlungsschicht und die Nutzdaten.
2. *Subframe.* Repräsentiert ein speziell codiertes, einzelnes Byte des Telegramms, das mit zwei Bit Redundanz versehen wurde (siehe Abbildung 2.4).
3. *Frame.* Fügt mehrere Subframes zur letztlich gesendeten Nachricht in Bit zusammen und versieht sie mit Start- und Endinformation (siehe Abbildung 2.5).

Nachrichten werden amplitudenmoduliert auf den lizenzfreien Funkbändern 315 MHz oder 868 MHz übertragen (siehe Abbildung 2.5). Durch eine feste Datenrate von 125 kbit s^{-1} und entsprechende Synchronisation muß nur für die 1-Werte eines Frames Energie aufgewendet werden; bei 0-Werten wird keine Trägerwelle gesendet. Die Codierung von Subframes aus den Telegramminhalten ist in Abbildung 2.4 erklärt.

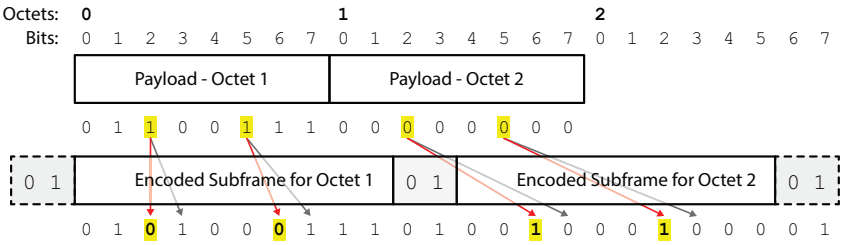
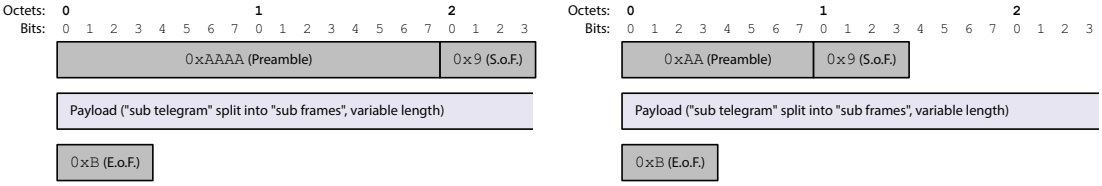


Abbildung 2.4: Darstellung des Subframe-Encoding im EnOcean-Protokoll. Das jeweils zweite und fünfte Bit eines Nutzbyte werden invertiert vor diesen eingefügt [19, S. 19], rote Pfeile. Damit wird auch bei Übertragung von `0x00` oder `0xFF` eine Synchronisation des Empfängers unterstützt, der ansonsten entweder gar kein Signal oder ein Dauersignal erhielte. Anschließend wird an jedes bis auf das letzte so codierte und erweiterte Byte die Sequenz `0b01` angefügt. Auf das letzte Byte folgt der End-of-Frame-Delimiter.



(a) Frameformat bei 315 MHz. **(b)** Frameformat bei 868 MHz.

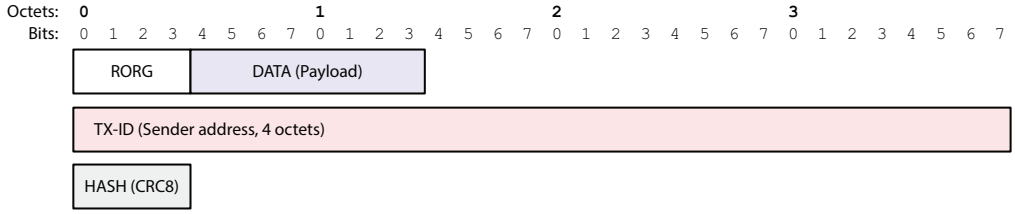
Abbildung 2.5: EnOcean überträgt Informationen in Frames. Nach einer Präambel zur Synchronisation des Empfängers beginnt die Start-of-Frame-Sequenz, `0b1001`. Darauf folgen die einzelnen Byte eines Telegramms, codiert in Subframes. Mit der Bitsequenz `0b1011` endet ein Frame. Der Unterschied zwischen den beiden Frequenzbändern liegt in der Länge der Präambel.

Jedes Gerät besitzt eine eineindeutige, 4 Byte lange Adresse. Sie wird in einem Telegramm als Absenderadresse übertragen. Es existieren verschiedene Telegrammtypen, die in Abbildung 2.6 dargestellt sind. Die zu übertragenden Nutzdaten sind für alle EnOcean-Geräte in den Equipment-Profiles definiert [20].

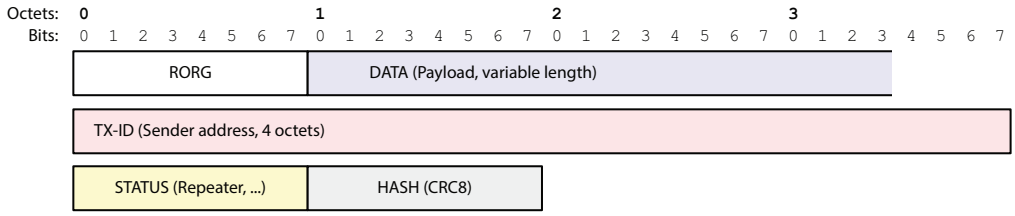
Kollisionen können bei batterielosen Geräten wegen des größeren Aufwands von CSMA/CA („Listen-before-Talk“) nicht vermieden werden; eine Bestätigung oder erneute Übertragung findet in diesem Fall nicht statt. Es wird stattdessen versucht, durch die geringe Länge eines Frames die Wahrscheinlichkeit für Kollisionen zu minimieren.

Geräte, die über das Netz mit Strom versorgt werden, können als Repeater für Nachrichten dienen. Die Funktion muß explizit aktiviert und parametrisiert werden. Dabei werden Geräte am Rande des Netzwerks durch den Installateur entweder als „Level-1“ oder „Level-2“ Repeater konfiguriert. Level-1 Geräte leiten nur erstmalig versendete Telegramme weiter, Level-2 Repeater leiten auch bereits einmal weitergeleitete Telegramme weiter.

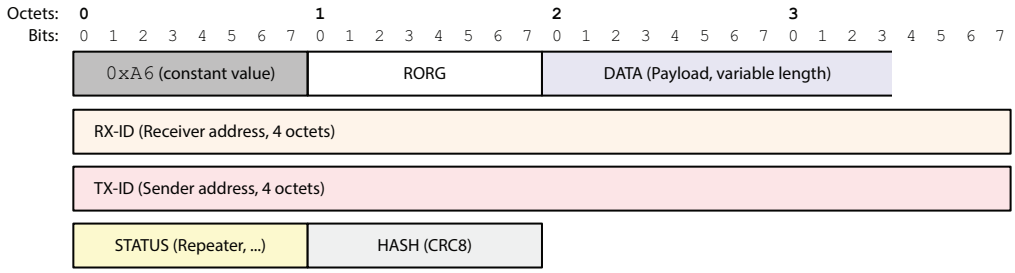
Im Protokoll werden unterschiedliche Sicherheitsfunktionen unterstützt [21], die optional und unabhängig aktiviert werden können:



(a) Telegrammaufbau für kleinste, batterielose Geräte (Schalter).



(b) Reguläres Telegramm, erweiterte Felder.



(c) Reguläres Telegramm mit Zieladresse.

Abbildung 2.6: Die Abbildungen zeigen verschiedene Telegrammformate, die bei EnOcean Verwendung finden. Nach einer Typeninformation `RORG` folgen die Nutzdaten, welche für jedes Gerät separat definiert werden. Auch enthalten ist stets die Absenderadresse und eine CRC-8 Prüfsumme. Im `STATUS`-Feld werden Informationen zum wiederholten Übertragen oder Weiterleiten codiert. Insbesondere für Bestätigungen von Geräten wird das erweiterte Format gewählt, das eine Zieladresse für Unicast-Kommunikation aufnimmt.

- Verschlüsselung der Nutzdaten (payload).
- Rolling-Code gegen Replay-Angriffe (`RLC`).
- Nachrichten-Authentizität, Cipher-based Message Authentication Code (CMAC). Der Austausch der statischen privaten Schlüssel findet über spezielle Telegramme statt.

Das ursprüngliche Alleinstellungsmerkmal der Energiegewinnung durch Piezo-Elemente hat an Bedeutung verloren; Sensoren wie Fenstergriffe oder Wetterstationen können auch durch Photovoltaik mit Energie versorgt werden, und andere Bereiche sind dank Anbindung an das Stromnetz von der Problematik nicht betroffen.

Auch in diesem Protokoll existiert keine Möglichkeit, flexible und dynamische Definitionen von Geräten zu etablieren. Haushaltsgeräte sind nicht verfügbar.

2.1.2.4 HomeMatic

Der Hersteller eQ3-AG bietet mit dem Produkt HomeMatic [22] eine der wenigen Lösungen im LowCost-Bereich an, für die eine große Zahl an Endgeräten verfügbar ist. Außer für die Domäne der Haushaltsgeräte sind aus allen Bereichen Sensoren und Aktoren verfügbar.

Abbildung 2.7: Zentrale Steuerung für eine HomeMatic Installation.

Geräte werden über das proprietäre Funkprotokoll *Bidirectional Communication System* BidCoS im 868 MHz-Band über FSK-Modulation miteinander verbunden. Auch eine Kommunikation mit eigenen Schaltschrankmodulen über den RS485-Feldbus wird über ein IP-Gateway („Zentrale“, „CCU“) unterstützt. Geräte werden auf zwei Arten eingebunden:

Zum einen kann zwischen bestimmten Aktoren und Sensoren eine direkte, statische Verbindung eingerichtet werden (z.B. *Taster* mit *Lampe*). Die Unterstützung dieser Verbindung ist begrenzt, da jede Interaktion vorab in der Firmware beider Geräte verankert werden muß.

Zum anderen können alle Geräte an ein Gateway („CCU“, Abbildung 2.7) angebunden werden, das eine Ethernet-Schnittstelle bereithält und umfangreichere Verknüpfungen ermöglicht. Es bietet umfangreiche Konfigurations- und Steuermöglichkeiten für die gesamte Installation:

- *Browserbasierte Benutzerschnittstelle.* Die CCU bietet eine Webbrowser-basierte Oberfläche für die Steuerung und Konfiguration der Installation. Eine Interaktion mit dem zugrundeliegenden Betriebssystem ist nicht nötig, aber möglich. So kann das System auch von technisch versierten Personen und Laien gleichermaßen in Betrieb genommen werden.

- *Überwachen angeschlossener Geräte.* Neben Batterieladeständen von entsprechenden Geräten läßt sich auch die Kommunikation mit solchen zur Fehlersuche einsehen.

- *Eigene Steuerprogramme.* Ein Modul ermöglicht das Einrichten von eigenen kleinen Programmen zur Steuerung des Smart Home. Alle Sensoren und Aktoren stehen darin zur Verfügung und können durch einfache Baukasten-Logik oder Schreiben von eigenem Code miteinander verknüpft werden.

- *XMLRPC-Schnittstelle.* Zur unabhängigen Programmierung bietet die Zentrale eine Schnittstelle über XMLRPC. Damit können auf ein externes Kommando hin die aktuellen Zustände von Geräten ausgelesen und neu geschrieben werden. Außerdem kann die Zentrale angewiesen werden, bei Änderung eines Wertes ihrerseits eine XMLRPC-Anfrage an eine bestimmte Unified Resource Locator (URL) abzusetzen, welche das betroffene Gerät und den neuen Wert beinhaltet.

- *Erhöhen der Antennenreichweite durch Zusatzmodule.* Über Ethernet/IP werden kleinere Einheiten („IP-Konfigurationsadapter“) im Heimnetz an die Zentrale angebunden und räumlich passend platziert. Geräte können nun an den Adapter mit der größten Signalstärke angekoppelt werden. Sind mehrere Adapter in einem Netz vorhanden, handeln sie die Zuordnung automatisch aus, indem die Signalstärke an jedem Gerät gemessen wird.

Das Funkprotokoll und die physikalische Spezifikation sind nicht offengelegt. Mit Hilfe von Reverse-Engineering wurden die Eckdaten und die Codierung von Basisfunktionen von Geräten ermittelt und stehen im Projekt Freundliche Hausautomatisierung und Energie-Messung (FHEM) [23] der Allgemeinheit zur Verfügung.

Der Hersteller suggeriert zwar eine vorhandene IT-Sicherheit, es wird aber laut [24] nicht verschlüsselt. Stattdessen werden Nachrichten nur signiert, womit sich folgende Eigenschaften ergeben:

- *Langsame Reaktion.* Es müssen drei Nachrichten ausgetauscht werden, von denen jede bis zu 10 mal übertragen wird im Fehlerfalle. Ein Gerät reagiert dann merklich langsamer.
- *Keine Verschlüsselung.* Kommando und Bestätigung werden im Klartext gesendet. Ein Angreifer kennt die jeweiligen Zustände.
- *Fehler in der Implementierung.* Manche Befehle werden ausgeführt, bevor die signierte Bestätigung eingetroffen ist.
- *Klonen der Zentrale.* Ist die ID der Zentrale bekannt, kann diese über den Ethernet-Adapter geklont werden. Eine Installation, die das Standardpaßwort verwendet, ist damit kompromittiert.

Wird die Zentrale nicht verwendet, kann auch eine universelle Kommunikationshardware, z.B. der Firma sh-Elektronik GmbH [25] basierend auf dem System-on-Chip (SoC) CC1101 von Texas Instruments, verwendet werden. Das Modul ist in Abbildung 2.8 dargestellt. In Kombination mit FHEM lassen sich damit HomeMatic-Geräte steuern und auslesen. Bislang (Juli 2013) wird das Projekt vom Hersteller geduldet.

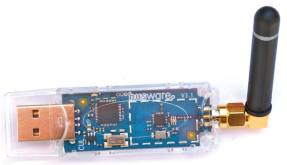


Abbildung 2.8: Funkmodul mit USB-Anschluß

2.1.2.5 KNX-Standard

KNX, die Abkürzung von Konnex, bezeichnet einen nicht-offenen Standard zur hauptsächlich drahtgebundenen Steuerung elektrischer Verbraucher (Sensoren, Aktoren) in Gebäuden [26], der von der KNX-Association spezifiziert wurde. Der Standard ging aus dem European Installation Bus (EIB) hervor und ist zu diesem weitgehend kompatibel bezüglich des Nachrichtenformats und des physikalischen Nachrichtentransports.

Geräte werden über unterschiedliche Techniken in ein Gebäudenetz integriert:

- *KNX-TP1.* Verdrillte Zweidrahtleitung zu jedem Gerät (Bustopologie).

- *KNX-PL110.* Übertragung über die Stromleitung (Baumtopologie) [27].
- *KNX-RF.* Drahtlose Kommunikation [28]; weniger verbreitet.
- *KNX-IP.* Ethernet- und IP-basierte Übertragung, die für die Vernetzung weit auseinanderliegender Gebäudeteile vorgesehen ist.

Die Übertragung von Befehlen findet in Telegrammen statt [29]. Durch die Broadcast-Eigenschaften des Mediums erreicht ein Telegramm alle angeschlossenen Teilnehmer eines Segments. Jedes Unicast-Telegramm wird vom Empfänger bestätigt, bei Gruppentelegrammen sendet jeder Teilnehmer eine Bestätigung. Durch elektrische Eigenschaften ist sichergestellt, daß auch bei mehreren Teilnehmern eine negative Bestätigung stets Vorrang erhält und der Absender das Gruppentelegramm erneut senden kann. In diesem Falle wird es als Wiederholung gekennzeichnet, damit Geräte, die korrekt empfangen hatten, den Befehl nicht erneut ausführen. Der Aufbau eines typischen Telegramms der Twisted-Pair-Verkabelung ist in Abbildung 2.10 dargestellt.

Bei der Adressierung von Zielgeräten wird zwischen Uni- und Multicast („Gruppen“) unterschieden. Daher ist die Zieladresse 17 bit lang und codiert im letzten Bit die Adressierungsform. Eine `0` entspricht einem Unicast-Ziel, eine `1` analog einer Gruppenadresse. Das erste Bit von Gruppenadressen hat stets den Wert `0`. In Abbildung 2.10 ist eine zweistufige Gruppenadressierung dargestellt; es existiert auch ein Betriebsmodus mit einer dreistufigen Hierarchie. Ein Gerät kann einer Anzahl an Gruppen zugeordnet werden.

Segmentkoppler leiten Telegramme in beide Richtungen weiter. Damit sich bei großen Installationen Telegramme nur begrenzt lokal ausbreiten, existiert ein Zähler, der vom Sender auf den Wert 6 gesetzt wird. Jeder Koppler verringert den Zähler um 1 und leitet das Telegramm weiter, es sei denn, der Wert hat bereits Null erreicht. So breitet sich eine Nachricht maximal über 6 Hops aus. Für wichtige Nachrichten (z.B. Alarmmeldungen) wird der Wert vom Sender auf 7 gesetzt. Dadurch werden Nachrichten bedingungslos von allen Kopplern weitergeleitet und der Zähler nicht dekrementiert. Die Nachricht verbreitet sich in der gesamten Installation.

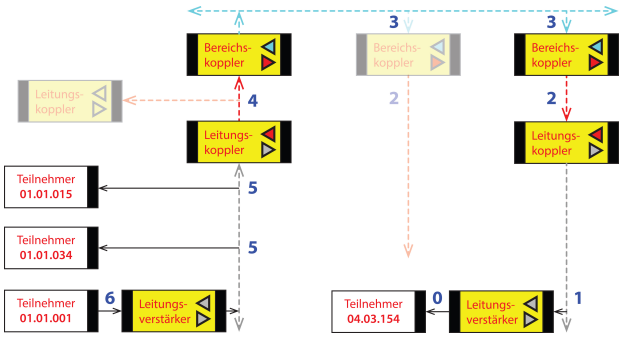


Abbildung 2.9: Demonstration des Dekrementierens des Routing Counters (blaue Ziffern).

KNX kann im regulären Telegramm 1 bis 16 Byte Nutzdaten an Empfänger übertragen. Die Länge des Datenfeldes ist in 4 Bit codiert, wobei der Wert `0` genau einem übertragenen Nutz-Byte entspricht.

Die Bestätigung von Telegrammen erfolgt durch ein einzelnes Byte, das einen von drei Werten annehmen kann. Für einen nicht korrekten Empfang wird `0b00001100` gesendet, für einen korrekten Empfang `0b11001100`. Da aufgrund der gewählten Übertragungstechnik `0`-Bit dominieren, führt eine einzige Nicht-Bestätigung insgesamt zu einem NACK.

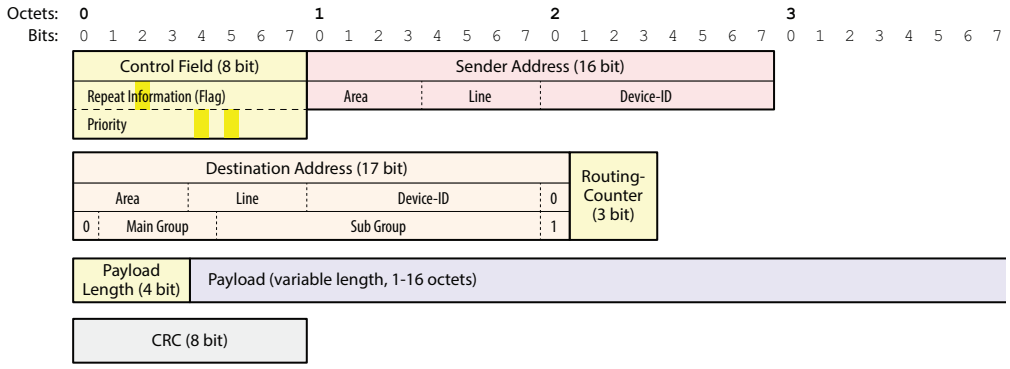


Abbildung 2.10: Schematischer Aufbau eines Telegramms im KNX-Protokoll. Abgebildet ist das Telegramm für Twisted-Pair-Verkabelung. In anderen Übertragungstechniken müssen unter Umständen mehr Informationen, z.B. zur Synchronisation, gesendet werden. Es ist keine Aussage über Datenraten getroffen. Andere Telegrammformate, die größere Nutzdatenmengen zulassen (KNX TP1-256) , werden im Rahmen der Arbeit nicht behandelt.

Ist der Empfänger beschäftigt, so ist die Bitfolge `0b11000000` zu senden. Auch in diesem Falle findet bei bestimmten Nutzdaten (Schalterdruck) eine erneute Übertragung statt.

Neben dem Nachrichtenformat spezifiziert KNX eine Reihe von Nachrichteninhalten für die Kommunikation von Geräten [30] (nicht öffentlich). Numerische Größen werden dabei in einem einzigen Telegramm versendet. Für freie Texte zur Darstellung auf Anzeigen oder ähnlich können mehrere konsekutive Telegramme verwendet werden. Gemeinsam mit vorigen Produkten ist KNX, daß die meisten Meta-Informationen (Datentypen, gültige Wertebereiche, Einheiten, etc.) über Codetabellen abstrahiert werden.

Der Hersteller Miele bietet mit dem Produktkonzept „Miele@home“ [31] eine Powerline-basierte Lösung zur Nachrüstung in seine Geräte an. Es liegen keine Informationen zur Zahl der Installationen oder zu Erfahrungswerten vor. Mit einem Preis von ~340 € [32] alleine für das notwendige Ethernet-Gateway ist die Zielgruppe der Endverbraucher allerdings deutlich eingeschränkt.

Von allen vorgestellten Produkten böte der Standard KNX durch seine Abstraktionsfähigkeit von Informationen (einige in Tabelle 1.1 gezeigt) auf technischer Seite die Möglichkeit, Haushaltsgeräte zu modellieren. Dem Einsatz widersprechen die folgenden Punkte:

- Vergleicht man reine Funktionen, so sind die vom KNX-Verbund zertifizierten Geräte sind mit Abstand die teuersten. Ein großflächiger Einsatz kommt insbesondere im Low-Cost-Segment nicht in Betracht, siehe auch vorige Ausführungen zu „Miele@home“.

- Jede Beschreibung von Geräten muß im Standard verankert werden. Dynamische Beschreibungen sind nicht vorgesehen.

Produkt	Medium	Frequenz in MHz	Code	maximale Größe		Datenrate in kbit s^{-1}	ACK
				Frame	Payload		
Z-Wave	Funk	*versch.*	2-FSK	*k.I.*	~230 B	9,6, 40	ja
ZigBee	Funk	868, 2400	2-PSK	127 B	104 B	20, 40, 250	ja
EnOcean	Funk	315, 868	OOK	16 B	4 B	125	optional
HomeMatic	Funk, RS485	868	2-FSK	*k.I.*	*k.I.*	20	ja
KNX-TP	2-Draht	*k.I.*	OOK	253 bit	16 B	9,6	ja
KNX-PL	Powerline	0,095-0,125	2-FSK	212 bit	16 B	1,2	ja
KNX-RF	Funk	868	FSK	36 B	8 B	32,768	optional
KNX-IP	100 Base-TX	31,25	4B5B	1522 B	1480 B	100 000	ja

Tabelle 2.2: Übersicht über die technischen Merkmale der in diesem Kapitel vorgestellten Produkte, die für eine Heimsteuerung geeignet sind.

2.1.2.6 Gegenüberstellung der übertragungstechnischen Merkmale

Die wichtigen technischen Eigenschaften der vorgestellten Systeme werden in Tabelle 2.2 gegenübergestellt. Werden lediglich die OSI-Schichten PHY, Data Link und Vermittlung der vorgestellten Produkte betrachtet, so sind alle vorgestellten Implementierungen potentiell für das Smart Home geeignet. Sie versenden einen bestimmten Inhalt über ein gemeinsames (Funk-)Medium an alle Geräte in Reichweite und bieten meist zusätzliche Merkmale wie bestätigte Übermittlung oder Adressierung von Zielgeräten.

Weiterhin werden unterschiedliche Netzwerktopologien unterstützt, wenn die geographische Ausdehnung der Installation nicht mehr nur sternförmig abgedeckt werden kann.

Auf Anwendungsebene können alle Produkte ihre „eigenen“ Geräte verwalten und bedienen. Eine Kompatibilität zwischen den Produkten besteht allerdings nicht und muß ggf. auf Anwendungsebene über externe Software realisiert werden.

2.1.3 Nachteile und Schwierigkeiten

Allen beschriebenen Produkten und Protokollen sind bestimmte Schwachpunkte sowohl im Konzept wie auch in ihren jeweiligen Implementierungen gemein, wenn man sie vor dem Hintergrund der Erwartungen und Anforderungen aller drei Interessensgruppen des ersten Kapitels betrachtet. Die Auffälligkeiten werden im Folgenden diskutiert.

2.1.3.1 Gateway als Single-Point-of-Failure

In Lösungen, in denen das Gateway die Kommunikation mit anderen Netzen und Protokollen übernimmt, stellt es gleichzeitig einen Single-Point-of-Failure dar. Ärgerlich ist das insbesondere, wenn eine Programmlogik nicht mehr mit Aktoren und Sensoren kommunizieren kann und das Smart Home stillsteht.

Zwar bieten die Lösungen die Möglichkeit, Geräte direkt miteinander zu verknüpfen, allerdings funktioniert das nur, wenn das Nachrichtentransportprotokoll zum einen zwischen allen beteiligten Geräten kompatibel ist und zum anderen das Transportmedium dezentral von jedem Teilnehmer aus selbst verwaltet wird (gemeinsames Medium).

Ist das Gateway zugleich Koordinator und fällt aus, funktioniert noch nicht einmal die Kommunikation zwischen Geräten.

2.1.3.2 Inkompatibilität zwischen Herstellern

Da die vorgestellten Lösungen untereinander nicht kompatibel sind, lassen sich Smart Devices nicht zwischen ihnen austauschen. Es besteht eine Abhängigkeit von den jeweiligen Herstellern.

Sollen innerhalb einer Installation Geräte dennoch gemeinsam von unterschiedlichen Produkten eingesetzt werden, etwa weil eine gewünschte Funktionalität in einem bestimmten Produkt nicht verfügbar ist, so ist in der Regel für jedes Produkt ein IP-Gateway zu verwenden, das Nachrichten zu einer zentralen Anwendungslogik sendet (z.B. FHEM), die dann die Nachrichteninhalte zwischen den Produkten übersetzt. Damit ergeben sich weitere Schwierigkeiten:

- Weitere Hardware muß mit zusätzlichem Aufwand angeschafft und betrieben werden.
- Die eingesetzte Logikschicht muß auch zukünftig gewartet werden.
- Es ist eine gemeinsame, verständliche Oberfläche bereitzustellen.

2.1.3.3 Duplizierte Hardware bei Herstellern

Ein bestimmtes Smart Device bietet in der Regel ein überschaubares Set an Funktionen. Da nun Hersteller ihre eigenen Transportsysteme implementieren, müssen sie funktional vergleichbare Smart Devices selbst entwickeln, testen, produzieren und verkaufen. Eine derartige Ver(sch)wendung von Ressourcen schlägt sich zum einen im Preis nieder und schadet zum anderen dem Gesamtkonzept des Smart Homes. Entschließt sich ein Hersteller gegen das Produzieren eines bestimmten Smart Devices, besteht für den Endbenutzer in der Regel keine Möglichkeit, die Funktionalität zu realisieren.

2.1.3.4 Keine offenen Protokolle und Standards

Keines der vorgestellten Protokolle bietet einen offenen Standard. Entweder sind Spezifikationen (offiziell) gar nicht verfügbar oder sie müssen kostenpflichtig von der jeweiligen Allianz bzw. dem Hersteller erworben werden. Häufige Argumente sind, die technische Unterstützung zu vereinfachen und dem Endkunden integrierte, funktionierende Produkte und Lösungen verkaufen zu können. Allerdings folgt auch daraus eine Teuerung, da sich Hersteller die Mitgliedschaft in den jeweiligen Allianzen durch den Endkunden über den Kaufpreis der Geräte bezahlen lassen.

2.1.3.5 Nur geringe Nutzdatenkapazität

Den Produkten ist gemeinsam, daß die Kapazität für Nutzdaten, auch ersichtlich aus Tabelle 2.2, auf wenige Byte begrenzt ist. Für die jeweiligen Anwendungsschichtprotokolle der Lösungen hat dies keine Konsequenz, diese sind auf den Nachrichtentransport hin zugeschnitten. Allerdings bedeutet das auch eine enge Verknüpfung der Schichten, wodurch ein Austausch eben jener Anwendungsschicht erschwert wird.

Alles in allem führt insbesondere diese mangelnde Flexibilität dazu, daß bislang nur vergleichbare und ähnlich klassifizierbare Smart Devices großflächig verfügbar sind. Für die Arbeit gilt es daher zu zeigen, wie Abstraktion möglichst weit oben im Schichtenmodell anzusiedeln ist und eine Unabhängigkeit von der Nachrichtentransportlösung erreicht werden kann.

2.1.3.6 Begrenzte Erweiterbarkeit der Protokolle

Die subjektiv größte Schwachstelle der vorgestellten Produkte ist allerdings die statische Definition der Anwendungsschichtprotokolle der vorgestellten Lösungen. Sie bieten nur minimale Abstraktion und codieren Adressen, Zustände, Befehle und Argumente in Bytes oder Teilen von Bytes.

Insbesondere vor dem Hintergrund, daß sich weiße Ware nur schwer in statischen Klassen abbilden läßt („Kühlschrank mit RGB-LED-Innenraumbeleuchtung“), erscheinen die folgenden Argumente nennenswert:

- Es existiert in den vorgestellten Lösungen keine Möglichkeit, Geräte vollständig dynamisch zu beschreiben. Informationen über ihre Eigenschaften und Dienste sind statisch in Protokoll und Implementierung der Firmware verankert. In der Regel können neue Funktionen nur ergänzt werden, wenn alle Beteiligten über die Neuerung, z.B. durch Spezifikations- und Firmwareupdates, informiert werden. Das betrifft insbesondere das Gateway, wenn neue Geräte auf den Markt kommen.
- Selbst wenn ein Protokoll mit „benutzerdefinierten“ Feldern beliebige Nutzdaten übertragen kann, so besteht noch immer kein Konsens zwischen Herstellern über deren Inhalt.

2.2 Standardisierte Anwendungsprotokolle

Sofern eine Schichtentrennung umgesetzt werden kann und Protokolle auf Anwendungsebene nicht mehr im Rahmen von *cross-layer optimization* zur Reduktion des Energieverbrauchs (z.B. für batteriebetriebene Smart Devices) in ihrer Nachrichtenlänge eingeschränkt werden müssen, lohnt ein Vergleich zwischen Protokollen der Anwendungsschicht.

Insbesondere interessiert, ob die Vorteile einer Betrachtung rein auf Anwendungsebene (z.B. Abstraktion von Geräteeigenschaften) eventuelle Nachteile hinsichtlich des Ressourcenbedarfs aufwiegen.

Die folgenden Ausführungen erfolgen vor dem Hintergrund des Smart Homes und der Domäne der Haushaltsgeräte. Anderen Domänen stehen in diesem Kapitel nicht im Fokus.

2.2.1 Abstrakte Betrachtung eines Anwendungsprotokolls

Ein Ausweg aus den in Kapitel 2.1.2 herausgearbeiteten Schwierigkeiten besteht darin, Abstraktion direkt in den Smart Devices selbst zu implementieren und neben den Nutzdaten (Befehle, Parameter, Zustandsmeldungen, etc.) auch ihre Bedeutung (z.B. Textbeschreibung, Einheiten, Wertebereiche, etc.) im Protokoll zu verankern. Parameter der Nachrichtenübermittlung und andere Implementierungsdetails rücken dann in den Hintergrund und können herstellerunabhängig realisiert werden.

Sofern ein universelles Protokoll rein auf Anwendungsebene realisiert werden kann, werden damit alle untergeordneten Schichten austauschbar. Es spielt keine Rolle mehr, über *welche Art* der Kommunikation ein Smart Device mit dem Heimnetz verbunden ist. Alle beteiligten Schichten müssen nur sicherstellen, *daß* eine Information eines Clients die Geräte erreicht. Eine Vielzahl unterschiedlicher Implementierungen der OSI-Schichten 1 mit 4 wird so zukünftig im Smart Home unterstützt.

2.2.1.1 Aufgaben

Zusammenfassend wird vorausgesetzt, daß die Geräte physikalisch über ein geeignetes, zuverlässiges Broadcast-Medium in Verbindung stehen. Zur Unterscheidung besitzt jedes Smart Device eine eigene und eineindeutige Adresse. Sie ist mit dem Hostnamen innerhalb von Rechnern vergleichbar und unabhängig von anderen Schichten.

Da über die untergeordneten Schichten ansonsten keine Aussage getroffen wird, bietet das Anwendungsschichtprotokoll folgende Funktionen:

- *Auffinden von Geräten im Netzwerk.* Durch bestimmte Mechanismen und Befehle sollen alle Geräte innerhalb eines logischen Segments dem anfragenden Client ihre Existenz bekanntgeben. Dieser Prozeß wird als „Device Discovery“ bezeichnet. Dabei sollte jedes Gerät zunächst in diesem ersten Schritt nur seine eigene Identifikation übermitteln, damit der Client eine Liste der bekannten Geräte aufbauen kann zur späteren weiteren Verarbeitung.
- *Ermitteln der Eigenschaften eines Geräts und seiner Dienste.* Für jede gefundene ID oder Adresse soll der Client über das Protokoll die Merkmale, Eigenschaften und Funktionen eines Smart Devices ermitteln können. Dies kann separat oder direkt mit dem ersten Schritt geschehen. Besonderer Wert ist dabei auf Anforderungen von Menschen und Maschinen zu legen: während Menschen semantische Informationen

zur Interpretation benötigen, müssen Maschinen binäre, eineindeutige Bezeichner für jede Funktion erhalten. Das Anwendungsprotokoll unterstützt möglichst beide Zielgruppen mit einem einzigen Nachrichtentyp. Dieser Prozeß wird als „Service Discovery“ bezeichnet.

- *Kommandieren eines oder mehrerer Geräte.* Mit geeigneten Nachrichten werden Befehle an Smart Devices abgesetzt. Unterstützt wird mindestens das Auslesen von Zuständen sowie auch das Schreiben von neuen Daten. Empfangene Befehle können optional quittiert werden. Dieser Prozeß wird als „Service Consumption“ bezeichnet.

- *Verbreiten von Ereignissen.* Eng mit dem vorigen Punkt ist die Aufgabe verbunden, bestimmte Maschinenzustände auf Entscheidung des Smart Devices selbst und nicht auf Anfrage eines Clients hin im Segment zu verteilen („Ereignisnachricht“). Optional können Smart Devices direkt auf solche Ereignisnachrichten reagieren („Verknüpfung“).

Nicht Aufgabe des Anwendungsprotokolls hingegen ist das Bereitstellen oder Beschreiben einer Anwendungslogik, mit welcher Smart Devices miteinander verknüpft werden. Auch das Zwischenspeichern von Anfragen und Ergebnissen oder das Führen eines Verzeichnisses aller bekannten Smart Devices zur schnelleren Abfrage durch Clients, werden anderweitig realisiert.

2.2.1.2 Wünschenswerte Eigenschaften

Nachdem die Aufgaben festgelegt wurden, werden auch die wünschenswerten Eigenschaften des Protokolls genannt:

- *Unterstützung von Broadcast-Kommunikation.* Das Medium wird zunächst nur als Broadcast-Medium angesehen. Daher muß das Anwendungsprotokoll eine fehlende Adressierung dort ausgleichen und unterstützt den Broadcast-Transport durch eigene Adressen und Geräte-Identifikatoren.

- *Geringe Nachrichtengröße.* Für einen minimalen Energieverbrauch ist die Nachrichtengröße so klein als möglich zu wählen und Redundanz darin möglichst zu vermeiden.

- *Effiziente Implementierung.* Auch ist zu beachten, daß das Protokoll auf kleinen Micro Controller Units implementiert werden wird. Es muß daher möglichst wenig Speicherplatz in Random-Access Memory (RAM) und Read-Only Memory (ROM) belegen und soll so wenig wie möglich Rechenzyklen bei seiner Verarbeitung („Parsen“) benötigen.

- *Verarbeitung als Datenstrom.* Aus vorigem Punkt läßt sich ableiten, daß Nachrichten als Datenstrom verarbeitet werden und nicht erst im Ganzen vorliegen müssen.

2.2.1.3 Vorteile bei Lösung auf Anwendungsebene

Inwieweit sich die Verschränkung zwischen Anwendungsprotokollen und untergeordneten Schichten (z.B. IP und TCP/UDP) tatsächlich lösen läßt, ist zu untersuchen. Da die Abstraktion bei dieser Umsetzung direkt im Smart Device stattfindet, vereinfachen sich auch die Aufgaben des Medien-Gateways, eingeführt in Abbildung 1.3. Die neue Form des Protokollstacks ist in Abbildung 2.11 dargestellt.

Damit werden die Vorteile einer Lösung rein auf Anwendungsebene angegeben:

- *Herstellerübergreifende Kommunikation zwischen Smart Devices.* Unabhängig von der Wahl der Transportmedien und -protokolle können Geräte aller Hersteller, die die Anwendungsschicht implementieren, miteinander kommunizieren. Für Endkunden bedeutet es eine größere Auswahl an Geräten mit ähnlichen oder gleichen Funktionen.
- *Lose Koppelung zwischen Smart Devices und Clients.* Wenn jedes Gerät seine Funktionsbeschreibung selbst anbietet, kann eine Koppelung zwischen Client und Server lose eingerichtet werden. Geräte mit unterschiedlichen Soft- und Hardwareversionen können im gemeinsamen Netzwerksegment koexistieren. Updates und neue Funktionen können von anderen Teilnehmern unabhängig durchgeführt werden. Gateways lassen sich austauschen und neue Geräte einfach in das Netzsegment integrieren.
- *Freie Wahl der Nachrichtentransportlösung.* Gerade in preiswerten Marktsegmenten für Smart Devices sind Kommunikationsmodule derzeitiger Standards und Protokolle (Ethernet, Wireless Local Area Network (WiFi), Bluetooth, etc.) wegen ihrer notwendigen Konfiguration und Protokollkomplexität nicht attraktiv. Bessere ist, wenn Hersteller lediglich ein Gerät mit Schnittstelle zum Heimnetz stellen und die (möglichst zuverlässige) Übertragung von und zu ihren Geräten selbst, ggf. proprietär, implementieren. Dabei können sie auf Kommunikationsmodule setzen, die ihren Anforderungen (Preis, Zulassung, Stromverbrauch, etc.) am besten gerecht werden.

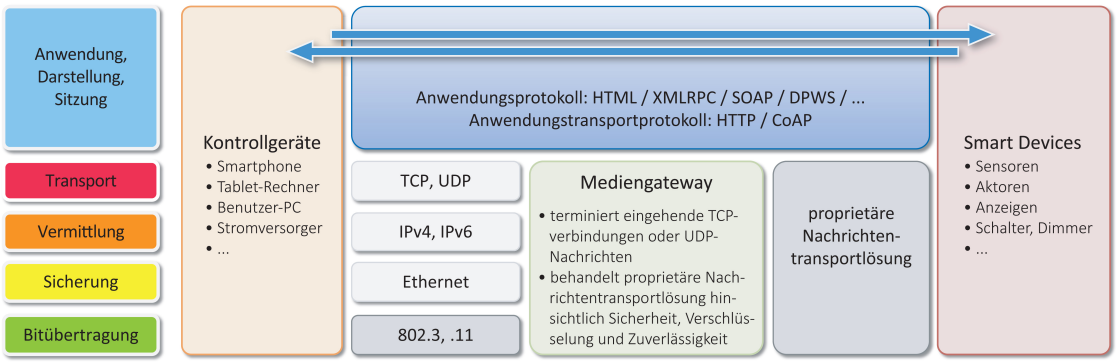


Abbildung 2.11: Vereinfachtes Zwei-Schichten-Modell der Kommunikation zwischen Smart Devices und anderen Heimnetzgeräten, wenn gemeinsame Protokolle für Transport und Nachrichtenformat auf Anwendungsebene verwendet werden. Zwischengeschaltete Komponenten wie Gateways oder Filter können einfacher realisiert werden.

- *Nur Broadcast-Kommunikation im Nachrichtentransport erforderlich.* Die Implementierung der unteren OSI-Schichten kann einfach gehalten werden. Insbesondere Adressierung und Routing sind zumindest bei der Annahme eines einzigen Hops in der Verbindung nicht erforderlich.

2.2.2 Hypertext Transport Protocol (HTTP) und Derivate

Eine der einfachsten Formen, abstrakt auf Anwendungsebene zu kommunizieren, stellt das Hypertext Transport Protocol (HTTP) dar [33]. Es definiert zwei Nachrichtentypen, *Anfrage* und *Antwort*. Im Wesentlichen besteht eine Nachricht aus vier Teilen:

- *Methode* bzw. Verb. Bezeichnet die Operation, die ausgeführt werden soll.
- *URL-Pfad.* Gibt an, auf welche Ressource auf dem Server die Methode anzuwenden ist. Entfällt in der Antwort.
- *Header.* Enthält über Schlüssel/Werte-Paare Zusatzinformationen („Metadaten"), die die jeweilige Anfrage oder Antwort betreffen.
- *Nutzdaten.* Enthält – optional – die Nutzdaten der Anfrage.

Codierung und Repräsentation einer Ressource werden im Protokoll angegeben. Interpretation und Darstellung bleiben allerdings der Anwendung überlassen.

Eine feste Zuordnung von Nachrichtenaufbau bzw. Nachrichtenbedeutung zu ihrem Inhalt ist nicht definiert. Das folgende Beispiel, bei dem eine Temperatur ausgelesen wird, verdeutlicht den Sachverhalt. Die nächsten beiden Statements sind nach HTTP äquivalent.

```
1 >>> GET /temperature HTTP/1.1

2 >>> POST /getTemperatureService HTTP/1.1
```

Listing 2.1: Äquivalente Beispiele einer HTTP-Anfrage zum Abfragen eines Elements.

Gleiches gilt bei Änderung von Ressourcen:

```
1 >>> DELETE /meinedatei.txt HTTP/1.1

2 >>> GET /deleteService?file=meinedatei.txt HTTP/1.1

3 >>> POST /deleteService HTTP/1.1
4 >>>
5 >>> file=meinedatei.txt
```

Listing 2.2: Äquivalente Beispiele einer HTTP-Anfrage zum Löschen eines Elements.

HTTP und seine Erweiterungen, z.B. WebDAV [34], definieren bestimmte Intentionen zu gegebenen Operationen. Die Operation `DELETE` etwa soll die bezeichnete Ressource tatsächlich entfernen, in Falle einer Datei eben diese aus dem Dateisystem löschen. Ob der Dienst das aber auch so umsetzt, bleibt offen; es handelt sich lediglich um eine seitens

der Spezifikation empfohlene Konvention. Daher ist für Menschen weitere Information („Dokumentation“) erforderlich.

Da HTTP eine bereits bestehende Unicast-Verbindung zwischen den beiden Teilnehmern voraussetzt, existiert kein Mechanismus zum Auffinden von Geräten in einem Netzwerk.

2.2.2.1 Service-Discovery

HTTP definiert keine Möglichkeit des Erkennens von Diensten. Im für Menschen geschaffenen World Wide Web (WWW) realisiert eine Umleitung zu einer Standardressource diese Funktion: die vom Browser initial gestellte Anfrage `GET /` kann in der Regel nicht direkt auf einen Inhalt im Dateisystem abgebildet werden. Daher wird der Browser angewiesen, stattdessen in einer weiteren Anfrage eine reale Datei (z.B. `/index.html` oder `/index.php`) als angefragte Ressource zu benennen.

Die Standardressource enthält ein von Menschen interpretierbares Verzeichnis („landing page“) der weiteren Dienste des Servers, oft über die Markup-Sprache Hypertext Markup Language (HTML) formatiert. Sie werden in konsekutiven `GET`-Anfragen abgerufen.

Der Transport von HTTP ist nur über TCP üblich, daher können Broadcast- oder Multicast-Nachrichten nicht verwendet werden.

2.2.2.2 Service-Description

HTTP definiert keine Beschreibung von Services. Der Designer einer Anwendung kann frei entscheiden, welche Operation auf welcher Ressource welche Manipulation eines Datenspeichers des Rechners nach sich zieht.

Der Begriff *Datenspeicher* sei sehr allgemein verwendet und bezeichnet neben einer Festplatte auch flüchtige Speicher wie Hauptspeicher oder Displayspeicher eines Rechners.

2.2.2.3 Service-Consumption

Der Aufruf von Diensten ist einfach auszuführen: es werden minimal nur die Elemente *Verb* und *URL* für eine Anfrage benötigt. Der Server antwortet mit HTTP-Statuscodes und gegebenenfalls mit einem eigenen Nutzdatenteil, dargestellt in Listing 2.3. Auch Fehler lassen sich abbilden, gezeigt in Listing 2.4.

```
1 >>> DELETE /meinedatei.txt HTTP/1.1

2 <<< HTTP/1.1 200 OK
3 <<< Content-Length: 54
4 <<<
5 <<< <html><body><p>Datei wurde entfernt.</p></body></html>
```

Listing 2.3: Beispielhafte HTTP-Anfrage und Antwort mit Übertragung von Nutzdaten.

```
>>> DELETE /meinedatei.txt HTTP/1.1

<<< HTTP/1.1 404 Not Found
<<< Content-Length: 54
<<<
<<< <html><body><p>Datei nicht gefunden.</p></body></html>
```

Listing 2.4: Beispielhafte Fehlermeldung auf eine Anfrage zum Löschen eines Elements.

2.2.2.4 Implementierungen

Da HTTP ein offenes und einfaches Protokoll darstellt, existieren viele unterschiedliche Implementierungen für die meisten gängigen Programmiersprachen. Ein wesentlicher Vorteil ist die festgelegte Reihenfolge von Elementen im Protokoll, wodurch ein Parser für Anfragen mittels eines einfachen Zustandsautomaten mit vier hauptsächlichen Schritten realisiert werden kann [35]:

1. *HTTP-Methode.* Ermittelt die auszuführende Methode und speichert sie als binär codierte Zahl für einen spätere Auswahl. Wenn die gewünschte Methode nicht unterstützt wird, kann bereits hier mit einer entsprechenden Fehlermeldung (`405 Method not allowed`) abgebrochen werden.

2. *URL-Pfad.* In diesem Schritt wird die übergebene Ressource auf ein Element in der Maschinensteuerung oder dem Programm abgebildet. Unter gängigen Webservern ist das häufig eine Datei im lokalen Dateisystem; bei eingebetteten Systemen kann einer textuellen URL aber auch ein anderes Element, z.B. der Inhalt einer Speicherzelle, zugeordnet sein. Ist die URL zu lang für ein verarbeitendes System, kann an dieser Stelle mit einer entsprechenden Fehlermeldung (`414 Request URI too long`) abgebrochen werden.

3. *Header-Elemente (optional).* Im dritten Schritt werden eventuelle Schlüssel/Werte-Paare ausgelesen; das Trennzeichen ist der Doppelpunkt. Hilfreich sind solche Informationen bei der Klassifizierung von Inhalten oder um der Gegenseite eigene Fähigkeiten mitzuteilen.

4. *Nutzdatenteil (optional).* Im vierten Schritt werden optionale Nutzdaten ausgewertet. Nachdem HTTP keine besondere Codierung und Formatierung oder gar Inhalte dieses Teils vorschreibt, muß in diesem Schritt der Parser auf die jeweilige Anwendung zugeschnitten sein.

Eine HTTP-Nachricht muß vor Verarbeitung nicht erst vollständig gespeichert werden. Auch eine Byte-weise Verarbeitung ist möglich, was insbesondere bei Geräten mit geringen Ressourcen einen wichtigen Vorteil darstellt.

2.2.3 Extensible Markup Language Remote Procedure Call (XMLRPC)

Das XML-Schema [36, 37] für einen Remote Procedure Call (RPC) wurde 1999 von Dave Winer ins Leben gerufen, um einen damals noch nicht existierenden Textstandard für einen HTTP-basierten RPC zu definieren. Wichtig waren dabei einheitliche Datentypen [37, Kapitel 2], ein wohldefiniertes Extensible Markup Language (XML)-Schema und eine möglichst einfache Zuordnung zu Datentypen anderer Programmiersprachen sowie eine einfache Implementierungsmöglichkeit.

2.2.3.1 Kommunikation und Nachrichtentypen

XMLRPC wird per HTTP zwischen Teilnehmern transportiert. Damit ist auch hier nur eine Unicast-Kommunikation vorgesehen. Mechanismen zum Auffinden von Geräten fehlen.

Angebotene Dienste werden als *Methoden* bezeichnet. Pro Anfrage können mehrere Methoden jeweils mit eigenen Parametern gleichzeitig aufgerufen werden. Die Definition von Methoden und ihren Parametern bzw. Argumenten sowie einer ontologischen Zuordnung ist dem Entwickler überlassen.

Argumente werden in der Reihenfolge ihres Auftretens innerhalb der Anfrage an die Anwendung übergeben, wobei eine Datentypprüfung bereits stattfindet.

Da XMLRPC Nachrichten nach einem statischen Schema aufbaut, läßt sich ein solches sowohl als Document Type Definition (DTD) wie auch als XML Schema Definition (XSD) angeben[38]. Beide Definitionen haben keinen offiziellen Charakter, können aber zur Verifikation einer Nachricht herangezogen werden.

2.2.3.2 Service-Discovery

Implementierung von XMLRPC unterstützen einen Discovery-Mechanismus für verfügbare Methoden (synonym verwendet mit „Dienste“) mit Hilfe der der Funktionen `system.listMethods()` und `system.methodSignature()`. Eine beispielhafte Ausgabe der angegebenen Methode `system.listMethods()` ist in Listing 2.5 abgedruckt. Der Aufruf der so ermittelten Methoden wird in Kapitel 2.2.3.4 beschrieben.

```
<?xml version="1.0"?>
<methodResponse>
  <params> <param> <value> <array> <data>
    <value><string>system.listMethods</string></value>
    <value><string>system.methodSignature</string></value>
    <value><string>example.add</string></value>
  </data> </array> </value> </param> </params>
</methodResponse>
```

Listing 2.5: XMLRPC-Response auf *system.listMethods()*.

2.2.3.3 Service-Description

Mit `system.methodSignature(#Methodenname)` lassen sich die erforderlichen Argumente für eine mit `system.listMethods()` zurückgegebene Funktion ermitteln. Beispielhaft ist die Antwort in Listing 2.6 für `system.listMethods(sample.add)` dargestellt.

```
<?xml version="1.0"?>
<methodResponse>
  <params> <param> <value> <array> <data> <value> <array> <data>
    <!-- Datentyp des Rückgabewertes -->
    <value><string>int</string></value>
    <!-- Datentyp 1. Parameter -->
    <value><string>int</string></value>
    <!-- Datentyp 2. Parameter -->
    <value><string>int</string></value>
  </data> </array> </value> </data> </array> </value> </param> </params>
</methodResponse>
```

Listing 2.6: XMLRPC-Response auf *system.methodSignature(example.add)*.

2.2.3.4 Service-Consumption

Der Aufruf einer XMLRPC-Methode wird stets mit dem HTTP-Verb `POST` eingeleitet. Ein Methodenaufruf besteht dabei aus den zwei Elementen *Methodennamen* und optionalen *Parametern*. Sie werden im Body der HTTP-Nachricht in XML-Kodierung übertragen. Bis auf *array* und *struct* Datentypen hat jedes Argument dabei genau einen Wert.

Eine beispielhafte Anfrage, die einen Dienst auf einem entfernten Rechner aufruft, ist ohne HTTP-Kopfzeilen in Listing 2.7 gezeigt. Bei einzelnen Methodenaufrufen wird der ursprüngliche Methodenname in einer Antwort nicht mehr zurückgegeben, siehe Listing 2.8.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>example.add</methodName>
  <params>
    <param> <value><i4>40</i4></value> </param>
    <param> <value><i4>50</i4></value> </param>
  </params>
</methodCall>
```

Listing 2.7: XMLRPC-Request zur Addition zweier Zahlen.

```
<?xml version="1.0"?>
<methodResponse>
  <params> <param> <value><i4>90</i4></value> </param> </params>
</methodResponse>
```

Listing 2.8: XMLRPC-Response auf die Anfrage in Listing 2.7.

Eine Fehlermeldung besitzt ein definiertes Format und gibt, wie in Listing 2.9 gezeigt, einen numerischen Code sowie einen durch Menschen interpretierbaren Text zurück.

```
<?xml version="1.0"?>
<methodResponse>
  <fault> <value> <struct>
    <member>
      <name>faultCode</name>
      <value><int>4</int></value>
    </member>
    <member>
      <name>faultString</name>
      <value><string>Too many parameters.</string></value>
    </member>
  </struct> </value> </fault>
</methodResponse>
```

Listing 2.9: Aufbau einer XMLRPC-Fehlerantwort.

2.2.3.5 Implementierungen

Auch für XMLRPC existiert eine Reihe von Implementierungen mit vergleichbarem Funktionsumfang. Ein Parsen erfolgt in zwei Schritten:

1. Der erste Schritt umfaßt das Extrahieren der RPC-Nachricht aus dem zugrundeliegenden Transport-Protokoll (z.B. HTTP).
2. Im zweiten Schritt wird dann der XML-Quelltext auf Schlüsselwort von XMLRPC hin ausgewertet und neben Methodennamen auch die jeweiligen Parameter empfangen.

Anzumerken ist, daß das Protokoll, genau wie HTTP, sequentiell aufgebaut ist. Damit können Implementierungen, die auf eine Schema-Verifikation verzichten, mittels einfacherem und ereignisgesteuerten Simple API for XML (SAX)-Parsing [39, Kapitel 20] XMLRPC auch zeichenweise verarbeiten. Ein vollständiges Speichern ist dann nicht notwendig.

2.2.3.6 Anmerkung zur Nachrichtengröße

Zu beachten ist, daß XMLRPC im Gegensatz zu HTTP einen größeren Overhead besitzt. Zur Übertragung eines einzigen Byte, siehe Listing 2.8, werden bei kürzestmöglicher Darstellung (des XML-Nutzdatenteils) ohne Zeilenumbrüche und Tabulatoren 112 Byte übertragen. Das ergibt einen Zuwachs von immerhin 11 100 %! Ein kurzer Vergleich mit HTTP ergibt einen dortigen Zuwachs durch Übertragung von 2 Byte von lediglich 50 %, da die textuelle Repräsentation des Zahlenwertes 90 im Dezimalsystem zwei Stellen in Anspruch nimmt.

Der Overhead erscheint trotz des Vorteils der expliziten Datentypdeklaration bei Parametern nicht gerechtfertigt, da die Anwendung ein übergebenes Datum in jedem Fall gegen bestimmte Beschränkungen prüft.

2.2.4 SOAP

SOAP, inzwischen erweitert zu den WS*-Spezifikationen, definiert ein XML-Schema, mit dessen Hilfe Webservices über XML-basierte standardisierte asynchrone Kommunikation genutzt werden können [40]. Die Anordnung der Elemente des Schemas in einer Nachricht sowie ihre Namen und Attribute sind in einem separat bereitzuhaltenden Webservice Description Language (WSDL)-Schema festzulegen. Ein SOAP-Dienst besteht damit aus zwei Teilen: dem eigentlichen Service und seiner Beschreibung mittels WSDL.

Unterschiedliche Aspekte der Kommunikation im Internet und möglicherweise auch Smart Home werden dabei durch separate Schemata abdeckt:

- *WS-Discovery.* Dient dem Auffinden von Geräten und ihren angebotenen Diensten im Netzwerk. Nutzt dazu Broad- und Multicast-Nachrichten mit bestimmten Inhalten, die von angesprochenen Geräten an den Absender mit der jeweiligen Dienstebeschreibung beantwortet werden.
- *WS-Addressing.* Bietet Adressierungsinformationen eines Dienst auf einem Gerät mit eineindeutigen IDs und Netzwerkadressen untergeordneter OSI-Schichten.
- *WS-Reliable Messaging.* Bietet Methoden und Funktionen, um bestimmte Nachrichten zuverlässig zu übertragen, unter anderem automatisches Neuübertragen und eine Abonnementverwaltung für Dienste-Anbieter und -Konsumenten.
- *WS-Security.* Behandelt Verschlüsselung, Vertraulichkeit und Integrität von einzelnen Nachrichten auf Anwendungsebene. Untergeordnete Schichten sind nicht betroffen; Netzwerkadressen und andere Protokolleigenschaften bleiben sichtbar.

Eine detaillierte Diskussion der WS*-Spezifikationen wird im Rahmen der Arbeit nicht vorgenommen. Es wird nur auf das Nachrichtenformat von SOAP eingegangen, das über eine Reihe von Protokollen (z.B. SMTP/Mail, FTP, etc.) übertragen werden kann.

2.2.4.1 Nachrichtenformat

Eine SOAP-Nachricht besteht aus einem Rahmen („Envelope“), optionalen Kopfdaten („Header“) und dem Inhalt mit XML-formatierten Nutzdaten („Body“). Eine Nachricht selbst macht von der XML-Erweiterung der Namensräume („Namespaces“) [41] Gebrauch, womit jedes Element und jedes Attribut in unterschiedlichen Namensräumen auch unterschiedliche Bedeutung haben kann. Eine beispielhafte Nachricht ist in Listing 2.10 abgebildet.

```
1 <?xml version="1.0"?>
2 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
3   <env:Header>
4     <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
5       <n:priority>1</n:priority>
6       <n:expires>2013-10-25T14:00:00+01:00</n:expires>
7     </n:alertcontrol>
8   </env:Header>
```

```
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm!</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

Listing 2.10: Aufbau einer SOAP-Nachricht. Die verwendeten Attribute müssen in den angegebenen Namensräumen definiert werden, bevor sie im Nachrichtentext verwendet werden können. Außerdem muß ein Client Zugriff auf die Definition der Namensräume haben, um die Attribute hinsichtlich ihrer Datentypen und Beschränkungen zu validieren.

SOAP bietet standardisierte Fehlermeldungen, die stets im Body der Nachricht übermittelt werden. Eine beispielhafte Fehlermeldung ist in Listing 2.11 dargestellt.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <env:Body>
    <env:Fault>
      <env:Code>
        <env:Value>env:Sender</env:Value>
      </env:Code>
      <env:Reason>
        <env:Text xml:lang="de">Server-Timeout</env:Text>
      </env:Reason>
      <env:Detail>
        <env:Text xml:lang="de">
        Server brauchte zu lange zum Antworten.
        </env:Text>
      </env:Detail>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Listing 2.11: Aufbau einer SOAP-Fehlermeldung. Die Elemente `Code` und `Reason` sind erforderlich, `Detail` ist optional. Alle Werte der genannten Elemente müssen wiederum im Namensraum definiert sein. Wenn die im Standard vorgesehenen Elemente nicht ausreichen, ist ein eigenes Schema mit den benötigten Elementen zu definieren.

2.2.4.2 Service-Discovery

SOAP bietet einen Mechanismus, mit dem über einen definierten Methodenaufruf die auf einem Diensteanbieter („Server“) registrierten Methoden und deren Schemata abgefragt werden können. Diese Beschreibung enthält neben dem Dienstschema auch Informationen zu Adressen und Transportmethode.

Mit WS* lassen sich angebotene Dienste durch WS-Discovery innerhalb eines Broadcast-Mediums für beteiligte Diensteanbieter („Geräte“) am gleichen Medium ermitteln. Hierzu wird eine standardisierte SOAP-Nachricht („Probe“) per Broad- oder Multicast im Medium verteilt, die bestimmte Suchkriteren enthält. Jedes Gerät, das die Nachricht empfangen hat und den Kriterien entspricht, antwortet mit einer Schemabeschreibung

seiner Dienste.

Der Discovery-Prozeß verläßt sich auf korrekt eingerichtetes Routing der untergeordneten OSI-Schichten. Erreicht eine Nachricht die Grenze des Broadcast-Mediums, so können auch passende Geräte dahinter nicht mehr entdeckt werden.

2.2.4.3 Service-Description

Ein Webservice wird über eine Instanz einer WSDL-Beschreibung definiert. Es werden dabei die Elemente, ihre erlaubte Struktur zueinander, minimale und maximale Anzahl sowie Datentypen definiert. Über spezielle fest definierte Elemente werden Aktionen vorgesehen, z.B. mittels einer URL. Der verarbeitende Client kann damit über den angegebenen Transportweg bereits etablierte Standards zum Aufrufen des Services verwenden.

WSDL definiert dazu verschiedene Element-Typen:

- *Service.* Definiert den Webservice.
- *Endpoint.* Definiert einen Dienste-Endpunkt und seine Erreichbarkeit (z.B. URL) innerhalb eines `service`-Element.
- *Binding.* Definiert einen Transportweg und eine dafür zulässige Operation.
- *Interface.* Definiert eine Schnittstelle auf einen Webservice und verknüpft die zulässigen Nachrichtentypen (`types`) mit einer Operation.
- *Types.* Beschreibt die Nachricht (Datentypen, Reihenfolge, Vorkommen, Beschränkungen) in ihrem Aufbau mittels XSD.

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/wsdl20sample"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/wsdl20sample">
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               xmlns="http://www.tmsws.com/wsdl20sample"
               targetNamespace="http://www.example.com/wsdl20sample">
      <xs:element name="request"> ... </xs:element>
      <xs:element name="response"> ... </xs:element>
    </xs:schema>
  </types>
  <interface name="Interface1">
    <fault name="Error1" element="tns:response" />
    <operation name="Opp1" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input messageLabel="In" element="tns:request" />
      <output messageLabel="Out" element="tns:response" />
    </operation>
  </interface>
  <binding name="HttpBinding" interface="tns:Interface1"
           type="http://www.w3.org/ns/wsdl/http">
    <operation ref="tns:Get" whttp:method="GET" />
  </binding>
```

```
26   <service name="Service1" interface="tns:Interface1">
27     <endpoint name="HttpEndpoint" binding="tns:HttpBinding"
28       address="http://www.example.com/rest/" />
29   </service>
30 </description>
```

Listing 2.12: Beispielhafte, quasi-minimale WDSL-Beschreibung. Nach Definition der beiden (nicht dargestellten) Nachrichtenformate innerhalb des `types`-Elements folgt eine Interface-Beschreibung, die eine Fehlermeldung und zwei reguläre Operationen definiert. Die beiden unteren Elemente weisen den verarbeitenden Client an, die Anfrage an eine bestimmte URL über HTTP mit der Methode `GET` zu richten.

Primitive Datentypen (Ganzzahl, Fließkommazahl, Zeichenkette, assoziatives Array, etc.) lassen sich zu komplexeren Datentypen (`complexType`) kombinieren. Logisch verknüpfte Daten, wie z.B. eine Postleitzahl und der zugehörige Städtename, können so einfacher verarbeitet werden. Eine beispielhafte Webservice-Definition (ohne XSD-Inhalte) ist in Listing 2.12 dargestellt.

In der Praxis hat sich die Verwendung von HTTP und die damit einhergehende Methodik der Adressierung und des Routings von Nachrichten auf verschiedenen Schichten mittels Angabe einer URL, Namensauflösung durch das Domain Name System (DNS) oder direkter Angabe von IP-Adressen etabliert.

2.2.4.4 Service-Consumption

Nach dem in der WSDL angegebenen Schema ist ein XML-Text zu konstruieren, der die erforderlichen Parameter und ihre Werte für eine Anfrage enthält, mindestens also den Namen der aufgerufenen Methode. Sodann ist die Nachricht zum Diensteanbieter zu übermitteln, der die Anfrage verarbeitet und die Antwort an den Client, wiederum nach dem Schema der WSDL-Beschreibung, zurücksendet.

2.2.4.5 Anmerkungen

Durch die Beschreibung von Webservices mittels WSDL bietet SOAP einem Entwickler Zeitersparnis: gängige Programmierumgebungen wie Visual Studio 2010 von Microsoft oder Eclipse bieten einfache Importwerkzeuge, um aus WSDL-Dateien direkt Code der jeweiligen Programmiersprache und Methoden-Stubs zu erzeugen. Der Entwickler versieht Letztere lediglich mit seiner Business-Logik und erzeugt passende Rückgabewerte.

Allerdings erreicht eine SOAP-Nachricht unter Umständen, abhängig von einer Vielzahl von Bedingungen, eine Größe von mehreren 10 kB. Soll diese Nachricht zusätzlich gegen ein bestehendes WSDL-Schema verifiziert werden, ist die Rechen- und Speicherkapazität von eingebetteten System möglicherweise nicht mehr ausreichend. Eine weitere Evaluation findet in Kapitel 2.2.8 statt.

2.2.5 Devices Profile for Web Services (DPWS)

Einen schlankeren Ansatz bietet das in seinem Funktionsumfang reduzierte und durch erforderliche, semantisch auf eingebettete Geräte ausgerichtete Datenfelder eher auf Anforderungen im Smart Home passende Devices Profile for Web Services (DPWS) [42].

Es basiert auf dem Standard SOAP und bedient sich damit ebenfalls XML, um auch ressourcenbeschränkte eingebettete Systeme mit der Technik der Webservices auszustatten. Besonderer Wert wurde auf einen geringeren Overhead als bei SOAP gelegt, damit auch Geräte mit wenig Datenspeicher nicht schon am Verarbeiten der umfangreichen Textkonstanten scheitern; von den spezifizierten WS*-Schemata werden daher nur Teile einiger Spezifikationen und Elemente verwendet:

- *WS-Addressing.* Adressierung von Diensten in IP-basierten Netzen mittels verschiedener Transportprotokolle (eMail, File Transfer Protocol (FTP), Leightweight Directory Access Protocol (LDAP), HTTP, etc.) unter Berücksichtigung der jeweiligen Netzwerkadressierungseigenschaften.
- *WS-Discovery.* Entdecken von Webservices und Geräten über Proxy-Services (*Managed Mode*) oder Broadcast-Nachrichten in lokalen Netzwerken (*ad-hoc-Mode*).
- *WS-Eventing.* Client- und Serverseitiges Behandeln von Ereignissen und Abonnements von Diensten.
- *WS-Policy.* Definiert Schemata für Sicherheit und Zugriffsberechtigungen für Webservices.
- *WS-MetadataExchange.* Bietet um Menschen- und Maschinenlesbarkeit von Diensten durch Angabe von textuellen Beschreibungen unterschiedlicher Sprachen.
- *Webservice Description Language (WSDL).* Bereits eingeführte und beschriebene Definition eines Webservices.

Von SOAP unterscheidet sich DPWS neben dem verringerten Set an unterstützten Schemata in seinem expliziten Ziel, sowohl für Maschinen als auch für den menschlichen Benutzer mittels XML-basierter Texte sprechende Informationen bereitzustellen. Auch wird explizit nur IP und UDP für den Nachrichtentransport vorgeschlagen.

2.2.5.1 Kommunikationsform und Nachrichtenaustausch

DPWS unterteilt einen Diensteanbieter in das angesprochene Device („Host“) sowie die darauf angebotenen Services („Hosted Service“). Ihre Relation wird in der Service-Beschreibung des Hosts anhand der WSDL und dem von DPWS festgelegten Schema definiert. Es finden sich darin Informationen zum Host, seinen Metadaten und den angebotenen Diensten. Eine Beschreibung eines Dienstes an sich ist separat für jeden solchen in einer eigenen WSDL-Beschreibung anzufertigen.

Sofern ein Client oder ein Server das (Sub-)Netz betritt oder sich Metadaten (Zieladresse, Textbeschreibung, Version, etc.) ändern, ist eine sogenannte `Hello`-Nachricht über Broad- oder Multicast abzusetzen, wie in Listing 2.13 dargestellt. Sie beinhaltet Absenderin-

formationen und informiert andere Teilnehmer über ein neues Gerät zur Abfrage der Metadaten.

```
<s:Envelope ... >
  <s:Header ... >
    <a:Action ... >
      http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Hello
    </a:Action>
    <a:MessageID ... >xs:anyURI</a:MessageID>
    <a:To ... >urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</a:To>
  </s:Header>
  <s:Body ... >
    <d:Hello ... >
      <a:EndpointReference> ... </a:EndpointReference>
      [<d:Types>list of xs:QName</d:Types>]?
      [<d:Scopes>list of xs:anyURI</d:Scopes>]?
      [<d:XAddrs>list of xs:anyURI</d:XAddrs>]?
      <d:MetadataVersion>xs:unsignedInt</d:MetadataVersion>
    </d:Hello>
  </s:Body>
</s:Envelope>
```

Listing 2.13: Beispielhafte `Hello`-Nachricht eines Devices, das das Netzwerk betritt. Im Wesentlichen wird die eigene ID, eine oder mehrere Absenderinformationen und optional eine Liste der eigenen Attribute verbreitet.

Ein aktives Suchen nach Diensten durch Clients ist nach dem Betreten des Netzwerkes über eine `Probe`-Nachricht möglich. In ihr gibt der Anfragende seine eigene Erreichbarkeit bzw. Routingadresse(n) an und legt gewünschte Eigenschaften der zu findenden Dienste fest:

- *Types-Element.* Enthält eine Liste der zu findenden Typbezeichnungen, Freitext
- *Scope-Element.* Enthält eine Liste von DNS, LDAP oder sonstigen einschränkenden Merkmalen

Sofern ein *Hosted-Service* den Kriterien entspricht, sendet er eine `Probe-Match`-Nachricht über Unicast-Adressierung an den anfragenden Client. Die Nachricht enthält die eigene Endpunktbeschreibung und Transportadresse des Dienstes. Über das Datenformat eines Endpunktes muß keine Aussage getroffen sein. Beispielsweise beschreibt DPWS bei Netzwerkdruckern nur den Transportweg der binären Druckdaten (Raster- oder PostScript-Daten, je nach Druckermodell).

2.2.5.2 Metadaten

DPWS führt eine feste Definition von Metadaten für Geräte und Services ein, die bestimmte Informationen beinhalten. Dabei wird zwischen Modell („ThisModel“) und Instanz („ThisDevice“) unterschieden, wodurch gerätespezifische Informationen festgehalten werden können, etwa die Seriennummer. Alle Felder können in mehreren Sprachen (z.B. Attribut `xml:lang="de_DE"` für deutsche Bezeichnungen) lokalisiert angegeben werden.

Für ein Modell bzw. eine Geräteserie sind mit die folgenden Daten definiert:

- *dpws:ThisModel/dpws:Manufacturer*. Herstellername
- *dpws:ThisModel/dpws:ManufacturerUrl*. URL der Herstellerwebseite
- *dpws:ThisModel/dpws:ModelName*. Name der Modellreihe („Laserdrucker“)
- *dpws:ThisModel/dpws:ModelNumber*. Bezeichnung der Modellnummern
- *dpws:ThisModel/dpws:ModelUrl*. URL der Modell-Reihe im Internet

Für ein spzifisches Gerät sind die folgenden Daten definiert:

- *dpws:ThisDevice/dpws:FriendlyName*. Name des Gerätes
- *dpws:ThisDevice/dpws:FirmwareVersion*. Firmwareversion, Zeichenfolge
- *dpws:ThisDevice/dpws:SerialNumber*. Eindeutige Seriennummer, Zeichenfolge

Die Versionierung der Metadaten findet in den jeweiligen Discovery-Nachrichten *Hello*, *Probe Match* und *Resolve Match* über das Element `wsd:MetadataVersion` statt.

2.2.5.3 Ereignisse und Abonnements

Eine notwendige Funktion im Smart Home ist das ereignisbasierte Aktualisieren von Werten. DPWS bietet eine Abonnement-Verwaltung nach Publish/Subscribe-Mechanismus. Die jeweiligen XML-Schemata entsprechen denen aus WS-Eventing. Ein Unterschied besteht einzig im Transport, er ist hier über UDP (Multicast) ausdrücklich zulässig und vorgesehen, um etwa Multicast-Events zu versenden.

Clients können sich bei einem Service für die Benachrichtigung anmelden („Subscribe“), den Abonnementstatus abfragen – wenn etwa einen gewissen Zeitraum lang keine Nachrichten mehr empfangen wurden – und das Abonnement beenden. Hosted-Services können Clients über die Einstellung des Dienstes informieren, wenn sie das Netzwerk verlassen. Notwendig ist diese Art der Verwaltung, da wegen des Transports über UDP ein zuverlässiger Nachrichtentransport nicht mehr sichergestellt ist. Client und Server müssen sich daher asynchron über Zustände austauschen können.

2.2.5.4 Implementierungen

Es existiert eine Forschungsgruppe an der Universität Rostock, die sich mit DPWS beschäftigt und offene Implementierungen für verschiedene Programmiersprachen zur Verfügung stellt [43]:

- *Webservices for Devices (WS4D)-gSOAP (C/C++).* WS4D-gSOAP ist ein Toolkit, das die Programmierung von Webservice-Konsumenten und Dienste-Anbietern in C oder C++ zu programmieren. Es setzt dabei auf das gSOAP-Framework, das SOAP-Funktionen bereitstellt und Nachrichtenverifikation gegen XML-Schemata durchführen kann. Durch die reine C-Implementierung wird eine Reihe von Architekturen unterstützt, wobei jeweils eine Adaption der Sockets stattfindet.

- *WS4D-uDPWS (C).* Insbesondere für batteriebetriebene Geräte, die kein Energy-Harvesting betreiben können, stellt sich die Frage nach einem energieeffizienten Kommunikationsprotokoll. Das Betriebssystem Contiki setzt dafür auf den Funkstandard Internet Protocol, Version 6 (IPv6)-over-LowPower-Wireless-Networks (6LoWPAN), der den IEEE-Standard 802.15.4 auf 868 MHz, 900 MHz und 2,4 GHz Frequenzbereichen verwendet. Damit werden Nutzdaten über IPv6 und UDP übertragen. Die gesamte Bibliothek, bekannt unter `WS4D-uDPWS`, setzt als Grundlage somit 6LoWPAN und Contiki voraus und ist in der vorliegenden Implementierung für die Hardware des Telos B Sensorknotens und des AVR-Raven Experimentierboards ausgelegt. Für eigene Hardware ist Treiber im Betriebssystem anzupassen.
- *JMEDS (Java).* Für die Programmiersprache Java existiert die Implementierung `JMEDS`. Sie unterstützt alle Java-Editionen und bietet gegenüber der *C*-Implementierung erweiterte Funktionen. Dazu zählen ein automatisch aus Geräteschreibungen generiertes Webinterface für IP-basierten Transport, ein automatisches Erzeugen von korrekten WSDL-Beschreibungen für Dienste und eine größere Modularität, mit der nicht-benötigte Teile der Bibliothek vom Entwickler ausgeschlossen werden können.
- *WS4D-JCOAP (JAVA).* Eine weitere, nennenswerte und exzellent gewartete Implementierung in Java ist WS4D-JCOAP. Als Transport für DPWS wird anstatt HTTP allerdings das CoAP verwendet, das für 6LoWPAN entwickelt wurde und anders als HTTP auch UDP-Broad- und Multicast-Nachrichten verarbeitet. Weiterhin ist auf Basis dieser Bibliothek bereits ein generischer JCOAP-zu-HTTP-Proxy, der automatisiert zwischen HTTP- und CoAP auf Anwendungsschicht vermittelt.

2.2.5.5 Anmerkungen

In den vorausgegangenen Ausführungen wurden nur wenige wesentliche Eigenschaften beschrieben. Dennoch ist bereits ein deutlicher Fokus auf den Einsatz im Smart Home erkennbar: insbesondere tragen dazu die vordefinierten und im Zuge der Internationalisierung von Software auch erforderlichen Felder der semantischen Beschreibung für Geräte und Dienste sowie die der Fokus auf eingebettete Plattformen bei.

Allerdings sind Entwickler auch bei DPWS auf ein statisches Schema beschränkt. Ob dies eine nennenswerte Einschränkung darstellt und ob die Abgrenzung von SOAP bereits ausreicht, wird in Kapitel 2.2.8 diskutiert.

2.2.6 Constrained Application Protocol (CoAP)

Ein aktueller Forschungsansatz im Bereich der Kommunikation im Smart Home wird von der Gruppe CORE der Internet Engineering Task Force (IETF) unter dem Namen „core working group“ betrieben. Im Wesentlichen zielt die Gruppe auf die Spezifikation eines Transportprotokolls, welches HTTP ersetzt (CoAP [44]), und ein darauf aufsetzendes Anwendungsprotokoll (CoRE [45]) zum Beschreiben, Auffinden und Benutzen von Diensten

auf Smart Devices ab.

CoAP ist ein reines Transportprotokoll und bietet keine Eingenschaften zum Beschreiben oder Auffinden von Diensten. Die vorgesehene Anwendungsschicht (CoRE) behandelt diese Punkte. Es werden in diesem Kapitel daher nur die Nachrichtentypen und der Nachrichtenversand beschrieben. Zusammenfassend ist CoAP wie folgt motiviert:

- *Verarbeitung von HTTP.* Für das Interpretieren sind eine Reihe von Zeichenketten-verarbeitenden Funktionen (`str[n]cmp()`, `str[n]tok()`, `str[n]len()`) notwendig, die möglicherweise nicht auf allen Plattformen zur Verfügung stehen und speicher- und rechenintensiv verarbeitet werden müssen.
- *Transport nur über TCP.* HTTP wurde für den Transport über das zuverlässige TCP/IP-Protokoll entworfen. Eine Nachricht kann nun wesentlich umfangreicher als die maximale Rahmengröße eines IP-Paketes sein, daher scheidet ein Transport über potentiell verlustbehaftete Transportprotokolle wie UDP aus. HTTP ist nicht gegen Verluste von beliebigen Teilen von Nachrichten geschützt.
- *Senden unbestätigter Nachrichten.* HTTP ist nur für Punkt-zu-Punkt-Verbindungen und Anfrage/Antwort-Nachrichtenaustausch vorgesehen. Gerade im Smart Home aber sind Ereignisse, die von mehreren Empfängern verarbeitet werden müssen und keine Antwort bedingen, ein realer Anwendungsfall.

CoAP verringert die Nachrichtengröße von HTTP durch binäre Codierung von Methode und Header-Keys und durch Wegfall von (nach Meinung der Autoren) unwichtigen Eigenschaften. Es ergibt sich eine geringere Komplexität und letztlich Energieeinsparung, die es für den Einsatz in eingebetteten Systemen prädestinieren.

Das Parsen der binären Daten wird durch feste Feldlängen unter Beibehaltung des HTTP-Schemas (Verb, URL, Header bzw. „Options“, optionale Nutzdaten) vereinfacht.

Weiterhin unterstützt CoAP Zuverlässigkeit auf Anwendungstransportprotokollebene, indem Nachrichten mit einer ID versehen und als „zu bestätigen“ markiert versendet werden. Timer und Zähler bestimmen, ob und wann die Anwendungssoftware über einen fehlgeschlagenen Versand benachrichtigt wird.

2.2.6.1 Nachrichtentypen und -Aufbau

CoAP ersetzt HTTP als Anwendungsprotokoll und setzt selbst auf User Datagram Protocol (UDP) als Transportprotokoll. Insbesondere der damit einhergehende potentielle Verlust von Nachrichtenpaketen wird vom Protokoll abgefangen und behandelt.

In CoAP existieren vier Nachrichtentypen, die im Feld `Type` übertragen werden:

- `CON` mit Codierung 0b00. Die Nachricht muss vom Server bestätigt werden. Die Antwort kann in einer `ACK`-Nachricht (mit oder ohne Nutzdaten) oder in einer `RST`-Nachricht bestehen.

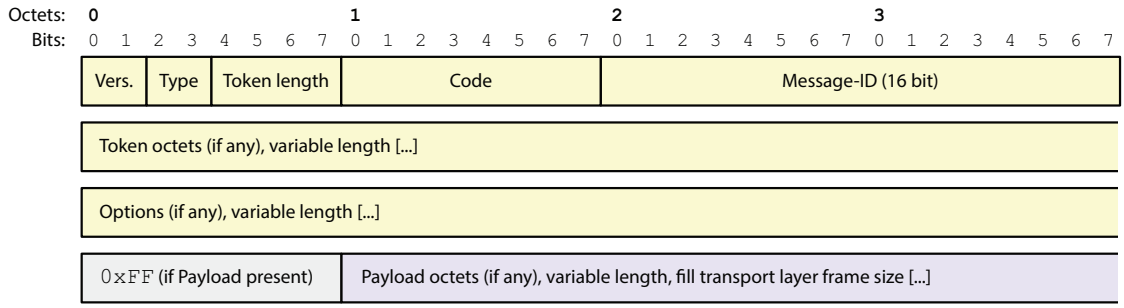


Abbildung 2.12: Aufbau einerCoAP-Nachricht.Das Feld `Version` ist konstant auf 1 gesetzt. Im Feld `Type` wird der Nachrichtentyp codiert. `Code` beinhaltet für Anfragen die Methode, bei Antworten einen dem HTTP ähnlichen Response-Code. Die `Message-ID` entspricht der TCP-Sequenznummer, während das `Token` mit einer Session-ID vergleichbar ist. Ohne optionale Inhalte kann ACK-Nachricht ohne Token aus nur 4 B bestehen.

- `NON` mit Codierung 0b01. Die Nachricht wird nicht bestätigt und kann eine Antwort mit Nutzdaten nach sich ziehen. Im Fehlerfalle wird keine Antwort gesendet.
- `ACK` mit Codierung 0b10. Bestätigt den Erhalt einer Anfrage. Sie kann gleichzeitig Antwortdaten enthalten oder signalisiert durch fehlende Daten, daß die Antwort später vom Server nachgereicht wird.
- `RST` mit Codierung 0b11. Enthält keine Antwortdaten und signalisiert einen Fehler.

Für CoAP sind, angelehnt an HTTP und WebDAV, vier Methoden definiert, die im `Code`-Feld der Nachricht bei Anfragen übertragen werden:

- `GET` mit Codierung 0b001. Fordert eine Ressource an.
- `PUT` mit Codierung 0b010. Überschreibt einen Wert einer vorhandenen Ressource.
- `POST` mit Codierung 0b011. Erstellt eine neue Ressource.
- `DELETE` mit Codierung 0b100. Löscht eine Ressource.

Ähnlich der TCP-Sequenznummer wird in CoAP eine 2 Byte große Sequenznummer definiert, die vom Absender erstmalig vergeben und für jede weitere Nachricht inkrementiert wird. Client und Server können so bei unterschiedlichen Verbindungen zu unterschiedlichen Teilnehmern die über UDP eintreffenden Nachrichten den jeweiligen Anfragen zuordnen und eine korrekte Reihenfolge sicherstellen. Der Aufbau ist in Abbildung 2.12 dargestellt.

Bei CoAP werden HTTP-Header durch sogenannte „Options“ ersetzt. Jede Option besteht dabei aus einem Header (ein oder zwei Byte) und bis zu 270 Byte Inhalt. Im Header codiert das High-Nibble des ersten Byte die Optionsnummer, die den Header-Namen („Key“) des HTTP-Headers ersetzt. Die Nummer ist dabei als Differenz zu ihrem Vorgänger gespeichert, mehrere Optionswerte des gleichen Typs sind damit mit einem Delta von 0 im ersten Nibble codiert. Ist eine Option größer als 14 Byte, so codiert das zweite Byte die restlichen Länge. Der Aufbau ist in Abbildung 2.13 abgebildet. Die Menge der verfügbaren Optionen ist im Gegensatz zu HTTP begrenzt und in Tabelle 2.3 dargestellt.

Auch der URL-Pfad wird in seinen Bestandteilen als Option codiert. Beispielhaft würde der Pfad `/test/index.html` in zwei Optionen `URI-Path:` mit Optionsinhalt `test` und `index.html`

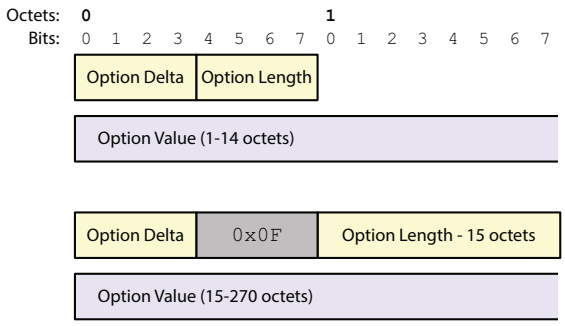


Abbildung 2.13: Aufbau des `Option`-Feldes in CoAP, sowohl für eine Option mit weniger als 15, als auch für eine Option mit 15 oder mehr Byte Nutzdaten. Die maximale Größe einer Option ist auf 270 B limitiert, da die beiden Längenteile addiert $15 + 255 = 270$ ergeben.

übertragen. Nachdem in diesem Beispiel pro solcher Option nur ein Header-Byte nötig ist, erhöht sich die Gesamtgröße der URL nicht.

In Tabelle 2.4 sind mögliche Werte für die `Content-Format:`-Option dargestellt. Die verfügbaren Werte werden in einem Byte codiert und ersetzen den `Content-Type:`-Header aus HTTP. Gegenüber dem freien Text ergibt sich eine wesentlich geringere Größe; allerdings sind die möglichen Werte auf die definierten Werte begrenzt.

Options-ID	Art	Mehrfach	Bezeichnung	Datentyp	Länge	Standard
1	Critical	ja	If-Match	`opaque`	0-4 B	*keiner*
3	Critical	–	URI-Host	`string`	1-255 B	-
4	Elective	ja	ETag	`opaque`	1-8 B	*keiner*
5	Critical	–	If-None-Match	*leer*	0	*keiner*
7	Critical	–	URI-Port	`uint`	0-2 B	-
8	Elective	ja	Location-Path	`string`	0-255 B	*keiner*
11	Critical	ja	URI-Path	`string`	0-255 B	*keiner*
12	Elective	–	Content-Format	`uint`	0-2 B	*keiner*
14	Elective	–	Max-Age	`uint`	0-4 B	60
15	Critical	ja	URI-Query	`string`	0-255 B	*keiner*
17	Critical	–	Accept	`uint`	0-2 B	*keiner*
20	Elective	ja	Location-Query	`string`	0-255 B	*keiner*
35	Critical	–	Proxy-URI	`string`	1-1034 B	*keiner*
39	Critical	–	Proxy-Scheme	`string`	1-255 B	*keiner*
60	Elective	–	Size1	`uint`	0-4 B	*keiner*

Tabelle 2.3: Verwendete Optionen sowie deren Eigenschaften im Constrained Application Protocol. Wie erkennbar, sind nur einige wenige Felder aus HTTP auch in CoAP verfügbar. Optionen, die als `Critical` gekennzeichnet sind, müssen von Clients mit einer `RST`-Nachricht beantwortet werden, wenn sie nicht unterstützt werden.

Optionswert	Bedeutung	Spezifiziert in
0	text/plain;charset=utf-8	RFC2046, RFC3676, RFC5147
40	application/link-format	RFC6690
41	application/xml	RFC3023
42	application/octet-stream	RFC2045, RFC2046
47	application/exi	EXI-Spezifikation
50	application/json	RFC4627

Tabelle 2.4: Mögliche Werte der CoAP-Option `Content-Format:`. Übertragene Inhalte müssen in einem der gelisteten Werte repräsentiert werden können. Ein Client kann eine bestimmte Ressource in einer der angegebenen Repräsentationen anfordern; kann der Server dies nicht erfüllen, ist eine entsprechende Fehlermeldung zurückzusenden. Gegenüber dem freien Text in HTTP ergibt sich durch nur ein Byte Platzbedarf eine wesentliche Reduzierung der Nachrichtengröße, wenn die Option verwendet wird. Außer beim Standardwert von `text/plain` ist keine Aussage über die Codierung des Inhalts getroffen.

Da CoAP im Gegensatz zu HTTP auch einen zeitlich asynchronen Nachrichtenaustausch unterstützt, bezieht sich die Sequenznummer nur auf einen einzigen Frage-Antwort-Zyklus. Damit kann auf eine Anfrage eines Clients lediglich eine einzige Antwort des Servers eineindeutig zugeordnet werden. Soll über mehrere zeitlich auseinanderliegende Nachrichten eine Zuordnung zu einem bestimmten Vorgang ermöglicht werden, so ist ein spezielles Header-Feld (`Token`) zu verwenden. Es wird für zusammenhängende Nachrichten, z.B. Auslesen eines Sensors, das über Minuten hinweg dauern kann, verwendet.

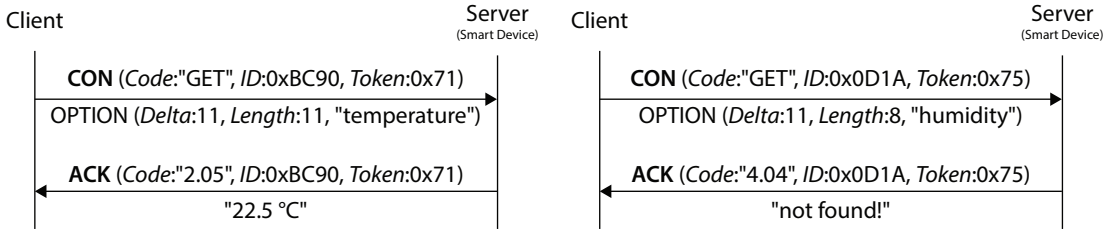
Client und Server können so Anfragen mit unterschiedlichen Sequenznummern einem einzigen logischen Vorgang zuordnen. CoAP bezeichnet diesen Mechanismus als „Request / Response-Matching“. Ein gebräuchliches Synonym im WWW lautet „Session-ID“.

2.2.6.2 Nachrichtenaustausch

Abgeleitet aus den vorigen Ausführungen werden einige Beispiele für einen Nachrichtenaustausch, entnommen aus der Spezifikation, angegeben. Insbesondere wird gezeigt, daß sich CoAP eng am Vorbild HTTP orientiert. In Abbildung 2.14 ist eine zuverlässige Kommunikation dargestellt, jeweils mit vorhandener und nicht vorhandener Ressource.

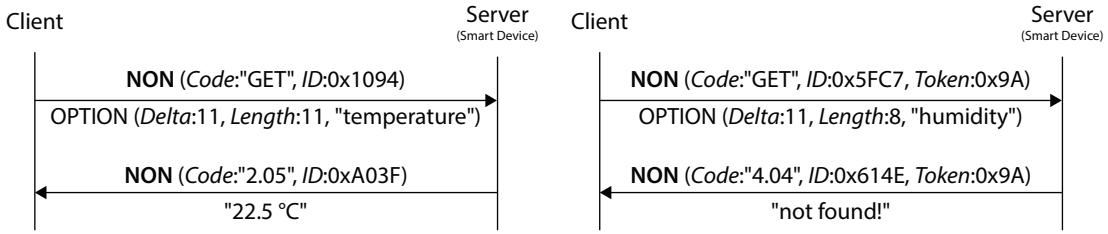
In Abbildung 2.15 werden zwei unzuverlässige Nachrichten verwendet. Sie können jederzeit auf dem Transportweg verlorengehen. Die Nachrichten-ID ist im Gegensatz zum bestätigten Austausch jeweils eine andere. Eine Zuordnung ist nur bei gesetztem Token möglich. Insbesondere auf gemeinsam genutzten Broadcast-Medien gewinnt diese Eigenart große Bedeutung: unter Umständen können bei Mißachtung dieses Verhaltens Antworten falsch zugeordnet werden.

Muß der Server erst eine Zeitlang eine Anfrage verarbeiten (Abbildung 2.16), z.B. intensive Rechenoperationen ausführen, und kann nicht unverzüglich antworten, so bestätigt er die `CON`-Anfrage mit einem leeren `ACK`.



(a) Antwort bei gefundener Ressource. **(b)** Antwort bei fehlender Ressource.

Abbildung 2.14: Zuverlässiger Nachrichtenaustausch. Anfragen und Inhalte werden mittels CON und ACK-Nachrichten ausgetauscht, unabhängig vom jeweiligen Ergebnis.



(a) Antwort bei gefundener Ressource. **(b)** Antwort bei fehlender Ressource.

Abbildung 2.15: Unzuverlässiger Nachrichtenaustausch. Anfragen und Inhalte werden mittels NON-Nachrichten ausgetauscht, wiederum unabhängig vom jeweiligen Ergebnis. Das Token sollte zur eindeutigen Zuordnung von Anfrage und Antwort gesetzt werden (2.15b), damit die Antwort zugeordnet werden kann.

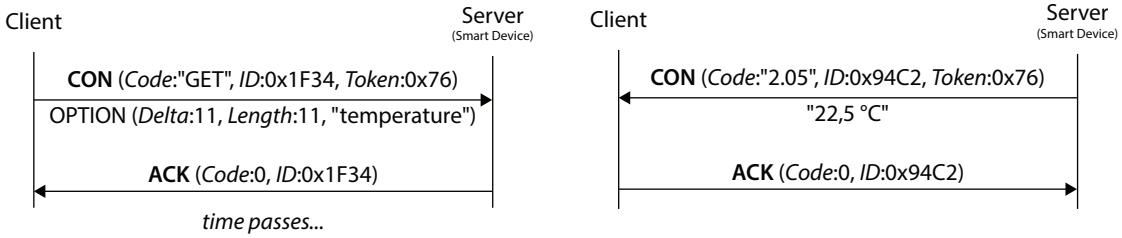


Abbildung 2.16: Darstellung einer zeitversetzten Antwort vom Smart Device unter Verwendung zuverlässiger Kommunikation. Zu beachten ist, daß die Bestätigungen außer der jeweiligen Message-ID keine Daten enthalten. Weiterhin ist im rechten Teil dargestellt, daß der Nachrichtentyp unabhängig von der Methode oder dem Nachrichtencode ist. Sowohl Anfragen wie auch Antworten können mittels CON-Nachrichten übertragen werden.

2.2.6.3 Implementierungen

CoAP läßt sich als reines Anwendungsschichtprotokoll in gängigen Programmiersprachen implementieren. Da es noch nicht als Standard, sondern lediglich als Entwurf verfügbar ist, sind finale Implementierungen derzeit (September 2013) noch nicht verfügbar. Es existieren allerdings Vorabversionen für verschiedene Programmiersprachen, die jeweils unterschiedliche Revisionen der CoAP-Spezifikation implementieren, siehe Tabelle 2.5.

2.2.7 Constrained ReSTful Environments (CoRE) Link Format

Da CoAP nur den Transport von Nutzdaten abdeckt, nicht aber die Aufgaben der Discovery und Description wahrnimmt, wurde von der Constrained ReSTful Environments Working Group das CoRE Link Format definiert, das zwischenzeitlich bereits als RFC6690 [46] veröffentlicht ist.

Obwohl als reines Anwendungsschichtprotokoll spezifiziert, setzt es auf CoAP auf und verwendet dessen Eigenschaften, wie etwa die Option `Content-Format:` mit dem Wert `application/link-format` oder die Multicast-Unterstützung. Im Folgenden wird das Link Format synonym mit CoRE bezeichnet.

2.2.7.1 Nachrichtenformat

Nutzdaten werden im Protokoll über das eigens definierte Link-Format ausgetauscht. Die vollständige Grammatik des Nachrichteninhaltes kann dem RFC entnommen werden. Ressourcen, die über das Link-Format verbreitet werden, werden im Nutzdatenteil von CoAP als reiner Text übertragen. Sie werden durch ein Komma voneinander getrennt und können mehrere Attribute besitzen, von denen die gebräuchlichsten nachfolgend gelistet sind:

Name der Implementierung	Sprache	Draft-Version	Letztes Updates
libcoap	C	13	Januar 2013
ncoap	Java	8	Juli 2013
jcoap	Java	13	Juni 2012
Californium	Java	13	Juli 2013
Copper	JavaScript	18	Juli 2013
TinyOS-Implementierung	nesC/C	13	Juli 2013
Contiki-Implementierung	C	13	Juli 2013

Tabelle 2.5: Liste der nach Meinung des Autors wichtigen Implementierungen von CoAP. Nur eine Implementierung beinhaltet den aktuellen Stand (Version 18); insbesondere durch mögliche Änderungen der Codetabellen („Value Registries“) sind die Bibliotheken daher möglicherweise inkompatibel zueinander. Closed-Source-Projekte sind nicht aufgeführt.

- *Unified Resource Identifier (URI)-Pfad, „href“.* Absoluter URI-Pfad zur Ressource auf dem Server, in spitzen Klammern gesetzt. Ein Client fordert den vollständigen Pfad an und setzt im Falle von CoAP die Option `URI-Path:` für jeden Teil.
- *Ressourcen-Typ, Attribut „rt“.* Definiert eine frei wählbare Zeichenkette, die den Typ einer Ressource näher spezifiziert. Ob dabei eine Ontologie verwendet wird, bleibt dem Entwickler überlassen. Vorgeschlagen ist außerdem, eine eventuelle Einheit in diesem String mit aufzunehmen, wobei keine Angaben zur Codierung Letzterer gegeben werden. Es können mehrere Werte des Attributes durch Leerzeichen getrennt angegeben werden.
- *Schnittstellenbeschreibung, Attribut „if“.* Gibt einen Link oder einen Namen einer Schnittstellenbeschreibung an. In dieser ist beschrieben, wie mit der genannten Ressource kommuniziert werden kann. Vorgeschlagen dafür ist eine Beschreibung über die Web Application Description Language (WADL) [47], die ihrerseits auf XSD beruht.
- *Maximale geschätzte Größe, Attribut „sz“.* Ist zu setzen, wenn der Server eine maximale Größe des Wertes abschätzen kann.
- *Textueller Name, Attribut „title“.* Definiert einen Namen, der einem menschlichen Benutzer die Interpretation der jeweiligen Ressource erleichtern soll.
- *Anker, Attribut „anchor“.* Kann mehrere Aufgaben erfüllen. Dient zum einen als alternativer (kürzerer) Name für eine bestimmte Ressource. Kann zum anderen aber auch vollkommen andere, auch WWW-Ressourcen oder Ressourcen auf anderen Devices, in die eigene Link-Format-Beschreibung einbinden.

2.2.7.2 Service-Discovery und Description

Das Protokoll orientiert sich in seinem Nachrichtenformat an den aus HTML bekannten Links. Zur Auflistung der verfügbaren Ressourcen auf einem oder mehreren Smart Devices sendet der Client über CoAP eine `GET`-Anfrage mit URL-Pfad `/.well-known/core`:

```
1 >>> HTTP: GET /.well-known/core HTTP/1.1

2 >>> CoAP-Methode:    GET
3 >>> CoAP-Option[11]: /.well-known
4 >>> CoAP-Option[ 0]: /core
```

Listing 2.14: Vergleich der Discovery-Anfrage zwischen HTTP und CoAP.

Die Discovery-Nachricht kann über Multicast versendet werden, worauf Service-Anbieter ihre Antwort über Unicast an den Absender zurücksenden. Im Falle von IP/UDP erfolgt dann eine Zuordnung über die IP-Adresse[1]. Eine Antwort erfolgt im Link-Format und

1 Anmerkung: Die Port-Nummer ist bei CoAP fest definiert und kann damit nicht zur Unterscheidung herangezogen werden. Im Rahmen von IPv6 bedeutet das aber wegen ausreichender Zahl von Adressen kein Hindernis.

könnte wie in Listing 2.15 ausfallen.

```
</sensors>;ct=40;title="Sensor Index",
</sensors/temp>;rt="temperature-c";if="sensor",
</sensors/light>;rt="light-lux";if="sensor",
</t>;anchor="/sensors/temp";rel="alternate"
```

Listing 2.15: Beispielhafte Antwort auf eine Discovery-Anfrage mittels CoAP. Zeilenumbrüche sind nur der Lesbarkeit halber eingefügt. CoAP-Protokollspezifika werden nicht dargestellt.

Wie bereits im vorigen Abschnitt erwähnt gibt CoRE nur den Pfad und einige wenige Attribute zu einer Ressource an, beschreibt aber nicht das Maschineninterface, mit welchem sich tatsächlich Daten abrufen lassen.

Dies wird mit Hilfe einer WADL-Beschreibung, die separat bereitzustellen ist und die detailliert die Eigenschaften der Schnittstelle beschreibt, realisiert. Dazu zählen die erlaubten Methoden (beschränkt auf die Fähigkeiten von CoAP), eventuelle Parameter und die zu erwartenden Antworten. Auch für diese muß letztlich wieder ein Schema definiert werden, das allerdings in der gleichen Datei platziert werden kann.

2.2.7.3 Gezielte Suche nach Diensten

CoRE definiert zudem eine einfache Schnittstelle zur Suche nach Diensten. Sie orientiert sich am Schema von HTTP und erwartet Sucheingaben in einem Schlüssel-/Wertepaar nach einem Fragezeichen des URL („query string“). Der Schlüssel codiert dabei das zu suchende Attribut, während der Wert den zu suchenden Wert angibt.

Dieser muß nicht exakt bekannt sein: ein abschließendes Asterisk („*“) ist ebenfalls zulässig und sucht nach dem ersten Teil einer Zeichenkette. Es ist dabei nur am Ende des Suchstrings erlaubt, da sich die Funktionalität mittels `strncmp()` einfach implementieren läßt. Listing 2.16 führt einige Beispiele an. Suchoptionen können nicht kombiniert werden, es wird nur ein einziges Attribut pro Anfrage unterstützt.

```
GET /.well-known/core?href=/temp*
GET /.well-known/core?title=Temp*
GET /.well-known/core?title=Temperatur
GET /.well-known/core?rt=temp*
```

Listing 2.16: Beispielhafte Suchanfragen nach Ressourcen, die eine Temperaturquelle beinhalten und deren Meßwerte ausliefern.

2.2.8 Nachteile und Schwierigkeiten

In diesem abschließenden Absatz werden die vorgestellten Protokolle für den Einsatz im Smart Home untersucht. Als Basis dienen die Punkte, die in Kapitel 2.2.1.1 und 2.2.1.2 für ein Anwendungs- und Steuerungsprotokoll aufgestellt wurden.

Auch etwaige Implementierungen der jeweiligen Protokolle sollen berücksichtigt werden. Da eine Evaluierung hierbei nur nur gegen konkrete Kriterien erfolgen kann, wird von der in der Einleitung erarbeiteten Anforderung der möglichst gering ausgestatteten Hardware ausgegangen: es erscheint zweckmäßig, die Eckdaten der kleinsten der später in der eigenen Implementierung verwendeten Plattformen dafür heranzuziehen. Sie besitzt 2 kB Random-Access Memory (RAM) und 32 kB Flash-ROM.

2.2.8.1 Hypertext Transport Protocol (HTTP)

Als ein sehr einfaches Protokoll ist HTTP für die Anwendung im Smart Home interessant. Allerdings ergeben sich folgende Schwierigkeiten:

- *Keine Discovery.* Das Protokoll verfügt über keine Möglichkeiten, Dienste und Geräte zu erkennen. Es ist ursprünglich nur für Punkt-zu-Punkt-Verbindungen vorgesehen.
- *Keine Dienstebeschreibung.* Da übertragene Inhalte frei wählbar sind, existiert kein festes Schema zur Beschreibung von Dienste- und Geräteeigenschaften.
- *Keine Broadcast-Fähigkeiten.* HTTP sieht als Anwendungstransportprotokoll keine Möglichkeit der Multi- oder Broadcast-Adressierung auf Anwendungsebene vor. Diese ist bei wie erwähnt unbekannten Transportlösungen aber notwendig.

Der beschriebenen Mängel und fehlenden Features wegen ist HTTP in seiner jetzigen Form für das Smart Home nur zur Steuerung von einzelnen Geräten zu verwenden. Für ein vollständiges Abbilden benötigter Funktionen fehlt die Unterstützung.

2.2.8.2 Extensible Markup Language Remote Procedure Call (XMLRPC)

Als erstes der untersuchten Protokolle verfügt XMLRPC über die Möglichkeit der Introspektion und damit – weiter gefaßt – über das Erkennen von Diensten auf einem bestimmten Gerät. Dennoch ergeben sich für die Verwendung im Smart Home einige Nachteile:

- *Transport über HTTP.* Damit ist zunächst nur eine Punkt-zu-Punkt-Kommunikation zwischen Client und bekanntem Server möglich. Die wichtige Funktion des Erkennens von Geräten in einem Netzwerk fehlt.
- *Fehlende Semantik.* Für maschinelle Benutzer wird die Schnittstelle über Introspektion eineindeutig beschrieben. Außer einem Methodennamen fehlt allerdings die für menschliche Benutzer wichtige Beschreibung des Gerätes an sich und jeder einzelnen Funktion.
- *XML-Verarbeitung notwendig.* Auf einem eingebetteten System muß zwingend XML verarbeitet werden für Anfragen und Antworten. Wenngleich dies mit SAX-Mechanismen wegen des statischen und einfachen Aufbaus des Protokolls prinzipiell funktioniert, stellt es einen unnötigen Overhead dar.

- *Nachrichtengröße.* Wie gezeigt, ist zur Übertragung desselben Nutzdatums im Vergleich zu HTTP ein Vielfaches an Übertragungsvolumen erforderlich.

Hauptsächlich der notwendigen XML-Verarbeitung und des gestiegenen Übertragungsvolumens wegen hält der Autor XMLRPC für nicht geeignet.

2.2.8.3 SOAP

Aufgrund der großen Flexibilität von SOAP ist eine Untersuchung lohnenswert. Durch standardisierte Datentypen wird zudem eine unterschiedliche lokale Repräsentation bestimmter Daten ermöglicht, z.B. die Repräsentation von Gleitkommazahlen in verschiedenen Sprachen. Die Zahl wird in einem definierten Format übertragen, kann dann aber im Client auf geeignete Weise angezeigt werden (z.B. mit Tausendertrennzeichen, Punkt statt Komma als Dezimalmarkierung, etc.).

Für den Einsatz im Smart Home ergeben sich aber die folgenden Nachteile:

- *Komplexer Aufbau von Nachrichten.* SOAP-Nachrichten sind komplex und können je nach Schemadefinition mehrere 10 kB groß werden. Nicht nur ist das für die eigentlich übertragene Information ein inakzeptabler Overhead, sondern überschreitet das auch die Speicherkapazitäten der Zielhardware.
- *Keine fest definierte Ontologie.* Zwar bietet die Beschreibung über eine Schemasprache bereits einen Rahmen für eine gemeinsame Kommunikation; allerdings muß und kann dieser für alle Nachrichten separat definiert werden. Clients und Server müssen somit dynamisch erzeugte Formate verarbeiten und validieren können, um eine fehlerfreie Kommunikation zu ermöglichen. Das Festlegen bleibt weiterhin jedem Teilnehmer und jedem Hersteller überlassen, mit der Konsequenz, daß möglicherweise keine gemeinsame Sprache existiert.
- *Lose Koppelung des Nachrichtentransports.* Zu begrüßen ist zwar die frei definierbare Verwendung mit gängigen Nachrichtentransportprotokollen (FTP, HTTP, etc.). Allerdings kann keine einzige davon in einem Smart Device auf einer eingebetteten Plattform vorausgesetzt werden.
- *Separate WSDL-Schemata.* Die Schemata für die Dienstebeschreibungen müssen separat vorgehalten werden. Es stellt sich die Frage nach der Versionierung, dem Konsistent-Halten und dem Hosting dieser separaten Beschreibungen. Alle drei Punkte auf eingebettetem Level lösen zu wollen, ist nicht praktikabel.
- *Publish/Subscribe-Mechanismus.* Dadurch, daß sich Clients bei Diensteanbietern auf einen bestimmten Wert für festgelegte oder dynamische Aktualisierungen einschreiben können, wird vom Server eine umfangreiche Funktionalität vorausgesetzt. Zum einen werden Ressourcen für die jeweilige Liste an Clients und ihrer jeweiligen Transportparameter benötigt. Zum anderen müssen aber auch die Abonnements an sich (Zeitpunkt letzter Übertragung, Status des Clients, etc.) mit Aufwand verwaltet werden. Für beide Anforderungen ist unter Umständen keine Hardwareunterstützung

(z.B. Uhrzeit, Speicher) vorhanden.

Zwar könnten benötigte Funktionen über SOAP abgebildet und auch mit gängigen Programmiersprachen abgebildet werden. Allerdings ist der Mehrverbrauch an Rechenleistung, Speicherplatz und vorzuhaltender Infrastruktur zu groß für den Einsatz auf eingebetteten Systemen.

Der Autor hält SOAP daher für ungeeignet für den Einsatz im Smart Home.

2.2.8.4 Devices Profile for Web Services (DPWS)

Das Protokoll beruht in wesentlichen Eigenschaften auf seinem Vorbild SOAP. Immerhin sind bei DPWS einige beschreibende Datenfelder festgelegt und müssen von Geräten und angebotenen Dienste zwingend belegt werden. Auch eine Mehrsprachlichkeit und Internationalisierung wird in diesem Rahmen unterstützt. Allerdings werden die meisten im vorigen Kapitel angegebenen Schwächen von SOAP übernommen:

- Auch das reduzierte Schema kann eine Einbindung und Verwendung von Smart Devices gegenüber SOAP nicht wesentlich vereinfachen und bietet möglicherweise keine ausreichenden Optionen zur Beschreibung von Geräten und Diensten. Entwickler sind an das Schema gebunden. Wie die Felder ursprünglich zustande kamen, ist nicht ersichtlich.
- Der Nachrichtenaufbau ist ebenfalls auf SOAP und WSDL gestützt und verliert auch durch die festen Definitionen nicht wesentlich an Komplexität.

Der Autor hält daher auch das Devices Profile for Web Services für ungeeignet.

2.2.8.5 Constrained Application Protocol (CoAP)

Zu beachten ist, daß das Constrained Application Protocol nur einen Ersatz für HTTP darstellt, der durch bereits besprochene Merkmale Energie und Übertragungszeit einspart. Für den alleinigen Einsatz im Smart Home ist das Protokoll dennoch nicht geeignet, da Merkmale zur Erkennung und Beschreibung von Diensten fehlen.

Die folgende Analyse diskutiert die Schwächen von CoAP durch einen Vergleich mit HTTP:

- *Transport über UDP.* Genau wie HTTP ist CoAP als Schicht-7-Protokoll spezifiziert, auf dem eine Anwendung, z.B. HTML-Webseiten, aufsetzt. Wie in [44, S. 10] angegeben, ist CoAP aber als Protokoll vorgesehen, das Nachrichten vorrangig über UDP austauscht. Eine Reihe von anderen Transportsystemen, die möglicherweise nicht auf Ethernet, IP oder UDP basieren, werden damit bereits zu Anfang ausgeschlossen, und das, obwohl sie im Bereich der ressourcenbeschränkten Geräte durchaus ihre Berechtigung haben: nicht jedes Endgerät wird in absehbarer Zeit wirtschaftlich Ethernet oder IP anbieten, Subnetze mit Gateways und proprietären

eigenen Transportsystemen werden weiterhin anzutreffen sein.

- *Zuverlässigkeit im Anwendungsprotokoll.* CoAP bildet durch seinen Mechanismus der bestätigten (ACK) Nachrichten und anderer Merkmale wesentliche Teile von TCP auf der Anwendungsschicht nach. Aus drei Gründen ist das wenig sinnvoll:

  1. TCP ist für eingebettete Systeme bereits seit Jahren implementiert [48].
  2. TCP kann potentiell Fragmente einer Nachricht wiederherstellen und muß nicht die gesamte Nachricht erneut übertragen.
  3. TCP besitzt einen Integritätscheck, während sich CoAP dafür auf die untergeordnete Schicht verläßt. Implementiert diese keine Integritätsprüfung, kann eine CoAP-Nachricht potentiell korrumpiert verarbeitet werden.

- *Multi- und Broadcast-Nachrichten.* Das Protokoll stützt sich auf die untergeordneten Schichten für Multi- und Broadcast-Adressierung. Wenn diese entsprechende Merkmale nicht bieten (z.B. eine serielle Schnittstelle), kann CoAP auf dem Nachrichtentransport nicht verwendet werden.

- *Flußkontrolle.* Im Gegensatz zu HTTP ist eine Flußkontrolle von Nachrichten vorgesehen. Dabei wird mißachtet, daß Verstopfungen auf einem Medium in allen beteiligten Schichten vorkommen können und eine Behandlung auf der jeweiligen Schicht getrennt vorzunehmen ist. Für die Anwendungsschicht muß der Nachrichtentransport vollkommen transparent ablaufen, sonst ist eine Schichtentrennung nicht mehr gegeben.

- *Unterstützung durch Client- und Serversoftware.* CoAP ist ein neues Protokoll und noch nicht vollständig spezifiziert. Eine Unterstützung in gängigen Programmiersprachen, die auch aktuell gehalten wird und mit jeder neuen Version eines Drafts aktualisiert wird, ist derzeit nicht erkennbar.

- *Kompression von Nutzdaten.* Bei HTTP können Nutzdaten nach gegenseitiger Vereinbarung über Header (`Accept`-Header) mittels verschiedener, von beiden Seiten unterstützter Algorithmen komprimiert werden. Im Gegensatz dazu bietet CoAP derzeit keine Möglichkeit der Aushandlung von Nutzdatenkompression, die Liste der in der `Content-Format`-Option möglichen Werte ist auf einige wenige limitiert, siehe Tabelle 2.4.

- *Mangelnde Permanenz der Codetabellen.* CoAP bedient sich zur Reduktion der Nachrichtengröße in weiten Teilen konstanten Codierungen mittels fest definierter Wörterbücher oder Codetabellen (z.B. zur Definition der HTTP-Methoden, des `Content-Type` und der gültigen Options). Selbst nach Finalisieren der Spezifikation müssen diese Codetabellen für neue Anforderungen erweitert werden. Entsprechend ist in diesem Falle die Firmware auf allen beteiligten Geräten anzupassen. Da sich gerade im Bereich der Smart Things die Anforderungen an die möglichen Nachrichteninhalte und auszuführenden Operationen nicht von vornherein festlegen lassen, stellt diese Eigenschaft von CoAP eine Einschränkung dar.

- *Erforderliche synchrone Uhrzeiten.* Manche Features von CoAP erfordern einen sekundengenauen Zeitstempel, etwa zum Berechnen der letzten Änderungszeit von Ressourcen für `if-match`, `if-non-match` und `max-age`-Optionen. Auf der designierten Zielhardware kann eine solche Uhrzeit nicht vorausgesetzt werden, da in diesem Falle Ressourcen zur Synchronisierung bereitgehalten werden müssen und die Einschränkung auf bestimmte Micro Controller Unit (MCU), eben jene mit einer Hardware-Realtime Clock (RTC) oder einem externen Modul, von einem Hersteller nicht verlangt werden kann.

- *Neukonzeptionierung von Application Level Gateways.* Eine weitere Schwierigkeit bei CoAP besteht in der Implementierung von Application Level Gateways. Ihre Regeln, die sich auf Verben, URI, Ports und Inhalte beziehen, müssen auf CoAP portiert werden, wobei wegen bestehender Differenzen die Regeln nicht einfach übernommen werden können. Insbesondere die Möglichkeit, asynchron Nachrichten vom Server zu einem späteren Zeitpunkt zum Client zu senden (bei HTTP über eine offene TCP-Verbindung), stellt ein großes Problem dar. Für eine zustandsgestützte Überwachung („stateful inspection“) muß das Gateway beide Kommunikationsrichtungen auf das korrekt gesetzte Token hin überwachen; es folgt daraus ein entsprechend größerer Bedarf an Rechenleistung.

- *Keine Angabe über Länge des Inhaltes („Payload“).* Das Protokoll begrenzt die mögliche Länge des Inhalts über die Paketgröße der Transportschicht. In der Spezifikation ist wiederum explizit von UDP die Rede. Nicht nur wird so die gebotene Schichtentrennung verletzt, sondern können auch stream-orientierte Transportlösungen („EIA-232“) nicht ohne weitere Adaption verwendet werden.

Selbst vor dem Hintergrund, daß CoAP noch nicht abschließend spezifiziert ist, läßt sich jetzt schon eine negative Abschätzung hinsichtlich seines Nutzens abgeben. Das Protokoll kann im Smart Home sein Pendant HTTP technisch zwar ersetzen, bietet aber aufgrund der beschriebenen Nachteile keine ausreichenden Anreize.

2.2.8.6 Constrained ReSTful Environments (CoRE) Link Format

Die Kombination aus CoAP und CoRE wurde insbesondere für das gesamte Internet of Things (IoT) entworfen. Eine Analyse, inwieweit es die Anforderungen dafür (die in der Arbeit nicht behandelt werden) erfüllen kann, würden den Umfang der Arbeit überschreiten. Für den Einsatz im Smart Home stellen sich zunächst konzeptionelle Fragen:

- *Externe Schema-Beschreibung.* Die Beschreibung des Maschineninterfaces ist, ähnlich wie bei SOAP und DPWS auf externe Ressourcen angewiesen. Gerade im Smart Home mit seiner begrenzten Rechenkapazität ist vollkommen unklar, wo diese Kapazitäten anzusiedeln sind und wie oft sie benötigt werden.

- *XML-Parsen auf Smart Devices.* Um die Maschinenschnittstelle verwenden zu können, müssen Clients unter Umständen eine WADL-Datei auswerten. Selbst wenn diese auf geeignete Weise zum Client gelangt, so kann eine Verarbeitung aufgrund der auch bei SOAP geltenden Beschränkungen nicht vorausgesetzt werden. Ressourcenschwache Geräte (z.B. Lichtschalter) können so unter Umständen nicht miteinander kommunizieren, sondern benötigten zwingend eine vermittelnde Instanz.

- *Integration mit DNS/IP.* Wohl mit Blick auf 6LoWPAN ist CoRE über CoAP eng mit IP-basierten Diensten wie DNS verwoben und unterstützt ausdrücklich das Verlinken und das Einbinden von externen HTTP-Ressourcen. Wie angeführt, kann auf Smart Devices keine Aussage über den Nachrichtentransport getroffen werden.

Auch in Protokoll und Nachrichtenformat finden sich Unklarheiten und Schwachstellen:

- *Einsprungpunkt* `/.well-known/core`. Weswegen gerade diese URI verwendet wird, ist nicht dargelegt. Etwa hätte ein kürzerer Punkt `/.core` die gleiche semantische Bedeutung.

- *Zweckentfremden der spitzen Klammern.* Gerade im Bereich der auf Standard Generalized Markup Language (SGML) basierenden MarkUp-Sprachen haben sich die bestimmte Zeichen zur Trennung von Bezeichnern und ihren Attributen und den respektiven Werten etabliert. Insbesondere bei relativen Pfaden mit nur einem Bestandteil (`</temperatur>`) besteht Verwechslungsgefahr mit schließenden Tags aus einer SGML-basierten Sprache. Entwickler, vor allem von Filtersoftware, müssen nun mehr Aufwand in die Robustheit ihres Codes investieren.

- *Suchfunktion im Smart Device implementiert.* Es ist nicht klar, warum eine Suche auf einem Smart Device realisiert werden soll. Zum einen ist sie für ressourcenbeschränkte Geräte laut Spezifikation optional. Wenn das der Fall ist und deswegen ein Client nicht mit einer gegebenen Funktionalität auf allen Geräten rechnen kann, ist diese von vorneherein wertlos. Zum zweiten ist – wie in allen Protokollen – keine Ontologie spezifiziert. Jeder Entwickler kann seine Ressourcen nach eigenen Kriterien benennen. Damit erübrigt sich eine Suche nach Diensten, da ihre Namen nicht vorab bekannt sind. Die Auswahl muß daher in jedem Fall auf dem Client stattfinden, der dafür *alle* vorhandenen Dienste aller Geräte ungefiltert benötigt.

Der beschriebenen Mängel wegen ist auch CoRE nicht geeignet.

2.2.8.7 Zusammenfassung

In Tabelle 2.6 sind die vorgestellten Protokolle gegenübergestellt. Zusammenfassend lassen sich in den beschriebenen Anwendungsprotokollen die folgenden Schwachpunkte aufführen:

Protokoll	Geräte			Dienste			Features		Typische Größe	
	Discovery	Description	Multicast-Adressierung	Discovery	Description	Multicast-Adressierung	Ereignisse	Parsen als Datenstrom	Nachricht	Code, ROM
HTTP	✗	✗	✗	✗	✗	✗	✗	✓	>20 B	<2 kB
XMLRPC	✗	✗	✗	✓	✓	✓	✗	✓	>200 B	>300 kB
SOAP	✗	✗	✓	✓	✓	✓	*nur P/S*	✗	>1000 B	*n/a*
DPWS	✓	✓	✓	✓	✓	✓	*nur P/S*	✗	>1000 B	>18 kB
CoAP	✗	✗	✓	✗	✗	✗	*nur P/S*	✓	>10 B	>10 kB
CoRE	✓	✓	✗	✓	✓	✗	✗	✓	>20 B	*n/a*

Tabelle 2.6: Abdeckung der Anforderungen des Smart Homes durch die vorgestellten Protokolle. Anmerkung: Nachrichten- und Code-Größe stellen nur eine qualitative Abschätzung dar. Sie hängen insbesondere von konkreten Nachrichteninhalten ab.

- *Komplexität von Nachrichten.* Alle untersuchten Protokolle unterstützen entweder durch explizite Spezifikation oder eben Fehlen einer solchen komplex aufgebaute Nachrichten. Insbesondere Protokolle, deren Nachrichten nicht als Datenstrom verarbeitet werden können, sind für eingebettete Systeme nicht geeignet.

- *Informationsgehalt, Redundanz und Nachrichtengröße.* Einige der Protokolle beschreiben in jeder übertragenen Nachricht ihr Schema erneut. Das ist allerdings nicht notwendig: es reicht, sich (auch dynamisch) zu Beginn der Kommunikation auf ein Schema zu einigen und dieses auf beiden Seiten zu speichern; danach können dem Schema entsprechende Nachrichten auch ohne dessen erneute Übertragung verarbeitet werden.

- *Codegröße und Speicherbedarf.* Komplexe Nachrichten erfordern aufgrund einer Vielzahl möglicher (und möglicherweise auch dynamisch definierter) Elemente entsprechende Unterstützung durch den Programmcode. Sie werden auf Wenn-Dann-Operationen abgebildet, wodurch Codegröße und Arbeitsspeicherbedarf entsprechend zunehmen. Gerade bei nicht-konstanten XML-basierten Nachrichten, die auf dem eingebetteten System zu verarbeiten sind, überschreitet der Bedarf an Ressourcen unter Umständen das Angebot.

- *Keine oder mangelnde Schichtentrennung.* Da die vorgestellten Protokolle auf IP und dessen Transportprotokolle aufsetzen, existiert keine oder nur unzureichende Schichtentrennung. Eine Interaktion von unterschiedlichen Lösungen und Produkten mehrerer Hersteller wird so erschwert und muß auch hier auf Anwendungsebene implementiert werden.

- *Fehlende Metainformationen.* Bis auf DPWS definiert kein Protokoll ein festes Format für Metainformationen für Gerätebeschreibungen. Bei Dienstebeschreibungen gilt das für alle Protokolle. Damit muß noch immer ein menschlicher Benutzer mit Hilfe

der statischen und nur „intelligent“ interpretierbaren und hoffentlich verständlichen Dokumentation entscheiden, welche Geräte und Dienste zueinander passen und wie sie zu verknüpfen sind.

Um die vorigen Ausführungen in Kontext zu setzen: die Argumentation ist weniger, daß die vorgestellten Protokolle insgesamt nicht funktionieren und gar keinen Nutzen im Smart Home haben. Auf entsprechend mächtigen, ressourcen-kräftigen ausgestatteten und mit IP-Fähigkeiten versehenen Plattformen sowie Nachrichtentransporten mit ausreichend hoher Datenrate (Ethernet, WiFi) können manche von ihnen durchaus erfolgreich eingesetzt werden.

Auch die Protokolle ohne feste Nachrichtentypen (HTTP, CoAP ohne CoRE) könnten zur Steuerung von Geräten prinzipiell verwendet werden, sofern sich beide Kommunikationspartner mit eigenen Definitionen auf ein Schema und eine für die menschlichen Benutzer verständliche Ontologie einigen.

Allerdings soll ein gewisser Fortschritt im Vergleich zum Status-Quo erzielt werden, daher folgt aus dieser abschließenden Diskussion die Aufgabe zu zeigen, daß sich die beschriebenen Anforderungen auch einfacher und für alle Zielgruppen vorteilhaft und nutzbringend umsetzen lassen, sowie daß mit nur wenig zusätzlichem Aufwand weitere Features implementiert werden können. Dies erfolgt in den folgenden beiden Kapiteln 3 und 4.

3 Anwendungsebene: Ein generisches Protokoll

Im vorausgegangen Kapitel konnte gezeigt werden, daß aktuelle Steuerungsprodukte aufgrund ihrer Spezifikationen nicht oder nur kaum für die Steuerung von Haushaltsgeräten geeignet sind. Untersuchte reine Anwendungsprotokolle könnten die Lücke teilweise füllen, stehen aber den Anforderungen für eine möglichst gering ausgestattete Hardware und einen möglichst geringen Preis entgegen.

Beiden Punkten wird in diesem Kapitel begegnet, indem ein universelles, erweiterbares Anwendungsprotokoll schrittweise entworfen wird, das die bislang besprochenen Anforderungen umsetzt und mit vergleichsweise wenig leistungsfähiger Hardware auskommt. Gleichzeitig wird es sich soweit als möglich an offenen und etablierten Protokollen anlehnen, um Herstellern, Entwicklern und Anwendern den Einstieg einfach zu ermöglichen.

Das Anwendungsszenario der Integration in ein Smart Grid und die dafür erforderlichen Funktionen werden abschließend unter Angabe von konkreten Beispielen behandelt und berücksichtigen dabei auch Überlegungen zur IT-Sicherheit rein auf Anwendungsebene.

Zur Abgrenzung zu bestehenden Protokollen trägt das entworfene Protokoll den Namen Smart Home Device Control Protocol (SHDP).

3.1 Ziele und Anforderungen an das Anwendungsprotokoll

Einleitend werden konkrete Ziele und Aufgaben des zu erstellenden Protokolls benannt.

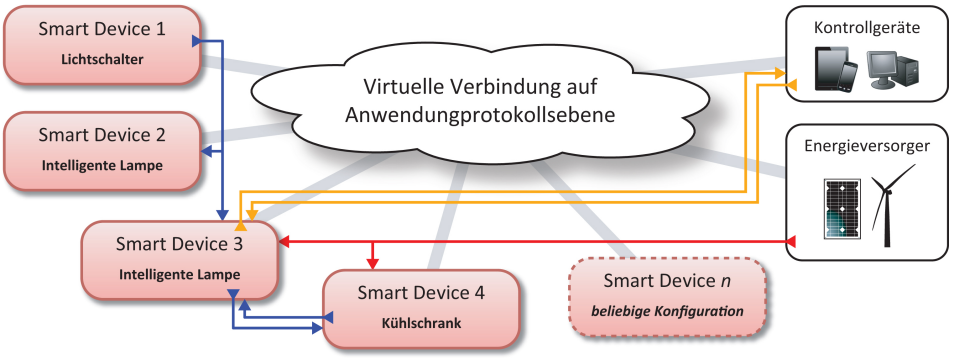


Abbildung 3.1: Darstellung der angestrebten Interaktion zwischen Smart Devices und Kontrollgeräten bzw. dem Energieversorgungsunternehmen. Die Anwendungsprotokollebene geht von Broadcast-Kommunikation aus.

Zunächst bietet dazu Abbildung 3.1 einen Überblick über die Verbindung rein auf Anwendungsprotokollebene und zeigt mögliche Kommunikationswege auf:

- Smart Devices kommunizieren untereinander ohne Mitwirken einer vermittelnden Instanz und ohne Berücksichtigung von konkretem physischen Nachrichtentransport oder zwischenliegenden Transportprotokollen. Die Kommunikation kann dabei durch externe Ereignisse (z.B. Drücken eines Schalters, Änderung eines Meßwerts) ausgelöst werden oder auch periodisch erfolgen. Im Bild ist dies mit blauen Pfeilen repräsentiert.
- Smart Devices empfangen Befehle von Steuergeräten und beantworten Anfragen nach Maschinenzuständen. Im Bild ist dies mit gelben Pfeilen repräsentiert.
- Externe Parteien, wie etwa Energieversorgungsunternehmen, erhalten Zugriff auf bestimmte Funktionen und Dienste, um Notfälle behandeln zu können oder dem Heimbesitzer einen Nutzen zu bieten. Im Bild ist dies mit roten Pfeilen repräsentiert.

Wie dargestellt sind keine Aussagen über Verbindungsarten, Protokolle, Datenraten, Latenzzeiten oder andere Parameter des Nachrichtentransports getroffen. Es geht auf der Anwendungsschicht rein um Funktionen, die die Smart Devices besitzen und anderen Geräten, bzw. dem Benutzer, zur Verfügung stellen. Auch eine Zugriffskontrolle im Sinne der IT-Sicherheit ist zunächst nicht von Bedeutung.

Einige Anforderungen an die *Funktionen* des Anwendungsprotokoll wurden im Zuge der Evaluierung vorhandener Protokolle in Kapitel 2.2.8 bereits ermittelt. Die folgenden Punkte ergänzen die dortigen Ausführungen um konkrete ausführliche Beschreibungen:

- *Entdecken von Geräten.* Innerhalb des lokalen Netzwerkes können Instanzen, z.B. Kontrollgeräte, mit dem Netz verbundene Geräte erkennen. Da sich die Menge der Geräte wahrscheinlich nur selten ändert, wird diese Funktion nicht oft ausgeführt.
- *Beschreiben von Geräten.* Das Protokoll bietet die Funktion, zum einen für Menschen wichtige, beschreibende Informationen („Name“, „Beschreibung“, „Modell“, „Hersteller“, etc.) über ein Gerät zur Verfügung zu stellen. Gleiches gilt für Informationen, die für andere Geräte wichtig sind, was beispielsweise eine eineindeutige Geräteadresse innerhalb des Netzwerksegments umfaßt.
- *Auflisten und Beschreiben von Diensten auf Geräten.* Genau wie das Entdecken von Geräten werden auch die jeweils angebotenen Funktionen in einem geeigneten Format ausgegeben. Nicht nur ein Mensch muß in der Lage sein, die Beschreibung zu interpretieren, sie muß gleichzeitig auch für andere Maschinen lesbare Information enthalten.
- *Nutzen von Diensten.* Die gelisteten Dienste müssen durch andere Geräte nutzbar sein. Das Protokoll bietet somit einen standardisierten Weg, aus der Beschreibung von Diensten die Methode zum Benutzen desselben abzuleiten („request/response“). Bekannt ist das Vorgehen aus dem Internet, bei dem der Benutzer eine Anfrage an einen Server stellt und eine Webseite ausgeliefert erhält.

- *Ereignisbasierte Nachrichten.* Zusätzlich sollen Nachrichten unterstützt werden, die keiner vorigen Anfrage bedürfen, sondern bei Eintreten eines Ereignisses versendet werden. Quellen hierfür sind vielfältig und werden vom Entwickler definiert (z.B. ein abgelaufener Timer, ein gedrückter Taster oder ein geänderter Temperaturwert). Dies schließt insbesondere wiederholte Ereignisse, z.B. durch langes Drücken eines Schalters oder durch Fortbestehen einer Bedingung, ein.

- *Verknüpfen von Smart Devices.* Um die ereignisbasierte Kommunikation nutzen zu können, werden Smart Devices miteinander verknüpft. Dies kann durch entsprechende Benutzerschnittstellen („Anlernknopf“) erfolgen, soll aber auch durch das Protokoll selbst unterstützt werden.

Auch an das *Format* des Anwendungsprotokolls werden Anforderungen gestellt:

- *Weitestgehend basierend auf offenen und bekannten Protokollen oder Standards.* Zu vermeiden ist nach Möglichkeit die Zahlung von Lizenzkosten jeglicher Art für die Nutzung. Ein offenes Projekt wird dank Beteiligung vieler Interessierter, sowohl aus dem privaten wie auch dem kommerziellen Umfeld, stets bereichert.

- *Broadcast-Kommunikation.* Weiterhin ist eine Unterstützung von Broadcast-Nachrichtentransporten vorzusehen, da darüber keine einschränkenden Aussagen getroffen werden (siehe auch nächstes Unterkapitel).

- *Skalierbarkeit.* Das Anwendungsprotokoll unterstützt eine beliebige Anzahl von Geräten in einem Netzwerksegment. Dies wird außerdem durch geeignete Wahl des Nachrichtentransports mit entsprechend großer Datenübertragungsrate unterstützt.

- *Möglichst geringer Speicherbedarf der Implementierung.* Mit dem Fokus auf kleine Mikrocontroller ist auch diese Anforderung nachvollziehbar, da hier nur wenig Speicherplatz verfügbar ist. Mit einer möglichen Implementierung beschäftigt sich Kapitel 4.

- *Möglichst geringe Nachrichtengröße.* Um Energie bei Transport und Verarbeitung von Nachrichten zu minimieren, sowie Verarbeitungszeiten kleinzuhalten, ist das Format für eine möglichst geringe Nachrichtengröße auszulegen. In Kapitel 5 wird es gegen bestehende Protokolle und deren typische Nachrichtengrößen qualitativ evaluiert.

3.2 Abgeleitete Anforderungen an den Nachrichtentransport

Das Anwendungsprotokoll stellt den Nutzdatenanteil des Nachrichtentransports dar. Er ist auf geeigneten Wegen zu den jeweiligen Geräten zu befördern. Basierend auf den im Kapitel 2 untersuchten Nachrichtentransportlösungen werden Anforderungen an den Nachrichtentransport für SHDP abgeleitet.

3.2.1 Physikalische Verbindung

Aus den in Kapitel 2.1 untersuchten Produkten wird für die physikalische Verbindung der Schluß gezogen, daß es innerhalb des Smart Homes nicht *den* Standard zur physikalischen Ankoppelung von Geräten und zum Transport von Nutzdaten gibt oder geben wird:

- *Unterschiedliche Energieversorgung.* Smart Devices haben unterschiedliche Voraussetzungen, was die Energieversorgung betrifft. Geräte wie Kühlschränke, Öfen und Leuchten sind stets mit dem Stromnetz verbunden und gelten als „stets empfangsbereit“ („always on“). Kleine batteriebetriebene Geräte wie Tür- und Fenstersensoren benötigen dagegen möglichst stromsparende Übertragungswege.
- *Kosten.* Mit der Nutzung eines jeden standardisierten untersuchten Übertragungsweges sind bestimmte Kosten (Lizenzierung, Mitgliedschaften, Zertifizierung, etc.) verbunden. Hersteller werden im Zuge der Gewinnmaximierung genau den Weg wählen, der ihnen am günstigsten erscheint. Das schließt ausdrücklich preiswerte proprietäre Lösungen, wie im Beispiel HomeMatic, ein.
- *Medienwahl.* Ein Hersteller kann zwischen verschiedenen Medien (Funk, diverse bedrahtete Lösungen) wählen. Von den untersuchten Standards deckt nur der geschlossene und proprietäre Standard KNX diese Medien ab.

Daher werden für die zu konzipierende Lösung nur unscharfe Anforderungen an den Nachrichtentransport formuliert:

- *Broadcast-Transport.* Jedes Smart Device kann mit jedem anderen Gerät des Segments kommunizieren, schematisch bereits in Abbildung 1.2 und 3.1 dargestellt. Eine Adressierung auf Geräte-Ebene (MAC-Adressen) ist nicht erforderlich.
- *Kein Routing erforderlich.* Ein Netzwerksegment für Smart Devices wird als Single-Hop-Netzwerk realisiert. Bei größerer Ausdehnung sind entsprechend transparent arbeitende Repeater oder Segment-Koppler einzusetzen, deren Spezifikation allerdings kein Teil der Arbeit ist.
- *Medienzugriff.* Zumindest muß eine Kollisionsvermeidung, angepaßt auf das jeweilige Medium z.B. durch Belegungserkennung, vorhanden sein. In manchen Implementierungen läßt sich dies aufgrund von Ressourcenbeschränkungen nicht umsetzen, Kollisionen bleiben damit möglich.
- *Halbduplex.* Um die Implementierung zu vereinfachen, wird von einem Betrieb im Halbduplexverfahren ausgegangen. Kabelgebundene und funkbasierte Lösungen, die nur einen Kanal zur Verfügung haben, können somit einfacher verwendet werden.
- *Ausreichend hohe Datenrate.* Für das gewählte Netzwerk- und Anwendungsprotokoll muß das Übertragungsmedium eine ausreichende Kanalkapazität und letztlich auch Brutto-Datenrate bieten. „Ausreichend“ kann aufgrund der nicht bestimmten Netzwerk- und Anwendungsprotokolle an dieser Stelle allerdings nicht qualifiziert werden, insbesondere, als daß auch eine Reaktionszeit des Smart Devices eine Rolle spielt.

Für eine Abschätzung derzeitig verwendeter Datenraten bieten die Tabellen 2.6 und 2.2 eine Übersicht.

Zusammenfassend ist die Spezifikation des physikalischen Transports kein Teil der Arbeit. Allerdings wird eine beispielhafte Implementierung auf Basis proprietärer Funkmodule in Kapitel 4 ausgeführt.

3.2.2 Netzwerkprotokoll

Analog zu Abbildung 1.2 umfaßt das Netzwerkprotokoll die OSI-Schichten 2 mit 4 und übernimmt die folgenden Aufgaben:

- *Broadcast-Übertragungen.* Die Netzwerkschicht muß einzig den Broadcast-Transport des Mediums unterstützen. Eine Adressierung durch Transport-Adressen wird nicht vorausgesetzt.
- *Optional: Zuverlässigkeit.* Auch bei Kollisionsvermeidung kann durch widrige Einflüsse auf den Nachrichtentransport eine Nachricht bei einigen Teilnehmern eines Segments verloren gehen. In diesem Falle ist vom Nachrichtentransport eine Wiederholung der Übertragung einzuleiten. Eine mögliche technische Lösung, die auf negativen Bestätigungen – engl. Non-Acknowledge (NACK) – beruht, wird in Kapitel 4.4 beschrieben.

Aus beiden vorigen Auflistungen ergibt sich, daß die Adressierung von Smart Devices nicht zwingend Teil des Netzwerkprotokolls sein muß. Eine Adressierung auf Anwendungsprotokollebene erscheint ausreichend.

Zu gängigen Transportimplementierungen, insbesondere 6LoWPAN, sei angemerkt, daß die Aufgaben eines Smart Devices nicht im Routen von Internetverkehr liegen werden. Ein komplexer Standard erscheint deswegen fragwürdig für den Einsatz im Smart Home.

3.3 Steuerung von Geräten

Wird die Kommunikation aus Sicht eines Smart Devices betrachtet, so liegt das „Interesse“ im Ausführen seiner designierten Funktionen und im Berichten und Kommunizieren seines Zustands. Dafür reicht schon ein vergleichsweise einfaches Protokoll aus, das nun schrittweise entworfen wird.

Zur Steuerung von Smart Devices setzt SHDP eine HTTP-Schnittstelle ein, welche in diesem Kapitel beschrieben ist. Zunächst werden Gedanken zur Eignung von Representational State Transfer (ReST) und HTTP zur Verwendung bei Smart Devices notiert.

Auf die danach gegebenen detaillierten Ausführungen zur Befehlsschnittstelle folgen Überlegungen und Angaben zu Fehlerwahrscheinlichkeit und -behandlung rein auf Anwendungsebene.

3.3.1 Analyse der HTTP-Header auf Notwendigkeit

Betrachtet man eine typische, von einem aktuellen Webbrowser abgesetzte HTTP-Anfrage, so wird neben Methode, Ressourcenidentifikation und optionalem Nutzdateninhalt auch eine Reihe von Header-Feldern übertragen (siehe Listing 3.1).

```
>>> GET /ressource HTTP/1.1
>>> Host: smartdevice01
>>> Connection: keep-alive
>>> Accept: text/html,application/xhtml+xml,application/xml;q=0.9
>>> User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (
    KHTML, like Gecko) Chrome/28.0.1500.95 Safari/537.36
>>> DNT: 1
>>> Accept-Encoding: gzip,deflate,sdch
>>> Accept-Language: de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4
```

Listing 3.1: Abbildung einer HTTP-Anfrage von Chrome 28. Laut Spezifikation ist für einen HTTP 1.1 Request ohne Nutzdateninhalt nur der `Host`-Header erforderlich. Alle anderen Felder sind *optional*. Die Gesamtgröße der Anfrage beträgt 355 B.

Die Größe der Headerfelder (Schlüssel und Werte) kann in einem Smart Device schnell den eigentlichen Nutzdateninhalt von möglicherweise nur einigen Byte überschreiten. Zu prüfen ist daher, ob und welche Headerfelder – abhängig von den Funktionen eines Smart Device – überhaupt benötigt werden.

Als *Funktion* eines Smart Devices sei in dieser Arbeit das Lesen oder Schreiben eines einzelnen Werts – auch als *Datenpunkt* bezeichnet – des Geräts festgelegt. Dafür werden die meisten der Headerfelder von HTTP nicht benötigt. Eine Verarbeitung der Methode, der Ressource und des Nutzdateninhaltes erscheint für die Anfrage ausreichend, weshalb nur zwei Felder überhaupt berücksichtigt werden und die folgenden Bedeutungen innehaben:

- `Host` Gibt die Adresse eines Smart Devices *auf Anwendungsebene* an.
- `Content-Length` Sofern innerhalb der HTTP-Anfrage Nutzdaten („body“) zu übertragen sind, wird die Länge in Byte hiermit übertragen.

In Listing 3.2 ist eine zu 3.1 äquivalente Darstellung gegeben, in der nur die laut HTTP-Spezifikation *erforderlichen* Headerfelder verwendet werden. Sie hätte exakt die selbe Ausgabe der Ressource `/ressource` zur Folge.

```
>>> GET /ressource HTTP/1.1
>>> Host: smartdevice01
```

Listing 3.2: Abbildung der validen, reduzierten HTTP-Anfrage. Die Gesamtgröße der Anfrage beträgt inklusive Zeilenumbruch `\r\n` nur noch 44 B und hat, zumindest auf dem angegebenen Dienst, die gleiche Ausgabe zur Folge wie der Request aus Listing 3.1.

So kommt auch HTTP mit geringen Nachrichtengrößen aus, ohne an Funktionalität zu verlieren. Es wird nicht in Abrede gestellt, daß die zusätzlichen Header in Anfragen einen Nutzen zur Vermeidung überflüssiger Übertragung unveränderter Nutzdaten bieten. Bezweifelt werden darf allerdings, daß dieser Nutzen auch bei Smart Devices mit wenigen Bytes an Nutzdaten gegeben ist.

3.3.2 Adressierung von Geräten

Bei HTTP kommunizieren zwei Geräte über eine Punkt-zu-Punkt-(Unicast-)Verbindung miteinander. Dazu verwendet der Client den `Host`-Header zur Bezeichnung des gewünschten Kommunikationspartners („Server“, „Host“) auf der Gegenseite. So können mehrere (virtuelle) Hosts auf einer einzigen öffentlichen Netzwerk-Adresse angeboten werden. Insgesamt war dieser Schritt wegen des stetig schrumpfenden Vorrats an öffentlichen IP-Adressen notwendig geworden.

Da eine Nachricht im Smart Home wegen des Broadcast-Transports stets alle Geräte eines logischen Segments erreicht, muß eine Lösung für die Adressierung auf Anwendungsprotokollebene gefunden werden, die die folgenden Adressierungsarten ermöglicht:

- *Unicast.* Ein einzelnes Gerät wird adressiert.
- *Multicast.* Ein Liste von benannten Geräten wird adressiert.
- *Broadcast.* Alle Gerät im Segment werden adressiert.

3.3.2.1 Adressierung von Geräten mit `Host`-Header

Bei Anwendungen im Internet ist der `Host`-Header bislang nur mit singulären Werten besetzt. Eine Multi- oder Broadcast-Adressierung wird nicht verwendet und wäre aufgrund der Punkt-zu-Punkt-Verbindung bei derzeitigen Anwendungen auch nicht sinnvoll.

Der Standard gestattet allerdings explizit ein mehrfaches Setzen eines Headers mit unterschiedlichen Werten. Sie sind entweder in einer Zeile durch Kommata getrennt anzugeben oder als Schlüssel/Werte-Paare auf mehrere Zeilen zu verteilen, siehe Listing 3.3.

```
1 >>> GET /temperatur HTTP/1.1
2 >>> Host: smart_device_1
3 >>> Host: kuehlschrank_1602
4 >>> Host: raumthermostat_fe4ecc03
```

Listing 3.3: Mehrfaches Setzen des `Host`-Headers bei einer HTTP-Anfrage. Obwohl dies für Server im Internet nicht praktikabel ist, kann diese Methodik im Smart Home für das Adressieren mehrerer Geräte auf Anwendungsprotokollschicht verwendet werden. Es wird von einer eineindeutigen Adresse („Hostname“) für jedes Smart Device ausgegangen.

Damit ergeben sich die folgenden Möglichkeiten:

- Die *Unicast*-Adressierung wird durch einen einzelnen `Host`-Header und die Geräte-ID des jeweiligen Smart Devices realisiert.
- Die *Multicast*-Adressierung wird durch mehrere Werte für den `Host`-Header realisiert.
- Die *Broadcast*-Adressierung wird durch einen einzelnen `Host`-Header mit dem reservierten Wert des Asterisk (`*`) realisiert.

Im Falle einer Multi- und Broadcast-*Adressierung* werden keine Fehlermeldungen generiert. Nur Bestätigungen über erfolgreiche Operationen (später in Kapitel 3.3.4 beschrieben) werden zurückgegeben. Die Benennung von Geräten, in Listing 3.3 bereits angedeutet, wird im Kapitel 3.4 ausführlich behandelt.

3.3.2.2 Verwendung des URL-Pfads

Neben der Adressierung über den `Host`-Header können Smart Devices auch über den URL-Pfad adressiert werden (siehe Listing 3.4). Unabhängig von eventuell angesprochenen Ressourcen ist das erste Segment des Pfades stets als Identifikator des angesprochenen Smart Devices zu betrachten, wenn *kein* `Host`-Header gesetzt ist. Der Fall wird in der Implementierung separat berücksichtigt.

```
1 >>> GET /smart_device_1/temperatur HTTP/1.1
```

Listing 3.4: Adressieren eines Smart Devices über den URL-Pfad. Ein `Host`-Header darf in diesem Falle nicht gesetzt werden, sonst würde der erste Teil des Pfades bereits als Teil einer Ressourcenbeschreibung interpretiert.

Durch diese Möglichkeit kann die Nachrichtengröße weiter reduziert werden, wenn nur *ein* oder *alle* Smart Devices angesprochen werden, da die zusätzlichen Bytes für den `Host`-Header entfallen.

Obwohl eine derartige Methodik bei Anwendungen im Internet mit Punkt-zu-Punkt-Verbindungen nicht sinnvoll erscheint, konnte so gezeigt werden, daß sich HTTP für Uni-, Multi- und Broadcast-Kommunikation verwenden läßt, ohne die Spezifikation zu verletzen, und daß es den Teil der Anforderung erfüllt.

3.3.3 Verfügbare Methoden

Für SHDP werden fünf Verben definiert. Die aus HTTP bekannten Verben `GET` und `POST` werden zum Abfragen und Verändern von Maschinenzuständen verwendet. Desweiteren werden die Methoden `EVENT`, `CONNECT` und `DISCONNECT` für ereignisbasierte Kommunikation eingeführt. Alle Verben werden im Folgenden näher beschrieben.

3.3.3.1 `GET`-Methode

Zur Abfrage von Zuständen in Smart Devices wird die Methode `GET` verwendet. Ein Aufruf enthält keine Nutzdaten und dementsprechend keinen `Content-Length`-Header. Zur Adressierung kann der `Host`-Header eingesetzt werden. Alternativ wird rein im URL-Pfad der Empfänger angegeben.

```
>>> GET /kuehlschrank_1602/*/temperatur HTTP/1.1

<<< HTTP/1.1 200 OK
<<<
<<< [Inhalt für Gefrierfach]
<<< [Inhalt für Kühlfach]
```

Listing 3.5: Veranschaulichen der Abfrage von mehreren Diensten auf einem Smart Device mit nur einer Anfrage. Der Asterisk kann ein vollständiges Segment des URL-Pfades ersetzen. Andere Verwendungen, insbesondere Teilersetzungen, sind derzeit nicht vorgesehen.

Auf Smart Devices sind Ressourcen in einer Baumstruktur angelegt, deren genauer Aufbau in Kapitel 3.4 beschrieben wird. Die Baumstruktur wird auf den URL-Pfad abgebildet. Um nur Teilbäume aus der Struktur auszuwählen, was auch für die folgenden Verben gilt, kann der Asterisk anstatt eines Bezeichners verwendet werden.

Listing 3.5 demonstriert dies anhand eines Beispiels. Die Implementierung adressiert damit *alle* angesprochenen Teile einer Ressource.

Unzulässig dagegen ist die Verwendung des Asterisk zur Teiladressierung von Ressourcen, wie in Listing 3.6 dargestellt.

```
>>> GET /kuehlschrank_1602/*/temp* HTTP/1.1

<<< HTTP/1.1 400 Bad Request
<<<
<<< [Fehlermeldung: Asterisk darf nicht als Teilstring-Ersatz verwendet
    werden.]
```

Listing 3.6: Da der Asterisk in dieser Version nicht für eine Teilersetzung verwendet werden darf, ist die dargestellte Anfrage ungültig.

3.3.3.2 POST-Methode

Mit Hilfe der POST-Methode werden Maschinenzustände gesetzt, sofern für eine Ressource zulässig. Listing 3.7 demonstriert das Setzen einer einzigen Ressource.

```
>>> POST /kuehlschrank_1602/gefrierfach/soll_temperatur HTTP/1.1
>>> Content-Length: 5
>>>
>>> -18.0

>>> POST /*/gefrierfach/soll_temperatur HTTP/1.1
>>> Content-Length: 5
>>>
>>> -18.0
```

Listing 3.7: Gültige POST-Anfragen im Anwendungsprotokoll. Der `Content-Length`-Header ist notwendig, um dem Server das Ende einer Nachricht anzuzeigen. Der Asterisk als Platzhalter für „alle“ Instanzen einer Ressource verwendet werden.

Sollen auf einem Gerät mehrere Werte gleichzeitig geschrieben werden, so können sie per Service-Multicast angesprochen werden. Das Vorgehen ähnelt dem bekannten Übertragen mehrerer Variablen bei HTML-Eingabeformularen und ist in Listing 3.8 dargestellt.

```
>>> POST /kuehlschrank_1602 HTTP/1.1
>>> Content-Length: 63
>>>
>>> gefrierfach.soll_temperatur=-18.0&kuehlfach.soll_temperatur=8.0
```

Listing 3.8: Gültige POST-Anfrage zum Setzen mehrerer Werte im Anwendungsprotokoll. Der Content-Length-Header ist notwendig, um dem Server das Ende einer Nachricht anzuzeigen. Ressourcen-Teile werden durch Punkt statt Vorwärts-Slash getrennt. Mehrere Ressourcen können adressiert werden; sie werden mittels Ampersand kombiniert. Der Mechanismus ist aus HTML-Formularen bekannt. Sonderzeichen in Werten sind geeignet zu codieren („URL-Encoding“).

3.3.3.3 EVENT-Methode

SHDP unterstützt eine ereignisbasierte Kommunikation, die nicht aus Anfrage/Antwort-Tupeln besteht, sondern nur von Smart Devices initiierte Nachrichten enthält [49]. Im Falle eines Ereignisses, z.B. der Änderung des Wertes einer Ressource, kann ein Gerät dies per Broadcast im Netzwerksegment verteilen. Das Protokoll an sich macht keine Angaben über die Definition des Ereignisses oder die Häufigkeit einer Meldung. Letzteres ist alleine Aufgabe der Anwendung auf einem Smart Device und hängt von Implementierung und Anforderungen ab.

Das Nachrichtenformat einer Ereignismitteilung orientiert sich am HTTP-Standard. Zur Kennzeichnung einer Ereignisnachricht wird die die Methode EVENT eingeführt. Der URL-Pfad gibt hier allerdings nicht mehr das *Ziel* einer Anfrage an (wie in Listing 3.4), sondern den vollständigen Pfad zur *Quelle*. Grund ist, daß laut HTTP-Spezifikation der URL-Pfad die von der Methode betroffene Ressource bezeichnet. Das ist hier der Fall, denn das Ereignis *betrifft* die angegebene Ressource.

EVENT-Nachrichten werden nicht bestätigt. Analog wird kein Host-Header gesetzt, da das Ziel der Nachricht nicht bekannt ist und eine Host: *-Adressierung redundant wäre. Daher muß ein Client, der ein bestimmtes Event erwartet, bei Ausbleiben eines solchen eine Fehlerbehandlung auf Anwendungsebene implementieren. Verglichen mit anderen Protokollen ist dies kein Rückschritt: auch hier muß letztlich die Anwendung im Fehlerfalle der untergeordneten Netzwerkschichten die Fehlerbehandlung übernehmen.

Im Nutzdatenfeld der Nachricht wird der neue Wert angegeben. Listing 3.9 zeigt eine vollständige EVENT-Nachricht.

```
>>> EVENT /smart_device_1/temperatur HTTP/1.1
>>> Content-Length: 4
>>>
>>> 19.5
```

Listing 3.9: Vollständige EVENT-Nachricht. Die Nutzdaten werden in Kapitel 3.4 besprochen.

Eine Event-Nachricht betrifft stets nur eine einzige Ressource. Der Asterisk oder mehrere Quellen, dargestellt in Listing 3.10, sind in einer Nachricht nicht zulässig.

```
>>> EVENT /kuehlschrank_1602 HTTP/1.1
>>> Content-Length: 53
>>>
>>> gefrierfach.temperatur=-18.2&kuehlfach.temperatur=8.4
```

Listing 3.10: Unzulässige Verwendung der EVENT-Nachricht. Weder der Asterisk noch mehrere Ressourcen-Quellangaben sind in einer Nachricht erlaubt.

Letztlich wird lediglich die Empfehlung abgegeben, mit EVENT-Nachrichten sparsam umzugehen. Binäre Protokolle eigenen sich eher für das Live-Streaming von Werten als HTTP, da ihr Overhead in der Regel geringer ist.

3.3.3.4 CONNECT-Methode

Mit Hilfe der CONNECT-Methode werden direkte Verknüpfungen zwischen Smart Devices eingerichtet. Eine oder mehrere Ressourcen auf einem Smart Device werden für die Benachrichtigung durch eine oder mehrere EVENT-Quellen konfiguriert.

Die *Ziel-Ressourcen* werden im URL-Pfad spezifiziert, wobei eine Mehrfachadressierung durch das das Asterisk-Zeichen zulässig ist. Die *Quell-Events* werden im Nutzdateninhalt der Nachricht angegeben, siehe Listing 3.11.

```
>>> CONNECT /lichtcontroller_0e3f88a9/kanal1/wechsel_eingang HTTP/1.1
>>> Content-Length: 130
>>>
>>> /lichtschalter_16883421/linker_taster/kurzer_druck
>>> /lichtschalter_1938482a/taster/kurzer_druck
>>> /lightswitch_afc2/top_key/pressed

<<< HTTP/1.1 200 OK
<<<
<<< [Antwort von lichtcontroller_0e3f88a9: Drei Event-Quellen mit der
    Ressource verknüpft.]
```

Listing 3.11: Mehrfache Event-Quellen in einer CONNECT-Nachricht. Die erfolgreich registrierten Quellen werden an den Client zurückgemeldet. Die in eckigen Klammern dargestellten Inhalte sind Teil eines späteren Kapitels.

Jede Zeile, getrennt durch Wagenrücklauf- und Zeilenvorschub-Zeichen (`\r\n`), enthält eine einzige Quelle. Es wird an dieser Stelle keine Aussage über die maximal mögliche Anzahl an Quellen für eine Ressource getroffen. Das hängt vom verfügbaren Speicher auf der Hardwareplattform des Smart Devices und der Implementierung ab.

Sofern die Anzahl der freien Event-Speicherplätze einer Ressource nicht ausreicht, oder die Länge einer Event-Quelle zu groß ist, wird eine Fehlermeldung mit HTTP-Status `413 Request Entity too long` und einer textuellen Beschreibung des Fehlers zurückgeliefert, siehe Kapitel 3.3.4.

Die beschriebene Funktionalität kann auch durch reine Hardware-Benutzerschnittstellen realisiert werden. So könnte ein Gerät etwa durch einen dedizierten Knopf in einen „Anlernmodus“ versetzt werden, während danach die entsprechende Quelle eine EVENT-Nachricht erzeugt. In gängigen Heimsteuerungsprodukten ist eine solche Funktion realisiert.

3.3.3.5 DISCONNECT-Methode

Um eine hergestellte Verknüpfung wieder von einem Gerät zu lösen, dient die Methode DISCONNECT. Analog zu CONNECT können mehrere Ressourcen gleichzeitig entfernt werden, siehe Listing 3.12. Als Antwort werden die nach der Operation bestehenden Verknüpfungen über ein bestimmtes XML-Schema (siehe Kapitel 3.4) an den Client zurückgemeldet. Die Anwendungslogik kann daraufhin ggf. eine Fehlerbehandlung durchführen.

```
>>> DISCONNECT /*/kanal1/wechsel_eingang HTTP/1.1
>>> Content-Length: 85
>>>
>>> /lichtschalter_16883421/linker_taster/kurzer_druck
>>> /lightswitch_afc2/top_key/pressed

<<< HTTP/1.1 200 OK
<<<
<<< [Antwort von lichtcontroller_0e3f88a9: Nur noch eine Event-Quelle mit
    der Ressource verknüpft.]
```

Listing 3.12: Mehrfaches Entfernen von Event-Quellen aus allen verbundenen Smart Devices, die einen Dienst `kanal1/wechsel_eingang` besitzen, mit einer DISCONNECT-Nachricht. Jedes adressierte Smart Device, das mindestens eine Ressource entfernen konnte, meldet die noch bestehenden Verknüpfungen über eine speziell formatierte Antwort an den Client zurück. Smart Devices, die zwar angesprochen wurden, aber keinen entsprechenden Dienst anbieten, ignorieren die Nachricht. In eckigen Klammern dargestellte Inhalte werden später erläutert.

3.3.4 Rückgabewerte über HTTP-Statuscodes

Anfragen an Smart Devices werden bei SHDP konform mit HTTP-Antworten und entsprechenden Statuscodes beantwortet, sofern das Device adressiert war.

3.3.4.1 Multi- und Broadcast-Kommunikation

Hierbei gilt, daß auf Multi- und Broadcast-Anfragen nur positive Bestätigungen (`HTTP/1.1 200 OK`) erfolgen, wenn ein Gerät eine Anfrage erfolgreich bearbeiten konnte.

Jegliche Fehler werden still ignoriert. Es ist Aufgabe der Anwendung, einen Fehler durch Ausbleiben einer positiven Antwort festzustellen und entsprechend zu reagieren, beispielsweise durch einen erneuten, per Unicast kommunizierten Versuch.

Da das Protokoll, wie bereits angedeutet, auch Dienste per Broadcast ansprechen kann, ist eine solche Logik hier notwendig:

- Wären etwa bestimmte Dienste nur auf einem Bruchteil der angeschlossenen Geräte verfügbar, so würden die Fehlermeldungen der zwar angesprochenen, aber nicht zur Beantwortung fähigen Smart Devices die Anzahl der positiven Rückmeldungen überschreiten. Die Information ist für die Anwendung aber nutzlos, da diese keine Anfrage an die betroffenen Geräte direkt stellte.

- Auf Transportprotokollebene kann nicht zwischen Geräten unterschieden werden (Broadcast-Transport!). Daher wird dies auf Anwendungsprotokollebene ermöglicht.

Die Lösung liegt in einer von allen Geräten in Reichweite nacheinander erzeugten, virtuellen HTTP-Antwort. Sie enthält nur einen einzigen Status, siehe Listing 3.13.

Das erste dem Client antwortende Smart Device erstellt einen passenden HTTP-Header. Es hat allerdings hat kein Wissen über die folgenden Antworten und kann keinen passenden `Content-Length`-Header erzeugen. Real bedeutet das keine Einschränkung, da ein fehlender Längenheader vom Client automatisch durch entsprechende Timer abgefangen wird. Nach dem letzten empfangenen Zeichen wird dieser aktiviert bzw. zurückgesetzt und löst mit Ablauf das Ende des Empfangs aus. Die Spezifikation von HTTP wird hierdurch nicht verletzt, allerdings muß die Anwendung robust gegen fehlerhafte und unvollständige Nutzdaten programmiert sein.

```
<<< HTTP/1.1 200 OK
<<<
<<< [Antwort von smart_device_1]
<<< [Antwort von kuehlschrank_1602]
<<< [Antwort von raumthermostat_fe4ecc03]
```

Listing 3.13: Virtuelle Antwort auf eine Multicast-Anfrage.

Eine passende Repräsentation des Inhaltes mehrerer Geräte wird in Kapitel 3.4 diskutiert.

3.3.4.2 Unicast-Kommunikation

Bei direkter Adressierung eines Gerätes, entweder über den URL-Pfad oder über einen gesetzten `Host`-Header, sind dagegen Fehlermeldungen zulässig und erforderlich. Dabei findet nur ein kleiner Teil der insgesamt in HTTP verfügbaren Statuscodes Verwendung; zum einen, um die Komplexität der Implementierung auf der MCU gering zu halten und zum anderen, da eine detaillierte Fehlerbeschreibung einfach nicht notwendig ist.

Die folgenden Status-Codes sind für Unicast-Kommunikation definiert, wobei in den Nutzdaten eine textuelle Repräsentation der Fehlermeldung für menschliche Benutzer enthalten sein kann:

- *200 OK*. Meldet ein erfolgreiches Ausführen der Anfrage und retourniert die Ressource(n) in passender Repräsentation.

- *400 Bad Request.* Die Anfrage war fehlerhaft aufgebaut; z.B. war der Asterisk zum Ersetzen eines Teilstrings des URL-Pfades anstatt eines gesamten Segmentes verwendet worden.

- *404 Not found.* Eine bestimmte Ressource wurde nicht gefunden. Wird nur gesendet, wenn der empfangene URL-Pfad keiner Ressource zugeordnet werden kann.

- *405 Method not allowed.* Verbietet die Methode für eine vorhandene Ressource. Würde eine Ressource etwa keine Events akzeptieren, so kann dieser Statuscode bei Aufruf der `CONNECT`-Methode zur Anwendung kommen. Darf auch dynamisch gesendet werden: beispielsweise könnte der Code auf einen `POST`-Request zum Öffnen der Türe einer Waschmaschine gesendet werden, wenn diese gerade in Betrieb ist. Um die Komplexität geringer zu halten, wird dies nicht über `423 Locked` abgebildet.

- *413 Request Entity Too Large.* Der Nachrichteninhalt ist zu groß. Der Fehler wird gesendet, wenn nach einer bestimmten Anzahl von Zeichen im Nutzdatenteil einer `POST`-, `CONNECT`- oder `DISCONNECT`-Anfrage keine Ressource identifiziert werden konnte.

- *414 Request-URL Too Long.* Der URL-Pfad ist zu lang. Der Fehler wird gesendet, wenn nach einer bestimmten Anzahl von Zeichen einer `GET`-, `POST`-, `CONNECT`- oder `DISCONNECT`-Anfrage keine Ressource identifiziert werden konnte.

- *500 Internal Server Error.* Das Smart Device hat einen internen Fehler festgestellt und kann die Anfrage nicht verarbeiten. Bezeichnet einen dauerhaften Fehler, den das Gerät nicht selbst beheben kann (CRC-Fehler im Speicher, Mangelnde Stromversorgung, etc.). Eine Fehlerbeschreibung erfolgt im Nutzdatenteil.

Andere Statuscodes werden bei SHDP nicht verwendet.

3.3.5 Fehlererkennung im Anwendungsprotokoll

Selbst bei postulierter Fehlerfreiheit und Zuverlässigkeit des Nachrichtentransports sollte ein Anwendungstransportprotokoll robust gegen Übertragungsfehler konzipiert sein. Wenn auch aufgrund von fehlender Redundanz keine Fehlerkorrektur möglich sein wird, so wird in diesem Unterkapitel untersucht, inwieweit bei HTTP-Nachrichtenübertragung Fehler erkannt werden können und wie das Protokoll darauf reagiert. Dazu wird die Hamming-Distanz zwischen zulässigen Code-Worten, hier die Verben und Ressourcen-Teile, berechnet oder abgeschätzt.

Auf OSI-Schicht 7 werden die folgenden zwei Fehlerklassen behandelt:

- *Bitfehler.* Ein oder mehrere Bits einer Nachricht wechseln ihren Wert. Damit erhält ein Zeichen einen anderen American Standard Code for Information Interchange (ASCII)-Wert und entspricht nicht mehr der ursprünglichen Bedeutung.

- *Nachrichtenlänge.* Ein oder mehrere Bits gehen verloren oder werden zusätzlich eingefügt, z.B. bei Bussen mit Taktrückgewinnung, die falsch synchronisiert sind oder asynchronen Bussen, die jeweils unterschiedliche Symbolraten konfiguriert haben.

Betrachtet man das Protokoll HTTP, so können in den vier bereits identifizierten Protokollteilen folgende Fehler auftreten:

- *Verb.* Keines der zulässigen Verben oder ein falsches Verb wird erkannt.
- *Ressource.* Es wird eine nicht vorhandene oder die falsche Ressource angesprochen.
- *Header.* Es wird kein zulässiger Header (`Host`, `Content-Length`) erkannt. Das Gerät antwortet nicht, da es nicht angesprochen wurde oder wegen fehlender Inhaltslänge die Verarbeitung ohne Fehlermeldung abbricht.
- *Body.* Es werden falsche Werte erkannt und in den Maschinenspeicher übernommen. Die Nachricht, insbesondere bei langen Inhalten, ist unvollständig.

In den folgenden Unterkapiteln werden einige der Fehler auf Relevanz und Vermeidbarkeit hin untersucht.

3.3.5.1 Hamming-Distanz der definierten Verben

In diesem Unterkapitel wird die Hamming-Distanz der fünf definierten Verben jeweils zueinander ermitteln. Da die Verben unterschiedliche Längen haben, wird für die nicht belegten Stellen das Null-Byte (`0x00`) angenommen. Das Vorgehen ist beispielhaft für die Verben `GET` und `POST` in Listing 3.14 abgebildet.

```
1 GET:  01000111 01000101 01010100 00000000
2 POST: 01010000 01001111 01010011 01010100
3 XOR:  00010111 00001010 00000111 01010100
```

Listing 3.14: Hamming-Distanz der HTTP-Verben `GET` und `POST` in ASCII-Codierung. Die Distanz beträgt 12 bit. Es müssen von übertragenen 32 bit genau die angegebenen 12 kippen, damit die falsche Operation ausgeführt wird. Kippen weniger, mehr oder andere Bits, würde der Programmcode kein gültiges Verb erkennen und mit `405 Method not allowed` antworten.

Aus Tabelle 3.1 ist die minimale Hamming-Distanz zwischen der binären Repräsentation der definierten Verben zu erkennen. Würde ein Programmcode nicht die gesamten Verben vergleichen, sondern die Verarbeitung schon nach einer Übereinstimmung fortsetzen, so genügten drei gekippte Bit (`POST` ↔ `DISCONNECT`) für eine fehlerhafte Operation. Nachdem sich die Nutzdaten zwischen `POST` und `DISCONNECT` aber in ihrer Semantik unterscheiden, erscheint das Risiko gering.

3.3.5.2 URL-Pfad und Ressourcen

Schwieriger gestaltet sich die Überlegung bei falsch erkannten Ressourcen im URL-Pfad, da der Wertebereich nicht mehr fest vorgegeben ist und vom Entwickler selbst gewählt

	GET	POST	EVENT	CONNECT	DISCONNECT
GET	*0 / 0*				
POST	9 / 12	*0 / 0*			
EVENT	6 / 14	12 / 16	*0 / 0*		
CONNECT	9 / 25	9 / 21	9 / 17	*0 / 0*	
DISCONNECT	8 / 35	3 / 27	11 / 31	8 / 20	*0 / 0*

Tabelle 3.1: Hamming-Distanz der ASCII-codierten, binären Repräsentation zwischen den definierten Verben bei SHDP. Der linke Wert gibt die Distanz nur bis zur Länge des kürzeren Verbs an; daran kann abgeschätzt werden, wie wahrscheinlich eine falsche Zuordnung bereits nach Übertragung der Anzahl an Bytes des kürzeren Verbs ist. Der rechte Wert bezeichnet die Distanz, wenn für fehlende Zeichen das Null-Byte (`0x00`) angenommen wird.

werden kann. Beispielsweise könnten Ressourcen auf einem Smart Device wie in Listing 3.15 benannt sein.

```
1 /led00: 00101111 01101100 01100101 01100100 01101100 00110000 00110000
2 /led01: 00101111 01101100 01100101 01100100 01101100 00110000 00110001
3 /led02: 00101111 01101100 01100101 01100100 01101100 00110000 00110010
4 /led03: 00101111 01101100 01100101 01100100 01101100 00110000 00110011
```

Listing 3.15: Hamming-Distanz von ungünstig benannten Ressourcen.

Hier beträgt die Hamming-Distanz nur noch zwischen ein und zwei Bit zwischen zwei bezeichnenden Ziffern, geschuldet der ASCII-Codierung. Aus etablierten Methoden der Softwareentwicklung [50, Kapitel 2] ist bekannt, daß genau diese fortlaufenden Nummern Fehlerpotential bieten, wenn etwa der Entwickler vergißt, sie bei Feldoperationen oder Feldindizes an allen Stellen konsistent anzupassen.

Eine Lösung besteht in der Erhöhung der Hamming-Distanz durch Ausdrücken von Ziffern mit Worten, wie im folgenden Listing 3.16 dargestellt.

```
1 /led_one: 01101100 01100101 01100100 01011111 01101111 01101110 01100101
2 /led_two: 01101100 01100101 01100100 01011111 01110100 01110111 01101111
3 XOR:      00000000 00000000 00000000 00000000 00011011 00011001 00001010
```

Listing 3.16: Hamming-Distanz erhöht durch zusätzliche Information bzw. Redundanz bei der Benennung von Ressourcen.

Mit der einfachen Methode konnte die Hamming-Distanz auf 9 bit erhöht werden. Außerdem führt jede andere Bitkombination für eine Ressource zu einem `404 Not Found`-Fehler.

3.3.5.3 Header

Die zwei verwendeten Header `Host` und `Content-Length` sind die einzigen erlaubten Zeichenketten. Ohne daß die Hamming-Distanz zwischen den diesen Codeworten gesondert berechnet werden müßte, kann auch wegen der unterschiedlichen Nachrichteninhalte (Text versus Zahl) eine Verwechselung hinreichender Sicherheit ausgeschlossen werden.

Auch im Falle von Bitfehlern besteht nur ein geringes Risiko, da anderslautende Headernamen stillschweigend ignoriert werden. Es besteht daher realistisch nur das Risiko, daß ein angesprochenes Smart Device aufgrund eines defekten `Host`- oder eines nicht erkannten `Content-Length`-Headers nicht reagiert.

3.3.5.4 Nutzdaten

Im Nutzdatenteil von HTTP existiert aus Sicht des Protokolls keine fixe Struktur. Alle Daten sind je nach definiertem Encoding zulässig. Damit ist eine zuverlässige Fehlererkennung an dieser Stelle nicht möglich, wenn ein Fehler nicht durch Verletzung von Konventionen des verwendeten Datenformats erkannt werden kann.

Bei Übermittlung von Kommandos an Smart Devices besteht das Risiko, daß einzelne Bitfehler insbesondere in den signifikanteren Stellen fatale Auswirkungen haben können. Beispielsweise soll die Drehzahl eines Motors auf 1000 Umdrehungen pro Minute gesetzt werden. Wie im folgenden Listing 3.17 abgebildet, ist nur ein einziges gekipptes Bit für eine vollkommen falsche und zu hohe Angabe des Zielwertes verantwortlich.

```
1 >>> POST /fancontrol_000030A0/motor_zero/target_speed HTTP/1.1
2 >>> Content-Length: 4
3 >>>
4 >>> 1000

5 1000: 00110001 00110000 00110000 00110000
6 9000: 00111001 00110000 00110000 00110000
7 XOR:  00001000 00000000 00000000 00000000
```

Listing 3.17: Fatale Auswirkungen eines Bitfehlers in Nutzdaten.

Selbst wenn ein gefährlicher Zustand einer Maschine in deren Programmlogik abgefangen und verhindert wird und sie durch ein fehlerhaftes Kommando nicht in einen solchen Zustand gebracht werden können darf, ist alleine die Möglichkeit der Gefährdung durch das Anwendungstransportprotokoll nicht hinzunehmen und muß ausgeschlossen werden. Dafür existieren zwei Möglichkeiten.

1. Sofern die gesamte Nachricht vor Versand bekannt ist, könnte der Header `Content-MD5` von HTTP eingesetzt werden. Eine kryptographisch schwache MD5-Quersumme über die Nutzdaten der Nachricht repräsentiert eine mit hinreichender Sicherheit prüfbare Integrität des Inhaltes. Allerdings ist zum einen, wie bereits gezeigt, der gesamte Inhalt nicht immer vorab bekannt. Und zum anderen muß auf der MCU ausreichend Speicherplatz für die gesamte Nachricht vorgehalten und weitere Programmlogik zur Prüfung der Nachricht implementiert werden.

2. Die zweite Lösung spiegelt geänderte Werte an den Sender zurück und ist für SHDP eher geeignet. Ein Smart Device sendet alle Werte, die sich durch ein Kommando (`POST`, `CONNECT` oder `DISCONNECT`) geändert haben, an den Client. Dessen Anwendungslogik entscheidet über weitere Schritte, z.B. ein erneutes Senden oder eine Benachrichtigung des Benutzers.

Ein kurzfristiges Fehlverhalten der Maschine kann so aber nicht ausgeschlossen werden. Nochmals sei daher auf die Aufgabenverteilung hingewiesen, nach der der Nachrichtentransport für die Integrität der Nachrichten einsteht.

3.4 Beschreibung von Diensten und Geräten

Während im vorigen Kapitel die Steuerschnittstelle basierend auf HTTP vorgestellt wurde, definiert dieser Teil der Arbeit die über SHDP übertragenen Gerätebeschreibungen. Zum einen wird die Auflistung von Smart Devices an einem Netzwerksegment („Discovery“) behandelt, zum anderen werden die Besonderheiten der Befehlsschnittstelle und ihre Abbildung auf HTTP-Befehle beschrieben.

3.4.1 Dienste und Metadaten in Smart Devices

Zunächst wird der logische Aufbau der Maschinenschnittstelle eines Smart Devices beschrieben. Letztere ist wegen eines einheitlichen Nachrichtenformats und einer einfachen Implementierung als dreistufige Hierarchie konzipiert:

1. *Geräte-Ebene.* In dieser Ebene wird die Identifikation („ID“) für ein Smart Device vorgehalten. Sie enthält ansonsten keine weitere Information und dient als logischer Container für Instanzen der folgenden Ebene.

2. *Metadaten- und Dienste-Ebene.* Zunächst enthält diese Ebene (optionale) Metadaten über ein Gerät. Ohne bereits hier Details zu nennen, können diese vom Entwickler bzw. Hersteller frei definiert werden. Ihre Anzahl und ihr Inhalt ist durch den verfügbaren Speicher auf der verwendeten Plattform begrenzt. Weiterhin existiert mindestens eine Instanz eines weiteren Typs, bezeichnet als „Service“. Auch eine Service-Instanz besitzt außer einem auf dem Gerät eindeutigen Namen keine weitere Information, sondern enthält mindestens eine Instanz eines *Datenpunktes*.

3. *Datenpunkt-Ebene.* Die Maschinenzustände des Smart Devices werden in dieser Ebene abgebildet. Ein Datenpunkt definiert dabei nur einen abstrakten Zugang zum Gerät. Welche Maschinenzustände auf einen Datenpunkt abgebildet werden, bleibt dem Entwickler überlassen.

Die vorgestellte Hierarchie wird nach außen hin in einem XML-Schema repräsentiert und in Antworten auf `GET`-, `CONNECT`-, `DISCONNECT`- und `POST`-Befehle verwendet, um Daten an den Client auszuliefern.

Essentiell ist hierbei, daß XML auf Smart Devices *nicht* verarbeitet werden muß, sondern lediglich on-the-fly erzeugt wird. Aufwand für SAX-Parser kann so vermieden werden. Wie bereits in den Definitionen zur ereignisbasierten Kommunikation ausgeführt, ist auch bei der Kommunikation zwischen Smart Devices keine Verarbeitung von XML notwendig. Die Nutzdaten enthalten den unverändert nutzbaren Nachrichteninhalt.

Wenn in folgenden Kapiteln Bezeichner und Namen definiert werden, so gilt, daß der Entwickler bzw. Hersteller diese frei wählen darf. Zur Wertefindung muß der Einsatzzweck und die Zielgruppe des Smart Devices bedacht werden. Es sind möglichst kurze und sprechende Namen in der jeweils am besten geeigneten Sprache zu verwenden. Aus Kompatibilitätsgründen sind nur druckbare ASCII-Zeichen gestattet. Gegebenenfalls ist vorab eine Transliteration der Fremdsprache durchzuführen.

Hintergrund ist, daß der Programmspeicher (Flash-ROM) auf kleinen MCUs begrenzt ist. Es gilt, die Maschinenschnittstelle (Gerät und Dienste) mit möglichst wenig und treffender Information so zu beschreiben, daß ein menschlicher Anwender ohne Lektüre der Dokumentation bereits der Interpretation der Schnittstelle fähig ist. Eine fest definierte Ontologie existiert aber nicht, vielmehr sind Kriterien wie Landessprache und Funktionalitäten von Diensten und Datenpunkten für eine passende Benennung heranzuziehen.

Wie für XML spezifiziert, werden alle Knotennamen nach Groß- und Kleinschreibung, d.h. ihrem Byte-Wert, unterschieden. Auch das vereinfacht die Implementierung geringfügig, da keine Zuordnungstabellen vorgehalten werden müssen, sondern Bytes mit der *C*-Funktion `strcmp()` verglichen werden können.

3.4.1.1 Geräte-Ebene für Unicast-Antworten

Jedes Smart Device erhält durch nicht spezifizierte Mechanismen (z.B. feste Vergabe bei Herstellung, Bootstrapping bei erstmaliger Integration in ein Netz, Eingabe an einer Benutzerschnittstelle) eine zeichenbasierte Identifikation (ID). Sie darf aus allen Zeichen bestehen, die für XML-Knotennamen gültig sind [51] unter Ausschluß von Unicode-Werten.

Die ID kann fest im ROM des Programmcodes oder in anwenderprogrammierbarem Speicher (Electrically Erasable and Programmable Read-Only Memory (EEPROM)) abgelegt sein. Sie ist auf der obersten logischen Ebene die einzige Information, die ein Smart Device enthält.

In Verbindung mit der bereits erwähnten XML-Ausgabe wird sie als Wurzelknoten des Dokumentes des einzelnen Gerätes gesetzt, wie in Listing 3.18 dargestellt. Außerdem erzeugt der Programmcode einen passenden HTTP- sowie XML-Header, der vorab auszugeben ist.

```
1 HTTP/1.1 200 OK
2 Content-Length: 201

3 <?xml version="1.0" encoding="iso-8859-1"?>
4 <fancontrol_000030A0>
5   <!-- Inhalt der Antwort des Smart Devices. -->
6 </fancontrol_000030A0>
```

Listing 3.18: Darstellung von HTTP- und XML-Header sowie des Rootknotens in den Nutzdaten des Protokolls. Die Codierung der Nutzdaten ist im XML-Header definiert und muß der Ausgabe des Smart Devices entsprechen.

3.4.1.2 Geräte-Ebene für Multicast-Antworten

Sind mehrere Geräte *angesprochen*, so wird ein gemeinsames Dokument on-the-fly generiert. Dazu ist eine Unterstützung durch das zugrundeliegende Nachrichtentransportsystem notwendig. Eine Busbelegung muß erkannt werden können. Im Unterschied zur Unicast-Antwort kann eine Device-ID nicht mehr als Wurzelknoten verwendet werden, da jedes Gerät gleichberechtigt und flach nebeneinander an einem Netzsegment existiert, und ein wohlgeformtes XML-Dokument nur aus einem Wurzelknoten bestehen darf.

Es wird daher ein neuer Wurzelknoten ohne Information eingeführt, der den Namen `bus` trägt. Er enthält in seinen Kind-Knoten die Antworten der einzelnen Smart Devices (siehe Listing 3.19), die in den folgenden Kapiteln beschrieben werden.

```
HTTP/1.1 200 OK

<?xml version="1.0" encoding="iso-8859-1"?>
<bus>
  <fancontrol_000030A0>
    <!-- Inhalt der Antwort von fancontrol_000030A0.-->
  </fancontrol_000030A0>
  <toaster>
    <!-- Inhalt der Antwort von toaster. -->
  </toaster>
  <!-- Antworten weiterer Devices... -->
</bus>
```

Listing 3.19: Darstellung von Multicast-Antworten in einem virtuell erzeugten Dokument. HTTP- und XML-Header sowie das `<bus>`-Element (in rot dargestellt) werden von einem Gerät erzeugt. Anschließend senden alle betroffenen Smart Devices ihre Antworten (cyan und blau dargestellt). Nachdem das letzte Gerät gesendet hat, schließt das erste Gerät das Dokument mit `</bus>`. Da die Gesamtlänge nicht bekannt ist, kann vom ersten Gerät kein `Content-Length` -Header gesendet werden. Stattdessen muss eine Fehlererkennung auf Anwendungsebene geschehen, indem auf das abschließende `</bus>`-Element gewartet wird.

Die Ausgabe des ersten Gerätes kann zum einen durch das am Bus zuerst antwortende Gerät erfolgen, das dann während der gesamten Transaktion aller Geräte aktiv bleibt und den Bus auf Aktivität überprüft. Bleibt eine solche für eine gewisse Zeitspanne, z.B. 5 s, aus, so wird das schließende `<\bus>` gesendet. In diesem Fall kann der `Content-Length`-Header nicht verwendet werden. Eine Anwendung muß damit den Empfang nach einer gewissen Zeitspanne selbst beenden und eine Inhaltsanalyse anstellen. Sie darf ein Dokument nur verwenden, wenn es der XML-Syntax entspricht.

Zum anderen kann die Ausgabe der türkis dargestellten Teile der Nachricht durch das Gateway erfolgen. Damit ist auf den Smart Devices weniger Komplexität erforderlich, denn die Logik, welche die Einordnung als erstes Gerät und die Erzeugung der passenden Header übernimmt, entfällt. Zudem kann das Gateway, wenn mit ausreichend Ressourcen ausgerüstet, zunächst alle Antworten der Smart Devices zwischenspeichern und die gesamte Länge berechnen. In diesem Fall ist der `Content-Length`-Header wieder mit einem gültigen Wert verwendbar.

Bei einer Multicast-Antwort wird eine einzige XML-Codierung für alle Geräte verwendet.

3.4.1.3 Metadaten zur Gerätebeschreibung

Um einem menschlichen Benutzer in (wenigen) Worten eine Funktionsbeschreibung und andere Attribute eines Gerätes zu übermitteln, werden innerhalb der Geräte-Ebene sogenannte Meta-Elemente definiert.

Ein Meta-Element stellt ein Schlüssel/Werte-Paar dar, dessen Inhalt und Bezeichnung vom Entwickler frei gewählt werden kann und in der Codierung des Gerätes (z.B. „iso-8859-1“) ausgegeben wird, siehe Listing 3.20.

```
<device_id>
  <meta type="$KEY$">$VALUE$</meta>
  <!-- Optional weitere Metaknoten. -->
</device_id>
```

Listing 3.20: Schema eines Metaknotens in einer Gerätebeschreibung. Der Name des Elements lautet stets `meta`. Es besitzt ein Attribut `type`, welches im Encoding des XML-Dokumentes einen beliebigen benutzerdefinierten Wert enthält. Auch der Wert des Elements ist benutzerdefiniert und im Encoding des Dokuments anzugeben.

In Kapitel 2.1.3 und 2.2.8.7 wurde argumentiert, daß eine fest definierte Semantik für Gerätebeschreibungen nicht ausreicht. Metaknoten sind daher optional und sollen so gewählt werden, daß Menschen, die ein Gerät einsetzen, möglichst kurz und prägnant über die Funktion des Gerätes informiert werden. Es sind daher so viele Metainformationen wie nötig und so wenige wie möglich zu definieren. Listing 3.21 gibt ein Beispiel an.

```
<cals000001>
  <meta type="deviceName">CS Awesome LightServ</meta>
  <meta type="deviceURL"></meta>
  <meta type="manufacturerName">TU Muenchen, Informatik, F13</meta>
  <meta type="manufacturerURL">http://www.os.in.tum.de</meta>
  <meta type="pcbRevision">1.1</meta>
  <meta type="firmwareRevision">1.2</meta>
  <!-- weiterer Inhalt der Gerätebeschreibung von cals000001. -->
</cals000001>
```

Listing 3.21: Beispielhafte Definition von Metaknoten in einer Gerätebeschreibung.

Metaknoten innerhalb der Gerätedefinition in einem eigenen Unterknoten zusammenzufassen wäre im Prinzip möglich. Allerdings böte eine solche Zusammenfassung für die Maschinenkommunikation keinen Vorteil; eine Abfrage aller Knoten kann auch so, z.B. über X-Path, erfolgen.

Weiterhin sind Metaknoten für die Maschine-zu-Maschine-Kommunikation nicht relevant. Wie auch durch Abbildung 3.2 demonstriert, sollen Metaknoten nur dem Anwender unverändert präsentiert werden.

Abbildung 3.2: Screenshot der interpretierten Metainformationen eines Smart Devices in einer beispielhaften Anwendersoftware. Die URL kann geklickt werden und ist unterstrichen, wie für Links im WWW gängig. Leere Elemente können ebenfalls entsprechend dargestellt werden.

Wenn möglich, kann allerdings eine Interpretation des Inhalts vorgenommen werden. Im Bild dargestellt ist das Unterstreichen einer URL, die nach Anklicken im Standardbrowser des Betriebssystems aufgerufen wird, sowie ein Kennzeichnen von leeren Elementen.

Ob ein Hersteller die Metainformationen zur Filterung von Geräten nach bestimmten Kriterien in seiner eigenen Anwendung heranzieht, bleibt ihm überlassen. Denkbar ist etwa ein Szenario, bei z.B. dem nach dem Wert in der Zeichenkette „Device Firmware Version“ gefiltert wird, um bestimmte Geräte mit bestimmten Firmware-Versionen aufzufinden und ihnen eine neue Version aufzuspielen. Dies ist allerdings ausdrücklich nicht vorgesehener Zweck der Metainformationen.

3.4.1.4 Dienste-Ebene

Als weitere Kindknoten innerhalb der Hierarchie eines Gerätes sind Diensteknoten („service nodes“) definiert. Ein Service-Knoten dient der logischen Gruppierung von Maschinenzustandsinformationen eines Smart Devices. Ähnlich der Geräte-ID tragen sie außer dem Knotennamen keine weitere Information.

```
<fancontrol_000030A0>
  <meta ...> ... </meta>
  <!-- weitere Metaknoten-Definitionen von fancontrol_000030A0. -->
  <Z-Achse> ... </Z-Achse>
  <X-Achse> ... </X-Achse>
  <Display-Einstellungen> ... </Display-Einstellungen>
  <!-- weiterer Inhalt der Gerätebeschreibung von fancontrol_000030A0. -->
```

```
8 </fancontrol_000030A0>
```

Listing 3.22: Beispielhafte Darstellung von Diensteknoten in einer Gerätebeschreibung.

Ein Diensteknoten darf jeden beliebigen Namen außer `meta` tragen. Wie erwähnt und in Listing 3.22 demonstriert soll jeder Knoten eine sprechende, übergeordnete, zusammenfassende und möglichst kurze Bezeichnung für eine Menge von Maschinenzuständen bekommen. Im Beispiel könnte `Display_Einstellungen` deswegen durch `Display` oder sogar `LCD` ersetzt werden.

Hersteller sind weiterhin angehalten, auf verschiedenen Geräten gleiche Funktionsgruppen auch mit gleichen Namen zu versehen. Haben z.B. ein Herd und ein Kühlschrank ein vergleichbares Benutzerinterface mit Display, so sollten die Service-Knoten auch beide `Display` oder ähnlich heißen. Zum einen erleichtert das einem Benutzer das Zurechtfinden innerhalb der Funktionsgruppen von Geräten. Zum anderen lassen sich damit umfangreichere Steuerungsfunktionen (siehe Kapitel 3.4.5) realisieren.

3.4.1.5 Datenpunkt-Ebene

Eine abstrakte Schnittstelle zu einzelnen Maschinenzuständen bietet die dritte Hierarchieebene. Sie ist innerhalb eines Dienste-Knotens angesiedelt. Sie ist die einzige Ebene, die (statische) Attribute für einzelne Elemente definiert. Außerdem besitzt sie – wie ein Meta-Element – einen eigenen Wert, der aus einem oder mehreren Maschinenzuständen abgeleitet wird.

Auch an Datenpunkte besteht die Anforderung ontologisch möglichst prägnant die Funktion und die Eigenschaften des Punktes über seinen Namen und die definierten Attribute auszudrücken. In der Beschreibung werden Eigenschaften direkt über XML-Attribute ohne Verwendung von Namensräumen definiert. Folgende Attribute sind möglich:

- *type*. Das Attribut gibt den Datentyp des Datenpunktes an. Es kann die Werte `int`, `float` und `string` annehmen und designiert den beinhalteten Wert analog als Ganzzahl, Fließkommazahl oder Zeichenkette. Zahlen sind stets vorzeichenbehaftet. Die Unterscheidung zwischen Ganz- und Fließkommazahl wird aus Implementierungsgründen getroffen. Sofern nur Festkommaberechnungen erforderlich sind, kann Programmcode für die Umwandlung von `float`-Variablen in eine Textrepräsentation sowie deren Berechnungsmakros eingespart werden.
- *min* und *max*-Beschränkungen. Beide Attribute sind optional. Für Zahlen geben sie den gültigen Wertebereich an. Ist eine Grenze nicht definiert, so wird die jeweils absolut größtmögliche Zahl implizit verwendet. Bei Ganzzahlen bedeutet das eine Länge von 31 bit; für Fließkommazahlen die Konstrukte `+/-INF`. Für eine Zeichenkette wird die minimale und maximale Anzahl an Byte unabhängig vom Encoding festgelegt. Bei Multibyte-Zeichensätzen wie UTF-8 ist auf eine ausreichende Dimensionierung zu achten. Ist bei Zeichenketten die minimale Länge `0`, so kann das Attribut entfallen.

- *access*. Das Attribut gibt Zugriffsmöglichkeiten auf einen Datenpunkt an. Mögliche Werte sind `r`, `w` und `rw`; der Buchstabe „r“ bezeichnet dabei lesenden, der Buchstabe „w“ schreibenden Zugriff, analog über die HTTP-Verben `GET` und `POST`. Das Attribut ist zwingend vorgeschrieben.
- *unit*. Um einen Datenpunkt näher zu beschreiben, kann optional eine im jeweiligen Encoding definierte Einheit angegeben werden. Auch hier ist keine Ontologie bindend definiert; der Entwickler ist angehalten, den Datenpunkt so treffend als möglich zu beschreiben. Die Einheit ist in einer Anwendungssoftware stets hinter dem Wert des Datenpunktes darzustellen, sowohl in Anzeigen des aktuellen Wertes, wie auch in Eingabefeldern zur Änderung des Wertes.
- *free-event-slots*. Sofern ein Datenpunkt Schreibzugriff bietet, kann der Entwickler den Empfang von `EVENT`-Nachrichten vorsehen. Im beschriebenen Attribut wird dann die Anzahl der freien Speicherplätze für eine Event-Quelle angegeben. Ist sie Null oder akzeptiert ein Datenpunkt keine Ereignisnachrichten, so entfällt das Attribut. Die Menge der insgesamt zur Verfügung stehenden Speicherplätze kann über die hier angegebene Zahl und die Liste der bereits belegten Speicherplätze (Kapitel 3.4.6) berechnet werden.

Der Wert des Elements, präziser sein erster Textknoten (null-indiziert), enthält den Wert des Maschinenzustandes, der beim Auslesen durch den Programmcode ermittelt wird. Beim Schreiben eines Datenpunkts – sofern gestattet – prüft das Programm den übergebenen Wert gegen eventuelle Beschränkungen und bildet ihn auf den Maschinenzustand ab, etwa durch Überschreiben einer Speicherzelle. Letzterer muß nicht notwendigerweise durch einen einzelnen Speicherzustand ermittelt werden. Auch eine vorherige Transformation ist denkbar: so könnte etwa der Datenpunkt `Durchschnittstemperatur` von einer Programmlogik durch Mittelwertbildung der Werte aller an einer Maschine angeschlossenen Temperatursensoren vor Auslieferung bestimmt werden.

Die bisherigen Ausführungen ergänzen das Beispiel aus Listing 3.22 im folgenden Listing 3.23 und demonstrieren, wie geeignet gewählte Knotennamen die Funktion eines Gerätes beschreiben. Selbst technisch wenig versierte Anwender würden in einer aus der XML-Beschreibung parametrisierten Oberfläche ein Gerät zielgerichtet bedienen können, selbst wenn ihnen nur die angegebenen Definitionen zur Verfügung stehen.

```
1  <fancontrol_000030A0>
2    <meta ...> ... </meta>
3    <!-- weitere Metaknoten-Definitionen von fancontrol_000030A0. -->
4    <X-Achse>
5      <Winkel type="int" access="rw" min="-90" max="90" unit="°">30</Winkel>
6      <Oszillationsstufe type="int" access="rw" min="0" max="3">0</
           Oszillationsstufe>
7    </X-Achse>
8    <Z-Achse>
9      <Winkel type="int" access="rw" min="-90" max="90" unit="°">30</Winkel>
10     <Oszillationsstufe type="int" access="rw" min="0" max="3">0</
           Oszillationsstufe>
11   </Z-Achse>
12   <Display-Einstellungen>
```

```
13      <Helligkeit type="int" access="rw" min="0" max="100" unit="%">100</
            Helligkeit>
14      <Kontrast type="int" access="rw" min="0" max="100" unit="%">56</
            Kontrast>
15    </Display-Einstellungen>
16    <!-- weiterer Inhalt der Gerätebeschreibung von fancontrol_000030A0. -->
17  </fancontrol_000030A0>
```

Listing 3.23: Darstellung von Datenpunkten und ihren Attributen in einem Smart Device.

Nach Definition des Document Object Model (DOM) sind alle Teile eines XML-Dokumentes stets Knoten oder „nodes“. Auch ein reiner Textwert zwischen einer Element-Definition und ihrem schließenden Pendant hat niemals nur einen Wert, sondern ist als eigener, impliziter Knotentyp „text node“ mit einem Wert definiert. Damit können Werte nur Attributen und den genannten Textknoten zugewiesen werden.

Wichtig wird diese Eigenschaft der Sicht auf ein XML-Dokument für die folgenden beiden Erweiterungen des Datenpunktes, denn der erste – null-indizierte – Inhalt eines Datenpunktes des Protokolls ist stets ein solcher Textknoten, der den Wert des Datenpunktes (abgeleitet aus einem Maschinenzustand) beinhaltet. Dies gestattet, innerhalb eines Datenpunktes weitere und explizit deklarierte Knotentypen („Elemente“) anzulegen.

3.4.1.6 Erweiterte Eigenschaften des Datenpunktes: label-Element

Numerische Datenpunkte können mit einem oder mehreren `label`-Elementen ergänzt werden, um bestimmte Werte oder Wertebereiche des Datenpunktes näher zu bezeichnen. Das Element besitzt zwei Attribute, `from` und `to`, die den für die Beschriftung gültigen Bereich definieren. Der Wert des Elements gibt den Text oder die Bezeichnung an, die ein Anwenderprogramm darzustellen hat, wenn sich der Wert des Datenpunktes innerhalb des von `from` und `to` definierten Bereiches befindet.

Das Attribut `to` ist optional. Wenn es fehlt, gilt implizit der Wert aus `from`. So kann ein singulärer Datenpunktwert bezeichnet werden.

Listings 3.24 gibt ein Beispiel für die Verwendung an.

```
1  <fancontrol_000030A0>
2    <meta ...> ... </meta>
3    <!-- weitere Metaknoten-Definitionen von fancontrol_000030A0. -->
4    <X-Achse>
5      <Winkel type="int" access="rw" min="-90" max="90" unit="°">
6        30
7      </Winkel>
8      <Oszillationsstufe type="int" access="rw" min="0" max="3">
9        0 <!-- Impliziter Text-Node -->
10       <label from="0">Keine Oszillation</label>
11       <label from="1">Schwache Oszillation</label>
12       <label from="2">Mittlere Oszillation</label>
13       <label from="3">Stärke eines Passatwindes</label>
14     </Oszillationsstufe>
```

```
15    </X-Achse>
16    <!-- weiterer Inhalt der Gerätebeschreibung von fancontrol_000030A0. -->
17  </fancontrol_000030A0>
```

Listing 3.24: Beispiel für label-Elemente eines Datenpunktes ohne Überschneidung von Wertebereichen. Eine geeignete Darstellung ist für diesen Fall ein DropDown-Menü.

Wertebereiche von Labels dürfen sich überschneiden, siehe Listing 3.25. In diesem Fall muß die Anwendung dem Benutzer alle geeigneten Werte in ihrer Oberfläche darstellen.

```
1  <umweltsensor>
2    <meta ...> ... </meta>
3    <!-- weitere Metaknoten-Definitionen von umweltsensor. -->
4    <Helligkeit>
5      <Sensor type="int" access="r" min="0" max="120000" unit="lx">
6        905 <- Impliziter Text-Node
7        <label from="0"     to="1000"  >Dunkelheit</label>
8        <label from="800"   to="5000"  >Dämmerung</label>
9        <label from="5001"  to="9999"  >Tageslicht</label>
10       <label from="10000" to="120000">Heller Sonnenschein</label>
11     </Sensor>
12   </Helligkeit>
13   <Windgeschwindigkeit>
14     <Sensor type="float" access="r" min="0" max="200" unit="kn">
15       6 <!-- Impliziter Text-Node -->
16       <label from="0"  to="0.9">0 Bft</label>
17       <label from="1"  to="3.9">1 Bft</label>
18       <label from="4"  to="6.9">2 Bft</label>
19       <!-- weitere label-Elemente für den Sensor-Datenpunkt. -->
20     </Sensor>
21   </Helligkeit>
22   <!-- weiterer Inhalt der Gerätebeschreibung von umweltsensor. -->
23 </umweltsensor>
```

Listing 3.25: Beispiel für label-Elemente eines Datenpunktes mit Überschneidung von Wertebereichen. Die Anwendung hat in diesem Falle beide Label-Werte darzustellen.

`label`-Elemente sind für die Maschine-zu-Maschine-Kommunikation nicht relevant und werden nur im Zuge der Geräteerkennung für eine tiefere Beschreibung von Datenpunkten ausgeliefert. Von einem Client wird ein Zwischenspeichern der Datenstruktur erwartet, damit der Erkennungsprozeß so selten wie möglich ausgeführt werden muß.

3.4.1.7 Erweiterte Eigenschaften des Datenpunktes: listensTo-Element

```
1  <fancontrol_000030A0>
2    <meta ...> ... </meta>
3    <!-- weitere Metaknoten-Definitionen von fancontrol_000030A0. -->
4    <X-Achse> <!-- ... --> </X-Achse>
5    <Z-Achse> <!-- ... --> </Z-Achse>
6    <Ventilator>
7      <!-- ... -->
```

```
8       <Umschalteingang type="int" access="w" min="1" max="1" free-event-
            slots="1">
9         1
10        <listensTo>/schalter/Taste/Kurzer-Druck</listensTo>
11      </Umschalteingang>
12    </Ventilator>
13    <!-- weiterer Inhalt der Gerätebeschreibung von fancontrol_000030A0. -->
14  </fancontrol_000030A0>
```

Listing 3.26: Beispiel für das listensTo-Element eines Datenpunktes für das Empfangen von EVENT-Nachrichten. Wie in der Attributbeschreibung zu `free-event-slots` angeben, gibt es die Anzahl der *freien* Speicherplätze für Eventquellen an. Im Beispiel bedeutet das eine von insgesamt zwei freien Quellen. Die bestehende Verknüpfung wird über `listensTo`-Elemente definiert.

Auf gleicher Ebene mit den optionalen `label`-Elementen befinden sich die Instanzen des `listensTo`-Elementes. Jedes Auftreten gibt eine Quelle eines Events an, auf welche der Datenpunkt registriert ist, siehe Listing 3.26.

Es sind hierfür nur Quellen zulässig, die am Segment einen anderen Datenpunkt eineindeutig bezeichnen. Der Asterisk als Platzhalter ist auf keiner der drei Ebenen zulässig. So kann eine Implementierung einfacher gestaltet werden.

Im Falle des EVENT-Verbs wird ein Vergleich der vollständig übertragenen URL des Events und einer lokal gespeicherten Tabelle angestellt. Kann in der Tabelle der exakte Wert aufgefunden werden, so gibt eine weitere Zuordnung die Datenpunkte des empfangenden Gerätes an, die den Wert des Events übergeben bekommen. Jeder Datenpunkt führt damit den selben Code aus, der für das Verb POST vorgesehen ist: nach einer Prüfung der Beschränkungen wird der Maschinenzustand anhand des Event-Wertes geändert.

Derzeit ist nicht vorgesehen, die Verknüpfungen über eigene GET-Anfragen durch Clients ermitteln zu können. Stattdessen werden sie nur im Zuge der Geräteerkennung in der Gesamtbeschreibung, bzw. nach Änderung durch CONNECT oder DISCONNECT ausgeliefert.

3.4.1.8 Vollständiges Beispiel

Im folgenden Listing 3.27 ist ein vollständiges Beispiel einer Gerätebeschreibung dargestellt. Die Verwendung aller bisher definierten Elemente und Attribute wird demonstriert.

```
1  <fancontrol_000030A0>
2    <meta type="deviceName"        >Lüfter-Demonstrator</meta>
3    <meta type="deviceDescription">Demonstriert das Beschreibungsschema des
         CS Device Network (CSDN) und bietet beispielhafte Implementierungen
         von Funktionen.</meta>
4    <meta type="serialNumber"      >0000C501</meta>
5    <meta type="firmwareVersion"   >1.5b</meta>
6    <meta type="pcbRevision"       >1.2</meta>
7    <X-Achse>
8      <Winkel type="int" access="rw" min="-90" max="90" unit="°">
9        30
```

```
10        <label from="0"           >Neutralstellung</label>
11        <label from="-90" to="-1">Luftstrom abwärts gerichtet</label>
12        <label from="1"   to="90">Luftstrom aufwärts gerichtet</label>
13      </Winkel>
14      <Oszillationsstufe type="int" access="rw" min="0" max="2">
15        0
16        <label from="0">Keine Oszillation</label>
17        <label from="1">Schwache Oszillation</label>
18        <label from="2">Mittlere Oszillation</label>
19      </Oszillationsstufe>
20    </X-Achse>
21    <Z-Achse>
22      <Winkel type="int" access="rw" min="-90" max="90" unit="°">
23        30
24        <label from="0"           >Neutralstellung</label>
25        <label from="-90" to="-1">Links geschwenkt</label>
26        <label from="1"   to="90">Rechts geschwenkt</label>
27      </Winkel>
28      <Oszillationsstufe type="int" access="rw" min="0" max="2">
29        0
30        <label from="0">Keine Oszillation</label>
31        <label from="1">Schwache Oszillation</label>
32        <label from="2">Mittlere Oszillation</label>
33      </Oszillationsstufe>
34    </Z-Achse>
35    <Ventilator>
36      <Drehzahl type="int" access="r" min="0" max="2000" unit="1/min">1543</
          Drehzahl>
37      <Stufe type="int" access="rw" min="0" max="100" unit="%">75</Stufe>
38      <Umschalteingang type="int" access="w" min="1" max="1" free-event-
          slots="1">
39        1
40        <listensTo>/schalter0312/Obere-Taste/Kurzer-Druck</listensTo>
41      </Umschalteingang>
42    </Ventilator>
43    <Display-Einstellungen>
44      <Helligkeit type="int" access="rw" min="0" max="100" unit="%">100</
          Helligkeit>
45      <Kontrast type="int" access="rw" min="1" max="10">6</Kontrast>
46    </Display-Einstellungen>
47  </fancontrol_000030A0>
```

Listing 3.27: . Darstellung eines vollständigen Beispiels der Gerätebeschreibung eines Smart Devices. Eine Besonderheit stellt die Modellierung des Datenpunktes `Umschalteingang` dar. Der Datenpunkt darf nur beschrieben werden und akzeptiert nur eine `1` als Wert. Außerdem besitzt er zwei Speicherplätze für Event-Quellen und ist bereits auf einen Schalter registriert. Ein sinnvoll modellierter Schalter würde beim Drücken eine `1` und beim Loslassen eine `0` per Broadcast versenden; somit würde ein Druck auf den Schalter zum Umschalten des Betriebszustandes des Ventilators führen. Nicht behandelt ist die Frage, ob die gesamte Maschine (Oszillationen, LCD-Beleuchtung) abschaltet oder nur der Lüfter seinen Zustand wechselt. Da der Datenpunkt aber innerhalb des `Ventilator`-Services angesiedelt ist, kann von Letzterem ausgegangen werden.

3.4.1.9 Anmerkungen und Hintergrundgedanken zur Hierarchie

Das konstruierte Beschreibungsschema für Smart Devices soll auf diesen möglichst einfach implementiert werden können. Daher besteht es aus einer statischen Hierarchie und spart explizite Typenzuordnungen über Attribute (`<name type="datapoint"...>`) ein. Letztere werden stattdessen implizit über die Stufe in der Hierarchie vorgenommen.

So können die erzeugten XML-Dokumente mit einfacheren X-Path-Suchanfragen traversiert werden: alle Datenpunkte eines Gerätes etwa erhält man mit der Abfrage `/$device-id$/*/*`. Der Code zur Beantwortung dieser Abfrage kann mit wenig Aufwand direkt im Smart Device realisiert werden. Es sind lediglich *C*-Funktionen wie `strcmp()` und `strtok()` und einige Vergleichsoperationen notwendig.

Wären Datenpunkte mit z.B. `type="dp"` typisiert und könnten in einer hierarchisch flexiblen Struktur angeordnet werden, sähe die Abfrage eines Clients komplexer aus: `//*[type="dp"]` (jedes Element, das ein auf „dp“ gesetztes „type“-Attribut besitzt). Eine derartige Abfrage flexibel zu unterstützen ist auf Smart Devices der vorgesehenen Hardwareklasse nicht mehr einfach möglich.

3.4.2 Geräteauflistung („Discovery“)

Ein Netzwerksegment wird als das kleinste Teilstück eines Broadcast-Mediums definiert, über welches eine Menge an Smart Devices fehlerfrei in einer bestimmten, von der Anwendung zu definierenden Zeit erreichbar ist. Begrenzt wird es durch die Eigenschaften des Mediums an sich (z.B. Kabellänge) oder die Adaption daran (Reichweite der Funkstrecke, basierend auf der Sendeleistung, Dämpfung, etc.).

Die erste Operation, die ein Client in einem unbekannten Netzwerksegment ausführt, ist die sogenannte *Discovery*. Er fordert mit dem Befehl `GET / HTTP/1.1` alle Smart Devices im Segment auf, ihre Gerätebeschreibung und jeweiligen Maschinenzustände in der definierten Beschreibung auszuliefern. Wie beschrieben kann die Aufgabe der Header-Erzeugung sowohl vom Gateway wie auch von einem verbundenen Smart Device (dem ersten, das antwortet) wahrgenommen werden.

Die Discovery ist der einzige Prozeß, in welchem beschreibende Elemente (`meta` und `label`) und Attribute von Datenpunkten ausgegeben werden (siehe Listing 3.27).

3.4.3 Geräteverknüpfung

Das Verb `CONNECT` wurde bereits in Kapitel 3.3.3.4 besprochen. Ein Smart Device prüft nicht, ob eine übergebene Event-Quelle tatsächlich existiert. Es wird nur die Struktur einer jeden übergebenen Zeile der Nutzdaten validiert.

Sofern das Format des Befehls dem Schema entspricht, wird für den oder die angegebenen Datenpunkte jeweils eine Verknüpfung bis zum Erreichen des Limits der Speicherplätze angelegt. Sind etwa nur zwei Speicherplätze für einen Datenpunkt verfügbar, aber drei

Quellen in der CONNECT-Anweisung angegeben, so werden nur die ersten beiden im Smart Device übernommen.

Dennoch ist die Antwort 200 OK, da mindestens eine Verknüpfung erfolgreich angelegt wurde. Die Antwort in den Nutzdaten enthält die angelegten Verknüpfungen und die Anzahl der noch freien Speicherplätze in jedem Datenpunkt, wie in Listing 3.28 dargestellt (Grundlage ist noch immer die Gerätebeschreibung aus 3.27).

```
>>> CONNECT /fancontrol_000030A0/Ventilator/Umschalteingang HTTP/1.1
>>> Content-Length: 41
>>>
>>> /schalter0C1A/Mittlere-Taste/Kurzer-Druck

<<< HTTP/1.1 200 OK
<<< Content-Length: 335
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>
<<< <fancontrol_000030A0>
<<<   <Ventilator>
<<<     <Umschalteingang>
<<<       <listensTo>/schalter0312/Obere-Taste/Kurzer-Druck</listensTo>
<<<       <listensTo>/schalter0C1A/Mittlere-Taste/Kurzer-Druck</listensTo>
<<<     </Umschalteingang>
<<<   </Ventilator>
<<< </fancontrol_000030A0>
```

Listing 3.28: Darstellung der Antwort (Nutzdaten) auf einen erfolgreichen CONNECT-Befehl. In der reduzierten XML-Antwort sind die nach Ausführung des Befehls verknüpften Quellen angegeben.

Wären noch keine Speicherplätze belegt gewesen, so würde außerdem das Attribut `free-event-slots` zusätzlich ausgegeben, dargestellt in Listing 3.29.

```
>>> CONNECT /fancontrol_000030A0/Ventilator/Umschalteingang HTTP/1.1
>>> Content-Length: 41
>>>
>>> /schalter0C1A/Mittlere-Taste/Kurzer-Druck

<<< HTTP/1.1 200 OK
<<< Content-Length: 283
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>
<<< <fancontrol_000030A0>
<<<   <Ventilator>
<<<     <Umschalteingang free-event-slots="1">
<<<       <listensTo>/schalter0C1A/Mittlere-Taste/Kurzer-Druck</listensTo>
<<<     </Umschalteingang>
<<<   </Ventilator>
<<< </fancontrol_000030A0>
```

Listing 3.29: Darstellung der Antwort (Nutzdaten) auf einen erfolgreichen CONNECT-Befehl, wenn noch keine Speicherplätze belegt waren. In der reduzierten XML-Antwort sind die Anzahl der freien Speicherplätze sowie die nach Ausführung des Befehls verknüpfte Quelle angegeben.

Eine Anwendung ermittelt aus den impliziten und expliziten Informationen der Antwort den Erfolg eines Befehls.

3.4.4 Entfernen von Verknüpfungen

Analog zum Hinzufügen von Verknüpfungen verhält sich das Entfernen selbiger. Über das Verb DISCONNECT werden die zu entfernenden Quellen an jeden betroffenen Datenpunkt gesendet. Es ist aus Implementierungsgründen nicht möglich, den Platzhalter (Asterisk) in einer zu entfernenden Quelle zu verwenden. Listing 3.30 ergänzt das Beispiel aus Listing 3.29 und demonstriert die Verwendung des Verbs.

```
>>> DISCONNECT /fancontrol_000030A0/Ventilator/Umschalteingang HTTP/1.1
>>> Content-Length: 41
>>>
>>> /schalter0C1A/Mittlere-Taste/Kurzer-Druck

<<< HTTP/1.1 200 OK
<<< Content-Length: 188
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>
<<< <fancontrol_000030A0>
<<<   <Ventilator>
<<<     <Umschalteingang free-event-slots="2"></Umschalteingang>
<<<   </Ventilator>
<<< </fancontrol_000030A0>
```

Listing 3.30: Darstellung der Antwort (Nutzdaten) auf einen erfolgreichen DISCONNECT-Befehl, wenn noch keine Speicherplätze belegt waren. In der reduzierten XML-Antwort ist die Anzahl der nach Ausführung des Befehls freien Speicherplätze angegeben. Wäre entgegen der Darstellung im Listing noch eine Quelle verknüpft, würde sie analog zu Listing 3.29 mit `listenTo`-Element angegeben.

Der Vollständigkeit halber gibt Listing 3.31 ein Beispiel für die Entfernung einer Verknüpfung aus zwei Aktoren gleichzeitig an. Der Mechanismus der verteilt erzeugten Antwort kommt hierbei zum Einsatz.

```
>>> DISCONNECT /Ventilator/Umschalteingang HTTP/1.1
>>> Host: fancontrol_000030A0
>>> Host: fancontrol_000030A1
>>> Content-Length: 41
>>>
>>> /schalter0C1A/Mittlere-Taste/Kurzer-Druck

<<< HTTP/1.1 200 OK
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>
<<< <bus>
<<<   <fancontrol_000030A0>
<<<     <Ventilator>
<<<       <Umschalteingang free-event-slots="2"></Umschalteingang>
<<<     </Ventilator>
<<<   </fancontrol_000030A0>
```

```
<<<    <fancontrol_000030A1>
<<<      <Ventilator>
<<<        <Umschalteingang free-event-slots="1">
<<<          <listensTo>/key0x991213/btn05/pressed</listensTo>
<<<        </Umschalteingang>
<<<      </Ventilator>
<<<    </fancontrol_000030A1>
<<< </bus>
```

Listing 3.31: Ergänzend zu Listing 3.30 wird hier die Entfernung der Verknüpfung mit dem Schalter aus mehreren Clients dargestellt. Jeder Client generiert eine eigene Antwort. Das zweite Device besitzt weiterhin eine Verknüpfung mit einem anderen Schalter.

Weder CONNECT noch DISCONNECT geben die aktuellen Maschinenzustände oder andere beschreibende Attribute und Elemente aus.

3.4.5 Gerätesteuerung

Die verfügbaren Verben wurden in Kapitel 3.3.3 bereits teilweise besprochen. Um eine Verarbeitung und Implementierung auf Smart Devices so einfach wie möglich zu halten, wird auf XML-Verarbeitung von Nutzdaten auf einem Smart Device verzichtet.

URL und Header werden wie beschrieben zur Bezeichnung der betroffenen Geräte und optional Datenpunkte verwendet. Die folgenden Ausführungen ergänzen Kapitel 3.3.3 um eine Sichtweise auf Funktionalitäten und erweiterte, vollständige Anwendungsbeispiele.

3.4.5.1 Adressierung von Geräten

Zunächst wird die Adressierung von Geräten zusammenfassend dargestellt. Listings 3.32 bis 3.34 geben die jeweils gültigen Nachrichtenformate an.

```
>>> $VERB$ /[$DEVICE-ID$|*]/$Dienst$/$Datenpunkt$ HTTP/1.1
```

Listing 3.32: Adressierung eines oder aller Smart Devices über den URL-Pfad.

```
>>> $VERB$ /$Dienst$/$Datenpunkt$ HTTP/1.1
>>> Host: [$DEVICE-ID$|*]
```

Listing 3.33: Adressierung eines oder aller Smart Devices über Host-Header.

```
>>> $VERB$ /$Dienst$/$Datenpunkt$ HTTP/1.1
>>> Host: $DEVICE1-ID$
>>> Host: $DEVICE2-ID$
>>> Host: $DEVICE3-ID$

>>> $VERB$ /$Dienst$/$Datenpunkt$ HTTP/1.1
>>> Host: $DEVICE1-ID$,$DEVICE2-ID$,$DEVICE3-ID$
```

Listing 3.34: Adressierung mehrerer Smart Devices über Host-Header. Anmerkung: das untere dargestellte Format ist in der vorhandenen Implementierung nicht unterstützt.

In der vorliegenden Implementierung ist die angegebene HTTP-Version nicht relevant; der weitere Text hinter dem URL-Pfad wird bis zum Zeilenumbruch (`\r\n`) ignoriert. Die beschriebene Adressierungsart gilt für alle fünf definierten Verben.

Eine Adressierung von Geräten darf, wie in den Listings dargestellt, nur entweder über den URL-Pfad oder über den `Host`-Header erfolgen. Eine gemischte Verwendung ist nicht zulässig, da bei gesetztem `Host`-Header der erste Teil des URL-Pfades bereits als Dienstename (siehe nächstes Unterkapitel) interpretiert wird.

3.4.5.2 Application Level Manycast: Adressierung von Diensten und Datenpunkten

Maschinenzustände werden auf Smart Devices in einer Hierarchiestufe, dem Dienst, gruppiert. Ein Dienst enthält mindestens einen Datenpunkt, trägt aber außer einen Namen keine weitere Information. Dienste können, da sie keinen Maschinenzustand abbilden, keine Werte empfangen oder ausliefern. Dafür ist eine Adressierung von enthaltenen Datenpunkten notwendig.

Dennoch können Datenpunkte anhand ihrer Dienste gefiltert adressiert werden. Die Methodik lehnt sich an die der Geräteadressierung an, wodurch entweder genau ein Dienst, mehrere oder alle Dienste auf einem Gerät angesprochen werden können. Gleiches gilt für Datenpunkte. Damit ergeben sich insgesamt 9 Kombinationen der Adressierung von Maschinenzuständen auf einem einzelnen Gerät.

Der Autor bezeichnet diese Form der Adressierung als *Application Level Manycast*.

```
>>> [POST|CONNECT|DISCONNECT] /[$Dienst$|*]/[$Datenpunkt$|*] HTTP/1.1
>>> Host: $DEVICE-ID$
>>> Content-Length: $Oktettlänge($NEUER_WERT$)$
>>>
>>> $NEUER_WERT$
>>> [optional weitere Zeilen bei CONNECT und DISCONNECT]
```

Listing 3.35: Schreiben eines singulären Wertes mehrere ihre Datenpunkte. Sowohl für den Dienst-Namen wie auch für die Datenpunkt-Ebene kann der Asterisk substituiert werden und spricht auf der Ebene alle vorhandenen Instanzen auf einem Gerät an.

Die kompakte Form der HTTP-Nachricht wird verwendet, wenn nur ein einziger oder alle Dienste sowie ein einziger oder alle Datenpunkte adressiert werden, da in diesem Fall der Asterisk als Platzhalter eingesetzt werden kann. Listing 3.35 demonstriert das Format. Der übergebene neue Wert wird an alle benannten Datenpunkte übermittelt.

Für das Verb `EVENT` ist nur die Adressierung über den URL-Pfad zulässig, wie in Listing 3.36 dargestellt. Der Pfad gibt dabei die *Quelle* eines Events an. Event-Nachrichten werden nicht bestätigt; ein Ausbleiben eines Events wird durch die Anwendung festgestellt und mit Unicast-Kommunikation gegebenenfalls fehlerbehandelt.

```
>>> EVENT /[$DEVICE-ID$]/[$Dienst$]/[$Datenpunkt$] HTTP/1.1
>>> Content-Length: $Oktettlänge($GEÄNDERTER_WERT$)$
>>>
>>> $GEÄNDERTER_WERT$
```

Listing 3.36: Broadcast eines singulären Wertes durch Events.

Sollen dagegen unterschiedliche Werte an unterschiedliche Datenpunkte übermittelt werden, so wird das erweiterte Format verwendet. Es ist nur für das Verb POST zulässig (bereits aus der Übermittlung von HTML-Formularen bekannt) und in Listing 3.37 abgebildet.

Die Abarbeitung der übergebenen Werte findet sequentiell in der Reihenfolge des Eingangs statt. Es ist daher zulässig (wenn auch nicht sinnvoll), mehrmals die gleichen Datenpunkte mit unterschiedlichen Werten zu belegen. Der letzte übergebene Wert bleibt erhalten. Insbesondere Datenpunkte, die unterschiedliche Repräsentationen des selben Maschinenzustandes widerspiegeln (z.B. RGB- und HSV-Farbrepräsentation einer mehrfarbigen Lichtquelle), sind beim Beschreiben zu beachten.

```
>>> POST / HTTP/1.1
>>> Host: $DEVICE-ID$
>>> Content-Length: $Oktettlänge($Nutzdaten$)$
>>>
>>> [$Dienst$|*].[$Datenpunkt$|*]=$NEUER_WERT$[&[$Dienst$|*].[$Datenpunkt$
    |*]=$NEUER_WERT$]
```

Listing 3.37: Schreiben mehrerer Werte an mehrere Datenpunkte in einer Nachricht. Sowohl für den Dienst-Namen wie auch für die Datenpunkt-Ebene kann der Asterisk substituiert werden und spricht auf der Ebene alle vorhandenen Instanzen an. Dienst- und Datenpunktname werden durch den Punkt getrennt. Der zu schreibende Wert folgt hinter einem Gleichheitszeichen. Zeichenkettenwerte sind URL-codiert zu übertragen. Die Verknüpfung mehrerer zu schreibender Werte geschieht durch das Ampersand.

Insbesondere das erweiterte Nachrichtenformat gestattet kombinierte Anweisungen in einer einzigen Nachricht bei geschickt gewähltem Benennen von Diensten und Datenpunkten. Einige Anwendungsbeispiele basierend auf der Gerätebeschreibung des Ventilators in Listing 3.27 sind in den folgenden Listings 3.38 bis 3.40 angegeben.

```
>>> POST / HTTP/1.1
>>> Host: *
>>> Content-Length: 21
>>>
>>> *.Oszillationsstufe=0
```

Listing 3.38: Erstes Beispiel für Application Level Manycast. Die Oszillation der beiden Achsen aller gleich modellierten Lüfter –auch unterschiedlicher Hersteller– wird gestoppt.

```
>>> POST / HTTP/1.1
>>> Host: *
>>> Content-Length: 32
>>>
>>> *.Oszillationsstufe=0&*.Winkel=0
```

Listing 3.39: Zweites Beispiel für Application Level Manycast. Die Oszillation der beiden Achsen aller gleich modellierten Lüfter – auch unterschiedlicher Hersteller – wird gestoppt; gleichzeitig wird die Neutralstellung der beiden Achsen angefahren.

```
>>> POST / HTTP/1.1
>>> Host: fancontrol_000030A0
>>> Content-Length: 5
>>>
>>> *.*=0
```

Listing 3.40: Drittes Beispiel für Application Level Manycast. Die adressierte Maschine wird so ausgeschaltet. Alle Datenpunkte in allen Diensten, die beschreibbar sind, erhalten den Wert 0. Da die `Kontrast`stufe keine 0 akzeptiert, bleibt ihr Wert erhalten. Gleiches gilt für `Drehzahl` (nicht beschreibbar) und `Umschalteingang` (akzeptiert nur den Wert 1).

Herstellern sei empfohlen, logisch ähnliche Funktionsgruppen auch gleich zu benennen, um ein gemeinsames Adressieren zu vereinfachen. Vor Verwendung des Application Level Manycast sollten die Gerätebeschreibungen der adressierten Geräte vom Anwender konsultiert werden, um die Folgen eines Befehls abzusehen.

Wie in den Listings dargestellt, darf eine Adressierung von Datenpunkten und Diensten nur entweder über den URL-Pfad oder über Application Level Manycast erfolgen.

3.4.5.3 Reduzierte Antwort auf Steuerungsbefehle

Ein Smart Device liefert auf `GET`- und `POST`-Anfragen, die keine Discovery-Anfragen darstellen, nur eine reduzierte Antwort aus. Die Nachrichtengröße wird so verringert. In einer reduzierten Antwort sind keine Metadaten, keine Event-Verknüpfungen, keine Beschriftungen und keine Attribute enthalten. Es werden alle angefragten oder geänderten Datenpunkte an den Client in der bereits eingeführten Struktur der XML-Gerätebeschreibung zurückgeliefert.

Listings 3.41 bis 3.43 ergänzen die Beispiele des vorigen Kapitels; die Grundlage bildet erneut die beispielhafte Gerätebeschreibung aus Listing 3.27.

```
<<< <fancontrol_000030A0>
<<<   <X-Achse>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
<<<   </X-Achse>
<<<   <Z-Achse>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
<<<   </Z-Achse>
<<< </fancontrol_000030A0>
```

Listing 3.41: Antwort des Smart Devices auf die Anfrage aus Listing 3.38.

```
<<< <fancontrol_000030A0>
<<<   <X-Achse>
<<<     <Winkel>0</Winkel>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
<<<   </X-Achse>
<<<   <Z-Achse>
<<<     <Winkel>0</Winkel>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
```

```
<<<     </Z-Achse>
<<< </fancontrol_000030A0>
```

Listing 3.42: Antwort des Smart Devices auf die Anfrage aus Listing 3.39. Beide geänderten Datenpunkte in beiden Services werden ausgegeben.

```
<<< <fancontrol_000030A0>
<<<   <X-Achse>
<<<     <Winkel>0</Winkel>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
<<<   </X-Achse>
<<<   <Z-Achse>
<<<     <Winkel>0</Winkel>
<<<     <Oszillationsstufe>0</Oszillationsstufe>
<<<   </Z-Achse>
<<<   <Ventilator>
<<<     <Stufe>0</Stufe>
<<<   </Ventilator>
<<<   <Display-Einstellungen>
<<<     <Helligkeit>0</Helligkeit>
<<<   </Display-Einstellungen>
<<< </fancontrol_000030A0>
```

Listing 3.43: Antwort des Smart Devices auf die Anfrage aus Listing 3.40. Alle beschreibbaren Datenpunkte erhielten den Wert 0; sofern dieser im zulässigen Wertebereich liegt, wurde er übernommen und ist in der Antwort enthalten.

Die Ausgabeform XML wurde gewählt, da sich Dokumente im Gegensatz zu z.B. JavaScript Object Notation (JSON) mit Hilfe von Extensible Stylesheet Language Transformation (XSLT) einfacher transformieren und mit Hilfe der X-Path-Sprache auch einfacher abfragen lassen. Nachteilig wirkt sich eine höhere Nachrichtengröße aus, die hauptsächlich durch die schließenden Elementnamen verursacht wird. Bei JSON wird ein Element mit geschweifter Klammer geschlossen und benötigt daher weniger Übertragungsvolumen.

3.4.5.4 Relative Werte in Steuerungsbefehlen

SHDP ermöglicht das Steuern von numerischen Datenpunkten neben absoluten auch über relative Werte. Dazu wird der übermittelte Wert vom Gerät zum aktuellen Zustand addiert oder von diesem subtrahiert und anschließend mit bereits beschriebenem Mechanismus gegen die Beschränkungen evaluiert.

Im Falle einer Über- oder Unterschreitung der Beschränkungen wird im Unterschied zu einem absoluten Wert die definierte Grenze als neuer Wert verwendet („clamping“). Relative Werte bewirken daher immer eine Zustandsänderung, außer, der aktuelle Wert befindet sich bereits an einer der beiden Grenze.

Numerische Werte können vorzeichenbehaftet sein und auch – für absolute Werte – so übermittelt werden. Daher sind relative Werte entsprechend zu kennzeichnen, indem das Vorzeichen des Wertes in Anlehnung an die *C*-Notation zum Inkrementieren oder Dekrementieren (`++i;` bzw. `--i;` bei gegebener numerischer Ganzzahlvariable `i`) doppelt

gesetzt wird. Ein einziges Vorzeichen wird nicht als relativer Wert interpretiert.

Listing 3.44 und 3.45 demonstrieren jeweils ein Beispiel mit entsprechender Antwort.

```
>>> POST / HTTP/1.1
>>> Host: fancontrol_000030A0
>>> Content-Length: 21
>>>
>>> Ventilator.Stufe=++10

<<< [HTTP- und XML-Header nicht gezeigt.]
<<< <fancontrol_000030A0>
<<<   <Ventilator>
<<<     <Stufe>85</Stufe>
<<<   </Ventilator>
<<< </fancontrol_000030A0>
```

Listing 3.44: Zu einem Datenpunkt eines Gerätes wird ein Wert addiert. Als Grundlage dient wiederum die Gerätebeschreibung aus Listing 3.27.

```
>>> POST / HTTP/1.1
>>> Host: fancontrol_000030A0
>>> Content-Length: 22
>>>
>>> Ventilator.Stufe=--200

<<< [HTTP- und XML-Header nicht gezeigt.]
<<< <fancontrol_000030A0>
<<<   <Ventilator>
<<<     <Stufe>0</Stufe>
<<<   </Ventilator>
<<< </fancontrol_000030A0>
```

Listing 3.45: In diesem Beispiel wird der Ventilator deaktiviert. Der bei absoluter Übertragung unlässige Wert -200 ist bei relativen Werten aufgrund des *Clampings* gestattet.

Gegen eine in Programmiersprachen üblichere Variante der Kombination mit dem Gleichheitsoperator (`+=` bzw. `-=`) sprechen zwei Gründe:

- Das Minuszeichen ist ein gültiges Zeichen für Elementnamen. Bei einem beispielhaften Kommando `Testknoten-=10` ist für den Anwender nicht direkt ersichtlich, ob ein relativer (`--10`) oder ein absoluter (`=10`) Wert an den `Testknoten-` geschrieben wurde, da das Minuszeichen auch Teil des Namens sein kann.
- Eine Adressierung von Datenpunkten über den URL-Pfad muß ebenfalls unterstützt werden. In diesem Falle tritt im Nutzdatenfeld nur der neue (relative) Wert ohne Gleichheitszeichen auf.

Absolute Werte werden mit einfachem Vorzeichen angegeben, wobei bei positiven Werten das `+` Zeichen auch entfallen kann. Es ergeben sich daher insgesamt fünf Kombinationen gültiger Werteangaben:

- `Wert`. Positiver, absoluter Wert. Maschinenzustand auf diesen Wert setzen, wenn er innerhalb der zulässigen Grenzen liegt.

- `+Wert`. Positiver, absoluter Wert. Maschinenzustand auf diesen Wert setzen, wenn er innerhalb der zulässigen Grenzen liegt.
- `++Wert`. Relativer Wert. Momentanen Maschinenzustand mit übergebenem Wert beaufschlagen und das Minimum von beaufschlagtem Wert und definierter oberer Grenze als neuen Maschinenzustand setzen. Ist keine obere Grenze gesetzt, müssen lediglich Überläufe verhindert werden.
- `-Wert`. Negativer, absoluter Wert. Maschinenzustand auf diesen Wert setzen, wenn er innerhalb der zulässigen Grenzen liegt.
- `--Wert`. Relativer Wert. Von momentanem Maschinenzustand den übergebenen Wert abziehen und das Maximum von berechnetem Wert und definierter unterer Grenze als neuen Maschinenzustand setzen. Ist keine untere Grenze gesetzt, müssen lediglich Überläufe verhindert werden.

3.4.6 Modellierung ereignisorientierter Kommunikation

Abschließend werden Gedanken zur Modellierung von Smart Devices und der Verknüpfung ihrer Datenpunkte notiert. Für die Ausführungen dient die Steuerung einer Lampe, bei der auch die Helligkeit über einen oder mehrere Schalter verändert werden kann, als Beispiel.

3.4.6.1 Konfiguration von Event-Quellen

Es ist ratsam, Event-Quellen durch den Benutzer konfigurierbar zu gestalten. Bei einfachen Geräten ohne eigene Benutzerschnittstelle kann dies auch über das beschriebene Protokoll erfolgen, siehe Listing 3.46.

```
<beispiel_taster_0AF2>
  <meta type="deviceName">Testtaster.</meta>
  <Taster>
    <Kurzer-Druck type="float" access="r" min="0.0" max="1.0">0</Kurzer-
        Druck>
    <Langer-Druck type="float" access="r" min="0.0" max="0.01">0</Langer-
        Druck>
  </Taster>
  <Konfiguration>
    <Kurzer-Druck-Wert type="float" access="rw" min="1.0" max="100.0">1.0
        </Kurzer-Druck-Wert>
    <Langer-Druck-Relativwert type="float" access="rw" min="-100.0" max="
        100.0">0.01</Langer-Druck-Relativwert>
    <Langer-Druck-Verzoegerung type="int" access="rw" min="-1" max="1000"
        unit="ms" >20</Langer-Druck-Verzoegerung>
  </Konfiguration>
</beispiel_taster_0AF2>
```

Listing 3.46: Die Beschreibung zeigt einen einfachen Taster, dessen Sendewerte passend zu den Event-Senken konfiguriert werden können. Ein Wert von -1 der Verzögerung bedeutet „keine Wiederholung“; damit kann auch der lange Tastendruck singulär verwendet und mit langem und kurzem Druck jeweils eine unterschiedliche Lampe nur geschaltet werden.

Der Taster ist so konfiguriert, daß bei einem kurzen Druck der Wert 1,0 und beim Loslassen (unter Umständen implizit konfiguriert) eine 0 gesendet wird. Durch die Anpassung des Sendewerte kann aber auch ein im Rahmen beliebiger anderer Wert eingestellt werden, um eine andere Modellierung des Ziels zu unterstützen. Beispielsweise könnte ein fiktiver Umschalteingang einen Wert von 23 erfordern, was vom Taster unterstützt wird.

```
<beispiel_taster_0AF2>
  <meta type="deviceName">Testtaster.</meta>
  <Taster>
    <Kurzer-Druck type="float" access="r" min="0.0" max="100.0">0</Kurzer-
        Druck>
    <Langer-Druck type="float" access="r" min="0.0" max="1.0">0</Langer-
        Druck>
  </Taster>
  <Konfiguration>
    <Kurzer-Druck-Wert type="float" access="rw" min="1.0" max="100.0">
        100.0</Kurzer-Druck-Wert>
    <Langer-Druck-Relativwert type="float" access="rw" min="-100.0" max="
        100.0">1.0</Langer-Druck-Relativwert>
    <Langer-Druck-Verzoegerung type="int" access="rw" min="-1" max="1000"
        unit="ms" >20</Langer-Druck-Verzoegerung>
  </Konfiguration>
</beispiel_taster_0AF2>
```

Listing 3.47: Das Listing zeigt die Konfiguration des Schalters, der für den Wertebereich 0-100 eingestellt wurde. Zu beachten ist, daß auch der `max`-Wert des `Kurzer-Druck`-Datenpunktes angepaßt wurde.

Bei Geräten mit mehr Schaltzuständen, z.B. einem Doppeltaster, könnte ein Zustand zum Ein- und der andere Zustand zum Ausschalten verwendet werden. Der Taster würde dann nur bei Druck einen Wert senden, nicht aber beim Loslassen.

3.4.6.2 Beispielhafte Modellierung: Dimmfunktion

Deutlicher tritt die Notwendigkeit bei der Modellierung einer Dimmfunktion hervor. Hersteller könnten die Helligkeitseingänge ihrer Lampen (mathematisch korrekt für Prozentwerte) mit einer Größe von 0,0 bis 1,0 modellieren; allerdings ist auch eine – für den Benutzer – intuitive Modellierung mit einem Bereich von 0 bis 100 und der Einheit *Prozent* denkbar.

Der lange Tastendruck wird mit dem Helligkeitseingang der Lampe verknüpft. Durch relative Werte kann dann sowohl ein Wertebereich von 0,0-1,0 (Listing 3.46) wie auch von 0-100 (Listing 3.47) abgedeckt werden, indem der im Beispiel angegebene Wert `Langer-Druck-Relativwert` entsprechend angepaßt wird. Außerdem kann durch den Relativwert und die Verzögerung, gesetzt in `Langer-Druck-Verzoegerung`, die Geschwindigkeit des Dimmvorgangs angepaßt werden. Für unterschiedliche Taster im Raum könnten auch unterschiedliche Geschwindigkeiten definiert werden, so daß beispielsweise der Taster nahe der Eingangtüre schneller und mit weniger Stufen steuert als einer am Sofa.

Die Dimmrichtung (heller oder dunkler) muß im Gerät abgebildet und gespeichert werden; ein Einzelschalter kennt den Zustand der Lampe nicht.

3.4.6.3 Schlußbemerkungen

Die vorangegangenen Listings verdeutlichen auch, daß eine Gerätebeschreibung nicht konstant sein muß. Je nach Funktionslogik des Smart Devices hängt seine Beschreibung vom Betriebszustand ab. Soll eine Änderung der Beschreibung anderen Devices bekanntgegeben werden, muß dies über ein eigenes Event realisiert werden. Die Modellierung hiervon ist allerdings nicht standardisiert und muß vom Entwickler nach eigenen Anforderungen erstellt werden. In jedem Falle aber ist nach Änderung einer Gerätebeschreibung ein Discovery-Prozeß notwendig. Es besteht keine Möglichkeit, nur die neuen Konfigurationsdaten in Erfahrung zu bringen.

Explizit hat sich der Autor gegen eine ReST-kompatible Abbildung über `PUT` und `DELETE`-Methoden und URL-Mapping auf die `listensTo`-Elemente entschieden. Die Verwendung eigens definierter Verben erscheint semantisch plausibler. Immerhin betreffen die Operationen bestimmte Datenpunkte. Ihre Verknüpfungen mit Event-Quellen sind nicht als eigenständige Teile zu sehen. Außerdem wird so eine Abgrenzung zu ReST vorgenommen.

Weiterhin wäre eine Abbildung auch nur schwer möglich, da bei mehreren `listensTo`-Elementen der Inhalt nicht mehr über den URL-Pfad abgebildet werden kann. Auch hier müßte eine eigene Semantik zwischen Smart Devices und Clients definiert werden.

3.5 Integration in bestehende Heimnetze

Gängige Heimnetze bestehen, wie in Abbildung 1.2 dargestellt, aus einem Internet-Gateway und mehreren Teilnehmern innerhalb des Netzes. Eine Vernetzung von (PCs, Drucker, Tablets, etc.) Geräten erfolgt in der Regel über kabelgebundenes Ethernet oder drahtlose Techniken wie 802.11a/b/g/n WLAN. Sogenannte IP-Gateways übernehmen eine Medienwandlung zwischen dem proprietären Bus der Smart Devices und dem Heimnetz. In einem angenommenen Szenario mit mehreren verschiedenen Smart Device Netzwerksegmenten bestehen zwei Herausforderungen:

- *Anpassen unterschiedlicher Datenraten.* Übertragungsraten von Smart Devices sind in der Regel um Größenordnungen geringer als die von Ethernet oder WLAN; typische Raten liegen hier bei $100\,\mathrm{Mbit\,s^{-1}}$, während auf einem KNX-TP-Segment eine Rate von $9{,}6\,\mathrm{kbit\,s^{-1}}$ vorliegt. Sinnvolle Regeln zur Verarbeitung und Weiterleitung möglichst aller vom höherkapazitiven Netz eintreffenden Nachrichten sind zu definieren, insbesondere wenn dort mehr als eine Nachrichtenquelle (z.B. mehrere Steuerungsrechner) existiert.

- *Filtern unwichtiger Informationen.* In gängigen Heimnetzen werden von vielen unterschiedlichen Diensten, insbesondere im Windowsumfeld, Broadcasts für das lokale Segment versendet. Ein Gateway muß daher geeignete Filtermöglichkeiten besitzen, um nur dem jeweiligen Anwendungsprotokoll entsprechende Nachrichten auch an das Smart Device Segment weiterzuleiten.

3.5.1 Medienkopplung

Zunächst sind die einzelnen Segmente von Smart Device Netzwerken physikalisch an das IP-Netz angebunden werden. Ein Gateway-Gerät terminiert dazu den proprietären Bus und bietet eine Schnittstelle ins Heimnetz.

Grundsätzlich kann dies zum einen über ein eigenständiges Gerät geschehen, das parallel zu anderen IP-Teilnehmern im Ethernet-Segment integriert ist und sich wie ein reguläres Netzwerkgerät verhält. Die andere Möglichkeit besteht in einer Erweiterung eines vorhandenen Internet-Gateways in Form eines Zusatzmoduls. Eine ausführliche Diskussion findet in Kapitel 4 statt.

3.5.2 Vermittlung und Transport

Unabhängig von der physikalischen Anbindung müssen Regeln auf IP-Ebene für Vermittlung und Transport definiert werden. Da das IP-Netz eine um ein Vielfaches höhere Übertragungskapazität bietet, eignet sich ein Transport über UDP-Broadcasts. Auch die maximalen Telegrammgrößen aus Kapitel 2 von ungefähr 270 B finden in einer UDP-Nachricht vollständig Platz.

Zu beachten ist aber, daß Broadcasts nicht automatisch auf andere (Sub-)Netze umgesetzt werden. Für das Betriebssystem OpenWRT existiert die Zusatzsoftware `udp-broadcast-relay` [52], das auf allen Netzwerkschnittstellen einen bestimmten UDP-Port abhört und eintreffende Telegramme auf definierte anderen Schnittstellen unverändert umsetzt.

3.5.3 Anwendungsschicht

Sofern innerhalb eines Heimnetzes mehrere Segmente existieren (etwa, wenn mehrere Hersteller unterschiedliche Übertragungstechniken einsetzen und sich entsprechend mehrere Segmente ergeben) und per IP-basiertem Netz zusammengeschlossen werden, so ist für Gateways – unabhängig vom Ort ihrer Implementierung – das folgende Verhalten definiert:

- *EVENT-Nachrichten* werden in beide Richtungen per Broadcast weitergeleitet. Für die IP-Schnittstelle bedeutet das einen UDP-Broadcast auf einem definierten Port, in der Implementierung des Autors 1045.

- *Steuerungsanfragen* von Clients werden im IP-Netz unicast an den Absender zurückgesendet. Bei Verwendung von TCP ist eine Zuordnung über die Verbindung möglich. Bei UDP muß die Zuordnung über Absender-IP und den Absender-UDP-Port erfolgen.

Zwischengeschaltete Medienwandler und Gateways verhalten sich das Anwendungstransport- und das Anwendungsprotokoll betreffend transparent und leiten die jeweiligen Antworten der Smart Devices unverändert weiter. Smart Devices nehmen alle Aufgaben der Anwendungsschicht im Rahmen des erstellten Protokolls selbst wahr.

Mit beschriebenem Verhalten können Smart Devices beliebiger Hersteller in unterschiedlichen Segmenten einerseits von Steuer-Clients angesprochen werden. Andererseits können sie dank des Event-Mechanismus aber auch herstellerunabhängig kommunizieren.

Da das Anwendungsprotokoll zu HTTP kompatibel ist, kann eine Reihe von Clients und Programmiersprachen bereits ohne Modifikation verwendet werden.

3.5.4 Integration für Fremdhersteller

SHDP erkauft die große Flexibilität und Abstraktionsfähigkeit mit notwendiger Rechenleistung und im Vergleich zu binären Protokollen wesentlich größeren Übertragungsvolumina. Eine Integration auf Smart Devices bereits bestehender Produkte aus Kapitel 2 erscheint daher nicht sinnvoll. Diese Lösungen sind erprobt und in sich abgeschlossen.

Dennoch kann eine Integration des Protokolls auf den Gateways umgesetzt werden. Ein Hersteller, z.B. von in Kapitel 2 beschriebenen Produkten, hat nur zu definieren, welche Meßwerte und Befehlseingänge der Geräte über das Anwendungsprotokoll zu exponieren und damit für eine Steuerung zu öffnen sind. An Nachrichtentransport und sonstiger Logik innerhalb der Fremdlösung muß nichts geändert werden.

3.6 Externe Parteien

Abschließend wird in diesem Kapitel basierend auf vorigen Ausführungen eine Möglichkeit der Einflußnahme externer Parteien, z.B. Energieversorgungsunternehmen, aufgezeigt. Ein bislang im Smart Grid-Szenario diskutierter und technisch realisierter Anwendungsfall sieht die Verwendung der Smart Meter vor, wogegen allerdings einige Gründe sprechen:

- *Verbreitung von Smart Metern.* Damit eine Steuerung durch Smart Meter flächendeckend umgesetzt werden kann, müssen diese in möglichst allen Haushalten landesweit installiert werden. Wegen Bestandsschutz bestehender Anlagen, unattraktiv hoher Invesitions- und Betriebskosten (knapp 100 € Einrichtungskosten und etwa 60 € Mietkosten pro Jahr, [53]), Mehrverbrauch an Strom, und derzeit für Endverbraucher keiner Vorteile ist diese Variante allerdings nicht absehbar.

- *Kein Standard zur Kommunikation mit Smart Devices.* Wie im Kapitel 2 gezeigt, existiert kein einheitlicher Standard zur Kommunikation mit Smart Devices.

- *Reichweite.* In größeren Wohnanlagen befinden sich Verteiler- und Zählerschrank in der Regel im Kellergeschoß. Ob und wie eine stromleitungsbasierte Kommunikation mit weiter entfernten Wohneinheiten (z.B. im 20. Stockwerk) insbesondere über mehrfache Verteiler und Zwischenverteiler funktioniert, ist ungeklärt.

- *Mangelnde Transparenz.* Es ist nicht absehbar, inwieweit ein Kunde die Kontrolle über seine Geräte einer „Black Box“ ohne Möglichkeiten der Einflußnahme übergibt. Eine offene und einsehbare Lösung, die Kunden weiterhin eine freiwillige Teilnahme ermöglicht, verspricht größere Akzeptanz durch die Endverbraucher.

Ein beispielhaftes Konzept zur Modellierung und Steuerung von Geräten basierend auf den Ausführungen dieses Kapitels der Arbeit wird daher im Folgenden entwickelt. Insbesondere auf einer freiwilligen Beteiligung des Endanwenders liegt großer Wert, da seine Akzeptanz des Systems oberste Priorität hat. Anzumerken ist allerdings, daß ein Smart Meter für Abrechnungszwecke benötigt wird. Die Menge des verbrauchten Stroms pro Zeitbereich an einem Tag (z.B. pro Stunde) muß dem Energieversorger detailliert und manipulationssicher übermittelt werden.

Die bereits erwähnten Anwendungsfälle einer Steuerung durch Energieversorgungsunternehmen sind der Ausgleich von Lastspitzen durch rechtzeitigen Energieverbrauch und eine Notfallbehandlung durch quasi-erzwungenen Lastabwurf, allerdings auf Ebene des einzelnen Haushalts.

3.6.1 Modellierung der Anwendungsschnittstelle von Smart Devices

Es wird davon ausgegangen, daß Smart Device neben der Steuerelektronik auch einen oder mehrere Hochlastverbraucher beinhaltet, deren Aktivität im Rahmen der Fernbedienung des Energieversorgers gesteuert werden soll. Bei Kühlschränken sind das beispielsweise der oder die Kompressoren, bei Öfen die Heizelemente und bei Waschmaschinen Heizelemente und Trommelmotor.

Angesprochene Hochlastverbraucher können zur Vereinfachung und teilweise wegen realer Bauweise nur binäre Zustände einnehmen: *ausgeschaltet* und *in Betrieb*. Ein aktiver Betriebszustand wiederum kann durchaus unterschiedliche Stufen besitzen (Herdplatte), das ist für dieses Kapitel aber nicht relevant.

Ein Smart Device erhält nun zwei abstrakte Schnittstellen, die innerhalb des Programmcodes geeignet auf Maschinenzustände abzubilden sind.

3.6.1.1 Notfall-Abschaltung

Zum ersten ist das ein Datenpunkt zur Notfallabschaltung des Gerätes. Der Hersteller belegt die Funktion hinter dem Datenpunkt mit Betriebszuständen, die einen minimalen Stromverbrauch zur Folge haben: Kommunikationsschnittstelle und Steuerung sind weiterhin aktiv, aber alle weiteren Module wie Lampen, Motoren, Heizelemente, etc., werden unverzüglich stromlos geschaltet.

Der Datenpunkt ist binär zu modellieren. Bei Eintritt des Notfalls wird eine 1 an ihn gesendet. Ist die Bedingung beendet, wird dies durch das Schreiben einer 0 signalisiert. Das Gerät setzt dann seinen vorigen Betriebszustand fort, wenn optional bestimmte Bedingungen erfüllt sind (z.B. ausreichend langer Stillstand eines Kompressors).

Außer der Anforderung, einen binären numerischen Zustand zu definieren, besitzt der Datenpunkt keine weiteren Erfordernisse. Im Prinzip kann jeder Hersteller den Datenpunkt unterschiedlich benennen. Daher muß eine geeignete Koppelung zum Energieversorger eingerichtet werden, womit sich das Unterkapitel 3.6.2 beschäftigt.

Eine beispielhafte Modellierung des Datenpunktes ist in Listing 3.48 dargestellt.

```
<fancontrol_000030A0>
  <meta ...>...</meta>
  <X-Achse> ... </X-Achse>
  <Z-Achse> ... </Z-Achse>
  <Ventilator> ... </Ventilator>
  <Display-Einstellungen> ...  </Display-Einstellungen>
  <SmartGrid-Kontrolle>
    <Notfall-Abschaltung type="int" access="rw" min="0" max="1">
      0
      <label from="0">Keine Abschaltung: Gerät läuft normal.</label>
      <label from="1">Minimaler Stromverbrauch: Ventilator und
          Oszillationen stoppen.</label>
    </Notfall-Abschaltung>
  </SmartGrid-Kontrolle>
</fancontrol_000030A0>
```

Listing 3.48: Modellierung des binären Datenpunktes zur Notfallabschaltung. Pro Smart Device darf nur ein Datenpunkt mit dieser Funktionalität existieren. Benennung und Struktur sind frei und bleiben jedem Hersteller überlassen. Auch eine Beschriftung mit `label`-Elementen ist optional.

Abschließend sei die Motivation für einen eigenen Datenpunkt erklärt, denn prinzipiell könnte die gleiche Funktionalität auch durch eigene, in anderen Datenpunkten exponierten Maschinenzustände realisiert werden. Der Autor hält dem entgegen, daß eben *nicht* alle erforderlichen Informationen durch Datenpunkte exponiert sein müssen:

1. Interne Zustände der Maschine (Ventile, Pumpen, Heizungen, Sensoren) sind möglicherweise nur innerhalb der Anwendungslogik verfügbar.
2. Ein Abfragen und Zurückschreiben all dieser Daten über potentiell schwache Nachrichtentransporte nimmt unnötig viel Zeit in Anspruch.

3. Ein Benutzer müßte all diese Maschinenparameter korrekt mit eigener Logik abfragen, interpretieren und passend zurückschreiben, was beliebige Fehlerquellen eröffnet.

Ein einziger singulärer Datenpunkt, der genau die erforderlichen Schritte ein- und auch wieder ausleitet, erscheint daher zweckmäßig.

3.6.1.2 Reguläre Steuerung bei Stromüberproduktion

Auch für den Fall von (regionaler) Überproduktion von elektrischem Strom, möglicherweise aufgrund von regenerativ betriebenen Kraftwerken, kann das Anwendungsprotokoll verwendet werden, um bestimmte Geräte zu aktivieren.

Wiederum ist hierfür eine binäre Entscheidung erforderlich, die allerdings diesmal im Smart Home getroffen wird. Im lokalen Netzwerk wird eine binäre Information verteilt, ob ein Energieverbrauch *gerade im Moment* im wahrsten Sinne des Wortes günstig ist (`1`) oder nicht (`0`). Jedes Gerät entscheidet anhand programmierter Anwendungslogik selbst über ein Aktivieren seiner Hochlastmodule, da bestimmte zur Entscheidung notwendige Informationen (z.B. Meßgrößen) nur lokal am Gerät selbst vorhanden sind. Für gängige Haushalsgeräte können folgende Überlegungen gelten:

- *Waschmaschine, Trockner und Spülmaschine.* Das Anwendungsszenario ist hier, daß der Anwender diese Maschinen morgens belädt und in einen Zustand „bereit“ versetzt. Gleichzeitig gibt er einen Endzeitpunkt für das Waschprogramm vor, an dem die Maschine ihr Programm spätestens beendet haben soll. Während des Tages wartet die Maschine auf den Anfang eines günstigen Fensters und startet das Programm. Das Ende des günstigen Fensters (und damit der Zeitraum insgesamt) interessiert in diesem Falle nicht, da das Programm in jedem Fall beendet werden muß und schon bei einer Minute günstigerem Strom „gespart“ wurde.
- *Kühlschrank.* Kühlschränke bieten eine flexible Nutzung des Fensters, da sie Energie bei guter Isolierung über Kälte speichern können. Mit Beginn des Fensters kann der Kompressor aktiviert werden; kann aber auch bei Erreichen der eingestellten Temperatur zur Nutzung des preiswerteren Strom weiterhin aktiviert bleiben und entsprechend tiefere Temperaturen anfahren.
- *Herd, Ofen, Mikrowelle, Kaffeemaschine, usw.* Diese Geräte sind von der Nutzung durch die Smart Grid Steuerung nicht betroffen; ihre Aktivität richtet sich nach den Bedürfnissen der Benutzer.
- *Beleuchtung.* Auch hier hat eine Steuerung durch das Smart Grid im Allgemeinen keine Bedeutung. Mit aufkommenden Light Emitting Diode (LED)-Leuchtmitteln allerdings kann ein Anwendungsfall für dimmbare Lampen konstruiert werden. Außerhalb des preisgünstigen Fensters wird die Helligkeit automatisch auf beispielsweise 75 % begrenzt; der Stromverbrauch liegt dann um einige Prozent niedriger bei fast keiner Einschränkung hinsichtlich der subjektiven Helligkeit.

Die beschriebene Modellierung bietet keine Möglichkeit der Abschätzung der Dauer des „günstigen“ Fensters. Es kann prinzipiell nach einer beliebig kurzen Zeit ohne Vorwarnung wieder durch den Energieversorger beendet werden. Daher liegt es in der Implementierung der Anwendungslogik innerhalb des Smart Devices, keinen Schaden entstehen zu lassen.

Auch bei diesem Datenpunkt ist zur Vereinfachung wie gesagt eine binäre Modellierung vorgesehen. Im Idealfall erhält jedes Hochlastmodul einen eigenen solchen Datenpunkt, so daß der Anwender stets selbst entscheiden kann, welche Teile einer Maschine ständig und unabhängig laufen sollen und welche er zur Steuerung durch Dritte freigeben will.

Die Exposition von Maschinenparametern über das Protokoll ist vom Hersteller frei definierbar, ein Beispiel in Listing 3.49 dargestellt.

```
<fridge_00017FBA5>
  <meta ...>...</meta>
  <Kuehlschrank>
    <Zieltemperatur type="float" access="rw" min="4.0" max="16.0">8.0</
        Zieltemperatur>
    <Aktuelle-Temperatur type="float" access="r" min="0.0" max="40.0">8.3
        </Aktuelle-Temperatur>
    <Nachtstrom type="int" access="rw" min="0" max="1">
      0
      <label from="0">Kein Nachtstrom: Kühlschrank funktioniert
          thermostatgesteuert.</label>
      <label from="1">Günstiger Nachtstrom: Kühlschrank kühlt bis 2°C
          unter Zieltemperatur, solange Nachtstrom aktiv ist.</label>
    </Nachtstrom>
  </Kuehlschrank>
  <Gefrierschrank>
    <Zieltemperatur type="float" access="rw" min="-28.0" max="-4.0">-18.0
        </Zieltemperatur>
    <Aktuelle-Temperatur type="float" access="r" min="-30.0" max="40.0">
        -19.1</Aktuelle-Temperatur>
    <Nachtstrom type="int" access="rw" min="0" max="1">
      0
      <label from="0">Kein Nachtstrom: Gefrierschrank funktioniert
          thermostatgesteuert.</label>
      <label from="1">Günstiger Nachtstrom: Gefrierschrank kühlt bis 6°C
          unter Zieltemperatur, solange Nachtstrom aktiv ist.</label>
    </Nachtstrom>
  </Gefrierschrank>
  <SmartGrid-Kontrolle>
    <!-- siehe voriges Listing. -->
  </SmartGrid-Kontrolle>
</fridge_00017FBA5>
```

Listing 3.49: Modellierung des Datenpunktes für günstige Energie. Mit der Bezeichnung „Nachtstrom“ ist die Funktionalität zutreffend beschrieben; die `label`-Elemente sind optional und können auch in der Dokumentation enthalten sein.

3.6.2 Verbindung mit Energieversorgungsunternehmen

Ein bisheriges Konzept sieht die Steuerung von Smart Devices über intelligente Stromzähler vor. Die Unwahrscheinlichkeit dieses Szenarios wurde bereits dargelegt, deswegen wird in diesem Unterkapitel eine Alternative unter Berücksichtigung des Smart Home Internet-Gateways beschrieben.

Das Gateway dient als Application Layer Proxy und übersetzt Informationen des EVU in Steuerbefehle für das Smart Home. Das EVU erhält somit zu keinem Zeitpunkt private Daten über die Ausstattung eines Haushalts. Stattdessen liefert es an alle Haushalte den nach bestimmten Kriterien (geographische Region, vertragliche Besonderheiten, etc.) bestimmten aktuellen Strompreis in einem wohldefinierten Format.

Über dessen Aufbau ist im Rahmen der Arbeit nur eine beispielhafte Aussage getroffen. Basierend auf Abbildung 1.2 wird für die Ausführungen dieses Kapitels von einer Struktur wie in Abbildung 3.3 ausgegangen.

3.6.2.1 Verbindung von Energieversorgungsunternehmen und Smart Home

Zunächst wird der Mechanismus der Verbindung zwischen EVU und dem Smart Home über das öffentliche Internet beschrieben. Für ein realistisches Szenario wird von einer dynamischen Einwählverbindung ausgegangen, die spätestens alle 24 Stunden getrennt wird und dabei eine neue öffentliche IP-Adresse erhält.

Ein Hintergrundprogramm auf dem Internet-Gateway ist durch den Benutzer mit vom EVU bereitgestellten Zugangsdaten („Benutzername“, „Paßwort“) parametrisiert. Sie

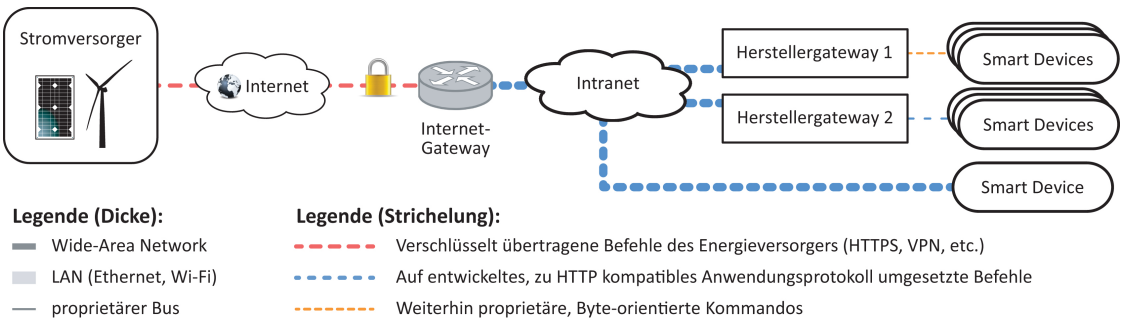


Abbildung 3.3: Anwendungsschicht im Smart Grid Szenario. Die Grafik 1.2 wird um Angaben zur Anwendungsschicht ergänzt. Befehle des Smart Grid werden über geeignet verschlüsselte Verbindungen – z.B. ein VPN – an das Internet-Gateway des Smart Home über tragen. Dieses setzt die Befehle des EVU anhand von Zuordnungstabellen, konfiguriert über die eingeführte Matrixdarstellung und Programmcode, in Anwendungsprotokollbefehle für Smart Devices um. Dabei kann ein Herstellergateway das Anwendungsprotokoll auch selbst implementieren, um „seine“ Smart Devices weiterhin über proprietäre, byte-orientierte Transportlösungen zu steuern. Ein Feedback an das Smart Grid durch Smart Devices ist nicht vorgesehen.

ermöglichen eine eineindeutige Identifizierung des Smart Homes, der zugehörigen Stromzählernummer und Vertragsdaten. Zur Verbindung mit dem EVU werden die folgenden drei Möglichkeiten betrachtet:

- *Virtual Private Network.* Eine Verbindung über ein VPN bietet dem Kunden und dem EVU einige Vorteile durch möglichen Automatismus unterschiedlicher Aufgaben.

  1. Die Verbindung wird mit dem EVU vom Client aus initiiert und selbsttätig durch „keep-alive“ Pakete aufrecht erhalten, z.B. nach Verbindungsabbruch. Das EVU besitzt stets minutiös aktuelle Informationen über den Verbindungszustand mit seinen Smart Homes.
  2. Notfallnachrichten können über IP-Broadcasts im VPN verteilt werden, so etwa über UDP-Telegramme, die versendet werden, solange ein Zustand anhält. Die Netzwerkschichten kümmern sich selbsttätig um den bestmöglichen und schnellsten Versand.
  3. Das VPN des EVU kann vom restlichen Netz des Smart Homes durch einfache Zonenregeln in der Firewall getrennt werden.
  4. Eine starke asymmetrische Verschlüsselung ist in gängigen Produkten, z.B. OpenVPN, bereits integriert. Aufwand entsteht allerdings für die Verwaltung der Zertifikate in einer Public Key Infrastructure (PKI).

  Die Nachteile dieser Lösung liegen im Verwaltungsaufwand (insbesondere bei Zertifikaten), der Endkundenbetreuung und erhöhtem Rechenaufwand durch Verschlüsselung und Aufrechterhalten der Verbindung. Ob die Zeitstempel der Einwahlen und das Loggen der öffentlichen IP-Adressen als Nachteil zu sehen sind, ist unklar. Das EVU kann aus den Daten jedenfalls keine Rückschlüsse über Smart Home Interna ziehen.

- *EVU-initiiertes Polling eines Dienstes des Smart Home Internet-Gateway.* Hierbei übermittelt der Hintergrunddienst bei Änderung der öffentlichen Adresse diese sowie die Bezeichnung und Adresse eines Callback-Dienstes (z.B. in Form einer vollständigen URL) an das EVU. Bei Notwendigkeit ruft das EVU die jeweilige über Hypertext Transport Protocol, gesichert (HTTPS) gesicherte URL auf und übergibt den neuen Wert als `GET`-Parameter. Vorteil dieser Lösung ist ein Verzicht auf VPN, allerdings muß auch hier das Zertifikatmanagement betrieben werden.

- *Kunden-initiiertes Polling eines Dienstes des EVU.* Ganz ohne Zertifikatsmanagement kommt die Lösung des „Pollings“ aus. Das Hintergrundprogramm ruft jede festgelegte Zeiteinheit (z.B. alle 10 s) eine definierte URL des EVU auf und erhält in der Antwort Informationen über eventuelle Notfälle oder geänderte Strompreise. Über die HTTP-Antwort `304 Not Modified` kann das Nachrichtenvolumen klein gehalten werden. Das beschriebene Vorgehen stellt die einfachste Form der Synchronisation dar und kann auch auf verhältnismäßig ressourcenarmen Internet-Gateways implementiert werden.

Von Webservice-basierten Protokollen wie SOAP wird wegen deren Komplexität für diesen Anwendungsfall abgeraten. Die Schnittstelle zwischen EVU und Gateways sollte so offen

und einfach wie möglich gestaltet werden. Szenarios, in denen dem EVU direkte Kontrolle über Smart Devices mittels HTTP gestattet wird, sind wegen der Datenschutzproblematik nicht betrachtet.

Abbildung 3.4 demonstriert den Nachrichtenfluß basierend auf der Lösung mit VPN.

3.6.2.2 Steuerung von Geräten über den aktuellen Strompreis

Ein Anwendungsszenario sieht ein Einschalten von Hochlastmodulen von Smart Devices im Falle einer Stromüberkapazität vor. Wann eine solche vorliegt, entscheidet alleine das Energieversorgungsunternehmen.

Über Vergleiche wie beispielsweise `(aktueller_wert==max - 0.03)` oder `(aktueller_wert==max - 0.06)` kann der Anwender detaillierte Verbrauchsschemata anlegen. Ist nur eine binäre Entscheidung notwendig, so reicht das Statement `(aktueller_wert<max)` aus.

Das Internet-Gateway entscheidet dann basierend auf vom Benutzer eingestellten Optionen, welche Geräte vom niedrigeren Strompreis profitieren sollen und übermittelt die entsprechenden (binären) Nachrichten an deren Eingänge (siehe voriges Unterkapitel). Die Geräte an sich entscheiden darauf hin über Änderungen ihrer jeweiligen Betriebszustände.

Sieht ein Gerät keine eigenen Datenpunkte zur effizienten Steuerung der Module vor, kann diese immer noch durch Programmlogik auf dem Internet-Gateway erfolgen, siehe Listing 3.50.

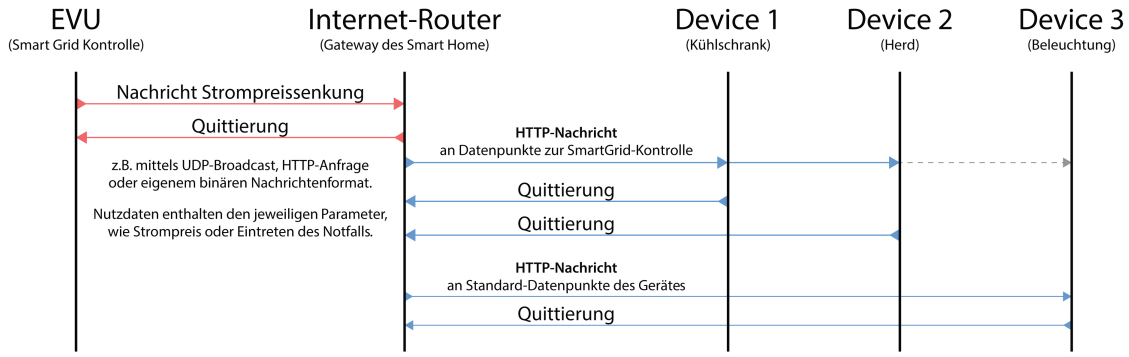


Abbildung 3.4: Steuerung von Smart Devices durch Energieversorgungsunternehmen. Nach Quittierung der Information seitens des Netzbetreibers werden die Smart Devices mit passenden Steuerbefehlen angesprochen. In zwei Schritten werden zuerst Smart Devices mit jeweiligem Kontrolleingang befehligt. Bei sinnvoller Benennung der Datenpunkte kann dies auch über Application Level Manycast geschehen. Dann werden Geräte angesprochen, die keine entsprechenden Datenpunkte besitzen, aufgrund ihrer wenig komplexen Funktionalität aber auch durch einfache benutzerdefinierte Regeln gesteuert werden können. Im Beispiel der Lampe könnte das eben die Helligkeit sein. Das EVU erhält keine Information über die Interna des Smart Homes.

```
>>> POST /*/*/Nachtstrom HTTP/1.1
>>> Content-Length: 1
>>>
>>> 1

<<< [HTTP- und XML-Header nicht gezeigt.]
<<< <bus>
<<<   <fridge_00017FBA5>
<<<     <Kuehlschrank><Nachtstrom>1</Nachtstrom></Kuehlschrank>
<<<     <Gefrierschrank><Nachtstrom>1</Nachtstrom></Gefrierschrank>
<<<   </fridge_00017FBA5>
<<<   <light_F31D>
<<<     <Ausgang1><Nachtstrom>1</Nachtstrom></Ausgang1>
<<<     <Ausgang2><Nachtstrom>1</Nachtstrom></Ausgang2>
<<<     <Ausgang3><Nachtstrom>1</Nachtstrom></Ausgang3>
<<<   </light_F31D>
<<< </bus>

>>> POST / HTTP/1.1
>>> Host: *
>>> Content-Length: 20
>>>
>>> *.Programm-Starten=1

<<< [HTTP- und XML-Header nicht gezeigt.]
<<< <bus>
<<<   <waschmaschine_9123>
<<<     <Programme><Programm-Starten>1</Programm-Starten></Programme>
<<<   </waschmaschine_9123>
<<<   <dishwasher_AD32>
<<<     <control_input><start_program>1</start_program></control_input>
<<<   </dishwasher_AD32>
<<< </bus>
```

Listing 3.50: Aktivierung von Smart Devices bei preiswerterem Strom. Geräte, die den `Nachtstrom`-Datenpunkt unterstützen, können über Application Level Manycast angesprochen werden. Alle anderen Geräte müssen mit eigenen `POST`-Anfragen in einen geeigneten Zustand versetzt werden, wie hier Wasch- und Spülmaschine.

3.6.2.3 Notfallabschaltung

Vergleichbar mit der Übermittlung des Strompreises wird über einen zweiten Nachrichtentyp die Notfallabschaltung von Smart Devices ausgelöst. Zu beachten ist, daß das Internet-Gateway auch hier als Proxy dient und der Benutzer die jeweiligen abzuschaltenden Geräte bzw. ihre Datenpunkte dort festlegt.

Auch für diesen Anwendungsfall gilt, daß zum einen bestimmte Notfalleingänge in Geräten angesprochen werden können, oder aber der Benutzer selbständig die Funktionalität und die zu beschreibenden Datenpunkt in Smart Devices auswählt. Das Ziel, so einen minimalen Stromverbrauch zu erreichen, ist dem Benutzer klar zu kommunizieren.

Auch in diesem Fall erfährt der Energieversorger keine Interna über Smart Devices der Kunden; sofern eine gewisse Zeitspanne nach Übermittlung der Notfallanweisungen noch kein ausreichender (freiwilliger) Lastabwurf erfolgt ist, muß dieser auf bisherige Weise erzwungen werden.

3.6.3 Schlußbemerkung

Ist aus irgendeinem Grund die Verbindung zwischen Internet-Gateway und Energieversorger unterbrochen, so besteht automatisch ein Rückfall auf bisherige Konzepte: Smart Devices operieren unabhängig von möglicherweise günstigerem Strom (ohne das jedoch prüfen zu können). Im Notfall muß ggf. in Zwischenverteilern des Netzbetreibers ein automatischer, netzseitiger Lastabwurf durchgeführt werden.

Letztlich ergeben sich durch das beschriebene Konzept Vorteile auf beiden Seiten für die Verbindung zwischen Smart Home und Smart Grid. Ohne Notwendigkeit für Smart Meter kann die geforderte Funktionalität realisiert werden.

4 Smart-Home: Exemplarische Umsetzung

In Kapitel 3 wurde mit SHDP ein flexibles Anwendungsprotokoll zur Steuerung von Geräten entwickelt. Seine Implementierung auf eigens entwickelter, exemplarischer Hardware wird gemeinsam mit den erforderlichen Komponenten auf dem Internet-Gateway in diesem Kapitel vorgestellt.

Zunächst werden unterschiedliche Möglichkeiten des Nachrichtentransports für die Anwendungsebene erläutert, ohne sich konkret auf ein Medium zu beschränken. Es wird insbesondere auf eine Schichtentrennung nach OSI-Modell eingegangen und die Interaktion von Geräten mit verschiedenen Anbindungen erläutert.

Nach einer Beschreibung der erforderlichen Komponenten auf dem Internet-Gateway und ihrer Konfiguration werden die erstellte Hardwareplattform und die daraus konstruierten Smart Device Demonstratoren vorgestellt, wobei insbesondere die in Kapitel 1 erarbeiteten Anforderungen unterschiedlicher Interessengruppen an eine Plattform berücksichtigt werden.

Trotz des minimalistischen Ansatzes der Hardware besteht im Smart Home ein großes Interesse an Kompatibilität. Wesentlich gilt dies auch für den Nachrichtentransport. Ein Unterkapitel beschäftigt sich daher mit dem Transport und der Verarbeitung des Internet Protokoll (IP) auf der erstellten Hardware und beschreibt die Implementierung eines darunterliegenden Unicast- und Broadcast-Protokolls für ein proprietäres Funkmedium. Anschließend wird die einfach zu parametrisierende Firmware vorgestellt.

Die Darstellung einer browserbasiert parametrierbaren Anwendungslogik für Internet-Gateway sowie Überlegungen zur IT-Sicherheit runden das Kapitel ab.

4.1 Nachrichtenrouting: Netzwerksicht und Topologie

Zunächst werden in diesem Teil zusammenfassend die unterschiedlichen Möglichkeiten des Nachrichtenroutings unter Berücksichtigung verschiedener Implementierungen der einzelnen OSI-Schichten dargestellt.

Die in Abbildung 4.1 gezeigten Smart Devices und alle beteiligten Netzwerkkomponenten spannen ein einziges virtuelles Smart Device Netzwerksegment auf, das auf Anwendungsprotokollebene die in Kapitel 3 erarbeitete Lösung implementiert. Die folgenden Unterkapitel behandeln jeweils eine Variante des Nachrichtentransports zu den Smart Devices.

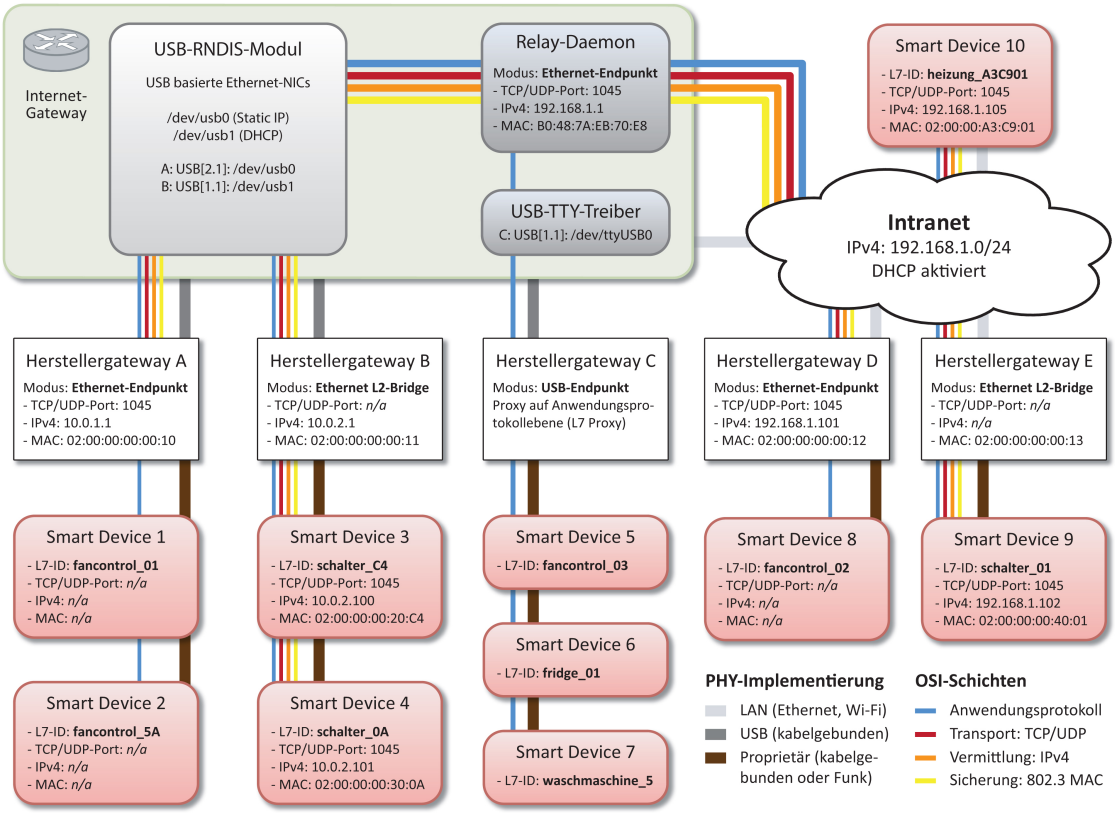


Abbildung 4.1: Darstellung eines virtuelles Smart Device Netzwerks im LAN. Die physikalische OSI-Schicht enthält mehrere unterschiedliche Implementierungen (802.3, 802.11, USB, proprietäre Lösungen) mit jeweils unterschiedlichen Datendurchsätzen und Latenzzeiten. Darüber befinden sich die aus der Netzwerktechnik bekannten Implementierungen der Schichten 2 mit 4 (802.3 Tagged MAC-Frames, IP-Vermittlung und TCP/UDP-Transport). Die Medienwandler und Ethernet-Endpunkte am Internet-Gateway werden über USB verbunden. Im Falle von Ethernetkarten werden die gesamten Frames über RNDIS getunnelt. Wie anhand der blauen Linien erkennbar, dienen alle Lösungen dem Transport des in Kapitel 3 entwickelten Anwendungsprotokolls.

Die gesamte Kommunikation findet über Broadcasts statt. Über Datenraten der proprietären physikalischen Transportsysteme ist an dieser Stelle keine Aussage getroffen. Auch bleibt offen, ob bestimmte Endpunkte und Bridges eine Filterung auf Anwendungsprotokollebene vornehmen und so die Übertragungsvolumina auf ihrem jeweiligen Teilsegment begrenzen.

Ein Discovery-Broadcast im gesamten, in Abbildung 4.1 dargestellten virtuellen Netzwerk erzeugt die in Listing 4.1 reduziert dargestellte Ausgabe der Gerätebeschreibungen. Smart Devices, die aus Sicht des Anwendungsprotokolls sowohl Gateway- wie auch Devicefunktionalität vereinen, senden ein umschließendes `<bus>`-Element.

```
1 [UDP-Broadcast auf Port 1045]
2 >>> GET / HTTP/1.1
```

```
[Antwort Gateway A (10.0.1.1:1045)]
<<< <bus>
<<<   <fancontrol_01>...</fancontrol_01>
<<<   <fancontrol_5A>...</fancontrol_5A>
<<< </bus>

[Antwort Gateway B, Smart Device 3 (10.0.2.100:1045)]
<<< <bus>
<<<   <schalter_C4>...</schalter_C4>
<<< </bus>
[Antwort Gateway B, Smart Device 4 (10.0.2.101:1045)]
<<< <bus>
<<<   <schalter_0A>...</schalter_0A>
<<< </bus>

[Antwort Gateway C (192.168.1.1:1045)]
<<< <bus>
<<<   <fancontrol_03>...</fancontrol_03>
<<<   <fridge_01>...</fridge_01>
<<<   <waschmaschine_5>...</waschmaschine_5>
<<< </bus>

[Antwort Gateway D (192.168.1.101:1045)]
<<< <bus>
<<<   <fancontrol_02>...</fancontrol_02>
<<< </bus>

[Antwort Gateway E, Smart Device 9 (192.168.1.102:1045)]
<<< <bus>
<<<   <schalter_01>...</schalter_01>
<<< </bus>

[Antwort Smart Device 10 (192.168.1.105:1045)]
<<< <bus>
<<<   <heizung_A3C901>...</heizung_A3C901>
<<< </bus>
```

Listing 4.1: Vollständiges Beispiel der Discovery für Abbildung 4.1. Für das Beispiel wird ein korrekt konfiguriertes Routing und Forwarding zwischen den Firewallzonen und Subnetzen vorausgesetzt. Dieses kann auch mit Drittherstellersoftware (z.B. `udp-broadcast-relay`) realisiert werden.

Die dargestellten Verbindungsmöglichkeiten zwischen Internet-Gateway und Smart Devices werden im Folgenden, ausgehend von „Herstellergateway A“ gegen den Uhrzeigersinn, mit ihren jeweiligen Eigenschaften beschrieben.

4.1.1 USB-Netzwerkkarte als Ethernet-Endpunkt

In dieser Konfiguration, in Abbildung 4.1 „Herstellergateway A“, wird ein Herstellergateway über USB mit dem Internet-Gateway verbunden. Das Herstellergateway emuliert eine reguläre Netzwerkkarte (Network Interface Card, NIC) und besitzt damit eine eigene

MAC-Adresse. Sie kann mit gesetztem Local-Bit, wie in Abbildung 4.1 demonstriert, auch vom Benutzer vergeben werden.

Der Host (hier das Internet-Gateway) weist der Netzwerkkarte eine eigene statische IP-Adresse zu. Ein Dynamic Host Configuration Protocol (DHCP)-Server ist in diesem Falle nicht notwendig, da das Herstellergateway selbst alle IP-Pakete terminiert.

Die Firmware des Herstellergateways extrahiert die Nutzlast der TCP- bzw. UDP-Pakete und setzt sie auf den proprietären Nachrichtentransport um. Optional kann an dieser Stelle bereits eine Filterung nicht unterstützter Headerzeilen oder falsch formatierter Anfragen erfolgen. Auch je nach Gerätebeschreibung unzulässige Werte können hier gefiltert werden. Für die beiden Transportprotokolle sind unterschiedliche Verhaltensweisen hinsichtlich der Antwort an Clients definiert:

- *UDP-Telegramme.* Die Antworten der Smart Devices werden im Herstellergateway in Telegramme verpackt und an den Absender, identifiziert durch seine IP-Adresse und den verwendeten Port, geschickt. Das Herstellergateway kann wegen des Halbduplexbetriebs des Smart Device-Segments stets nur eine „Verbindung“ behandeln, daher muß entweder eine Inspektion auf Anwendungsebene stattfinden, um das Ende einer Kommunikation durch die Smart Devices festzustellen, oder das Ende wird über einen Timer ermittelt. Erst danach können neue UDP-Telegramme verarbeitet werden. Aufgrund dieser Schwierigkeit bietet sich an, nur `EVENT`-Nachrichten über UDP zu empfangen.
- *TCP-Verbindung.* Bei diesem Transportprotokoll kann dem Stack die Anzahl der gleichzeitig zu behandelnden Verbindungen konfiguriert werden. Darüber hinausgehende Verbindungen werden in einer Warteschlange geparkt, bis entweder ein Timer überläuft oder der Server die Verbindungsanfrage annimmt. Antworten von Smart Devices werden über die bestehende Verbindung gesendet; der Stack übernimmt die richtige Adressierung des Clients automatisch.

Anzumerken ist, daß der proprietäre Nachrichtentransport eigene MAC-Adressen für jedes Smart Device besitzen kann. In diesem Szenario ist das allerdings nicht relevant, da im Smart Device Netzwerksegment nur Broadcast-Kommunikation stattfindet.

4.1.2 USB-Netzwerkkarte als Ethernet-Brücke

Eine weitere Möglichkeit ist, die USB-Netzwerkkarte lediglich zur Medienwandlung einzusetzen und Ethernet-Frames unverändert auf den proprietären Bus umzusetzen, in Abbildung 4.1 „Herstellergateway B“. Damit entfällt eine Implementierung des Ethernet-Stacks im Herstellergateway, das nur noch der korrekten Weiterleitung der Frames in beide Richtungen dient und preiswerter realisiert werden kann.

Die Netzwerkkarte bietet auch in diesem Fall die Schnittstelle für ein eigenes IP-Subnetz, das durch den DHCP-Dienst des Internet-Gateways für alle angeschlossenen Smart Devices eigene Adressen bereitstellt. Da Broadcasts an Netzwerkgrenzen terminieren, bleibt das

Smart Device Subnetz (im Beispiel: 10.0.2.0/24) von Nachrichten der regulären und um Größenordnungen leistungsfähigeren Netzwerkgeräte (Personal Computer, Tablets, etc.) unberührt.

UDP-Telegramme und TCP-Verbindungen werden von jedem Smart Device selbständig je nach Kapazität des proprietären Nachrichtentransports behandelt. Auf Ethernet-Ebene besitzt jedes Smart Device eine eigene MAC-Adresse, die, wie im vorigen Unterkapitel, mit gesetztem Local-Bit eigenständig im Netz verwaltet werden kann.

Jedes Smart Device besitzt eine eigene IP-Adresse und erscheint bei der Discovery mit eigenem Herstellergateway. Aus diesem Grund sollten Letztere möglichst wenig Bedeutung für das Anwendungsprotokoll haben. Relevant ist nur die Liste der Geräte, unabhängig vom jeweiligen Übertragungsweg.

Anzumerken ist, daß auch der proprietäre Nachrichtentransport noch einmal eigene MAC-Adressen besitzen kann, die sich von den sechs Bytes der Ethernet-MAC-Adressen unterscheiden. In diesem Fall ist eine geeignete Abbildung zu finden, damit Unicast-Übertragungen auch auf proprietärer Ebene bereits zielgerichtet stattfinden können.

Die beschriebene Architektur scheint geeignet hinsichtlich der Funktionalität und Flexibilität eines Smart Devices, da hier die Vorteile von Ethernet und IP-Adressierung in jedem Gerät zur Verfügung stehen: Bei ausreichender Kapazität lassen sich beliebige Dienste (Internet Control Message Protocol (ICMP), Network Time Protocol (NTP), DNS, etc.) direkt im Gerät nutzen.

4.1.3 USB-Gateway mit virtueller serieller Schnittstelle

Ein USB-Gerät kann mit wenig Aufwand auch eine serielle Schnittstelle emulieren. Waren dafür früher eigene Microchips (z.B. der Firma FTDI [54]) erforderlich, so besitzen gängige Mikrocontroller (MCU) heute generische USB-Peripherie. Diese kann in der Firmware mit Deskriptoren für die jeweiligen Klassen programmiert werden; das Gerät verhält sich entsprechend.

Auf dem Host ist ein Treiber, empfehlenswert als Kernelmodul, für den Betrieb von virtuellen seriellen Schnittstellen erforderlich. Dem Betriebssystem erscheinen sie als reguläre lesbare und beschreibbare Dateien, die mit gängigen Systemcalls wie `fopen()`, `fgets()`, `fputs()` und `fclose()` bearbeitet werden können. Mittels `select()` läßt sich die Verfügbarkeit von Daten zum Lesen ermitteln und eine Fehlerbehandlung reduzieren.

Um Daten zwischen IP-Netzen und der virtuellen seriellen Schnittstelle auszutauschen, ist ein Userland-Programm erforderlich. Es bindet auf TCP- und UDP-Sockets jeweils auf dem Port 1045 für alle internen Schnittstellen und terminiert dort ankommende IP-Nachrichten. Der Mechanismus zur Absendererkennung ist der selbe wie in Kapitel 4.1.1.

Da das reine Anwendungsprotokoll nicht verbindungsorientiert arbeitet, muß die virtuelle serielle Schnittstelle neben Analyse des Datenverkehrs (auf einen gesetzten `Content-Length`-Header, ein schließendes `</bus>`-Element oder das schließende Element der Geräte-ID) auch

mit einem Timer überwacht werden. Dieser wird bei Empfang eines Zeichens gestartet bzw. zurückgesetzt und löst bei Ablauf eine vollständige Übertragung („flush“) aller bislang empfangenen Daten an den IP-Kommunikationspartner aus. Der Timeout ist geeignet groß zu wählen; d.h. die Signallaufzeit und die Back-off-Zeiten des proprietären Busses müssen berücksichtigt werden. Weiterhin wird ein *flush* auch bei Erreichen der Maximum Transmission Units (MTU) des Ethernet-Segments ausgelöst.

Die beschriebene Architektur stellt die einfachste Variante der Kommunikation mit Smart Devices dar. IP-Daten werden direkt im Gateway terminiert und nur die für die Smart Home Funktionalität wichtigen Anwendungsdaten an potentielle Empfänger ausgeleitet. Es muß keine hardwarelastige Schnittstelle für Ethernet implementiert werden. Auf dem proprietären Nachrichtentransport werden für die Funktionalität überflüssige Headerinformationen (Ethernet, IP, TCP, UDP) eingespart.

4.1.4 Ethernetgerät als Ethernet-Endpunkt

Die bislang beschriebenen Lösungen eignen sich nur für Herstellergateways, die in der Nähe des Internet-Routers platziert werden können. Müssen diese jedoch weiter entfernt angebracht werden, z.B. wegen ansonsten mangelhafter Funkreichweite, so bietet sich eine Lösung über ein eigenständiges, Ethernet-fähiges Gateway an. Die Anbindung erfolgt entweder über kabelgebundenes Ethernet, wobei das Gerät mittels Power-over-Ethernet (PoE) auch mit Strom versorgt werden kann. Alternativ, bei eigener Stromversorgung, ist auch eine Anbindung über einen passenden WLAN-Standard möglich.

Der zweite wesentliche Unterschied besteht in der Adressierung des Gateways. Im Gegensatz zur Beschreibung in Kapitel 4.1.1 befindet es sich jetzt als reguläres Gerät neben allen anderen Teilnehmern im lokalen Netz (LAN) und erhält vom DHCP-Server eine IP-Adresse dieses Netzes zugeteilt.

Damit ergibt sich eine höhere Anforderung an die Rechenleistung und Speicherkapazität des Gateways, da es nun alle im Subnetz vorhandenen Broadcasts auf Ethernet-Ebene empfängt und diese gegebenenfalls verarbeiten muß. Eine Inspektion der Ethernet-Nutzdaten ist dafür in jedem Falle notwendig und eine entsprechend hohe Rechenleistung erforderlich.

Für das Design der Hardware empfiehlt sich eine Vorverarbeitung des Netzwerkverkehrs durch dedizierte Ethernet-Controller. Ein häufig verwendeter Controller ist der ENC28J80 der Firma Microchip [55], der über eigene Sende- und Empfangspuffer verfügt und mit rudimentären Ethernet-Filtern durch die MCU parametrisiert werden kann.

Die Kommunikation mit den Smart Devices verläuft ansonsten per Broadcast über das Anwendungsprotokoll, wobei das Gateway das Management des proprietären Nachrichtentransports übernimmt und die Nutzdaten zwischen Ethernet und Bus umsetzt.

Die beschriebene Architektur stellt den einfachen Fall der Implementierung mit abgesetztem Gateway dar. Smart Devices müssen lediglich den proprietären Transport implementieren und können auf einen Ethernet- und IP-Stack verzichten. Da eine Adressierung von

Smart Devices sowieso auf Anwendungsebene erfolgt, entstehen keine Nachteile.

4.1.5 Ethernetgerät als Ethernet-Brücke

Muß das Gateway aus Kostengründen einfach gehalten werden, kann auch eine Variante mit Ethernet-Bridging implementiert werden. Smart Devices werden in diesem Fall direkt an das innerhäusliche LAN angebunden, und lesen den gesamten Netzwerkverkehr mit.

Im Gegensatz zur Implementierung in Kapitel 4.1.2 werden Broadcasts nicht gefiltert, sondern über den proprietären Nachrichtentransport an alle Smart Devices dieses Segments weitergeleitet. Da durch dedizierte Controller keine Filterung in Hardware implementiert werden kann (Eine Filterung auf Transportebene wäre erforderlich), müssen Smart Devices diese selbst und möglicherweise in ihrer Firmware vornehmen. Insbesondere Broadcasts durch PCs, z.B. SOAP/WS* Webservice-Suchen, würden für die Verarbeitung durch Smart Devices möglicherweise wenig Sinn ergeben.

Dennoch kann eine Implementierung eines IP-Stacks auf jedem einzelnen Smart Device durchaus Sinn ergeben, wenn neben dem Anwendungsprotokoll noch weitere, IP-basierte Dienste verwendet werden. Für Küchengeräte kommt beispielsweise das NTP zur Anzeige der aktuellen Zeit auf Uhrendisplays in Betracht. Besitzt ein elektronischer Fotorahmen eine entsprechende Ausstattung, wäre auch ein Zugang per FTP zum Management der gespeicherten Bilder denkbar.

4.1.6 Smart Device mit Ethernet-Verbindung

Eng an den vorigen Abschnitt anknüpfend können Smart Devices auch direkt mit dem Ethernet-Segment verbunden sein. Denkbar ist ein solches Szenario bei Unterhaltungselektronik (TV, Verstärker, Spielkonsolen), die inzwischen gängige Internetdienste wie YouTube oder Rich Site Summary (RSS)-Feeds in eigenen lokalen Anwendungen darstellen können.

Eine Implementierung des Anwendungsprotokolls und Exponierens von einigen wichtigen Funktionen erscheint damit denkbar. In einem TV-Gerät könnten beispielsweise Datenpunkte zur Senderwahl, zur aktuellen Lautstärke und zu Bildeigenschaften (Helligkeit, Kontrast, etc.) eine Steuerung durch andere Smart Devices ermöglichen. Ein weiterer Anwendungsfall wäre ein „intelligenter“ Blu-ray-Abspieler, der mit Beginn des Films alle im Raum befindlichen Lampen ausschaltet oder langsam herunterdimmt und sie bei Ende des Filmes wieder einschaltet.

Die beschriebene Architektur bindet ein Smart Device direkt an das häusliche Ethernet an. Da das in der Regel für die meisten weißen Güter aus Kostengründen nicht stattfindet, ist der Vorschlag für Geräte der Unterhaltungsindustrie empfohlen. Zu beachten sind die notwendigen Ethernet-Fähigkeiten und ausreichend Filterkapazitäten. Über absolute Größen kann an dieser Stelle keine Angabe gemacht werden; die Abhängigkeit von der jeweils verwendeten Hardware ist gegeben.

4.2 Smart Home Internet-Gateway

Mit vorangegangen Ausführungen widmet sich dieses Unterkapitel der Beschreibung der Hard- und Software des zur Demonstration verwendeten Internet-Gateways bzw. Internet-Routers. Ein solcher wird als zentraler Bestandteil eines Smart Home angesehen und dient als Basis für eine Reihe von Aufgaben. Nach Erläuterung Letzterer werden zwei ähnliche Hardwareplattformen, ihr Betriebssystem sowie ihre Ausstattung mit Zusatzsoftware beschrieben. Damit wird die Grundlage für die Steuerung des Smart Home gelegt.

Die vorgestellte Lösung setzt soweit wie möglich auf offene und kostenfrei verfügbare Software sowie eigene Implementierungen.

4.2.1 Aufgaben

Der Internet-Router eines Smart Home übernimmt innerhalb des Netzes bestimmte Aufgaben und bietet den Geräten und Anwendern eigene Dienste. Neben der reinen Internet- und Netzwerkkonnektivität integrieren manche Hersteller in höherwertigen Modellen auch weitere Peripherie, z.B. für Funktelefonie nach dem Digital Enhanced Cordless Telecommunications (DECT)-Standard. Dies spielt für die Smart Functionality allerdings keine Rolle und wird im Folgenden nicht weiter beachtet oder vorausgesetzt.

Zur Wahrnehmung der im Folgenden gelisteten Aufgaben besitzen Internet-Router eine webbrowserbasierte Konfigurationsschnittstelle für den Anwender, der so die Dienste des Routers parametrisiert:

- *Internetverbindung (Wide Area Network (WAN)).* Die eingebettete Plattform eines Internet-Routers bietet mindestens zwei Ethernet-Schnittstellen. Eine davon stellt mit Hilfe weiterer Hardware über den kontraktierten Provider einen Zugang zum Internet her („Einwahl“). Vorrangig in Deutschland kommt dabei die Digital Subscriber Line (DSL)-Technik über Point-to-Point-Protocol-over-Ethernet (PPPoE) mit unterschiedlichen Datenraten zum Einsatz. Netzwerktechniken wie Network Address Translation (NAT) und Port Address Translation (PAT) stellen bei IP-Konnektivität eine Trennung der Internetrechner von den Geräten im Smart Home sicher. Der Router benötigt zur Einwahl die von Provider bereitgestellten Zugangsdaten. Er übernimmt das Management der Internetverbindung (erneute Einwahl, Trennen bei Inaktivität).
- *Firewall.* Da Rechner im Internet unter Umständen Dienste auf lokalen Hosts nutzen sollen, muß ein Verbindungsaufbau von außerhalb möglich sein. Das Firewall-Modul im Router überwacht die eingehenden Verbindungen und prüft deren Attribute gegen bestehende Regeln (z.B. „offene Ports“) und Einhaltung der Protokollformate („Stateful Inspection“). Entspricht eine Anfrage diesen Regeln, werden die jeweiligen IP-Pakete unter Umschreibung der Absenderadresse („Masquerading“) an den internen Rechner durchgeleitet. Eine Prüfung zwischen Schnittstellen im internen Netz ist in der Regel nicht vorgesehen, kann unter Umständen aber eingerichtet

werden, beispielsweise für ein (offenes) Gastnetz für WLAN-Geräte.

- *Kabelgebundenes Ethernet (LAN).* Meistens bieten Geräte bereits einen integrierten Hub für kabelgebundene Ethernetgeräte, der 4 bis 8 Anschlüsse aufweist und (Stand August 2013) Geräte bereits mit Übertragungsgeschwindigkeiten von bis zu $1\,\mathrm{Gbit\,s^{-1}}$ anbindet. Dabei existiert eine Rückwärtskompatibilität zu bisherigen Datenraten von $10\,\mathrm{Mbit\,s^{-1}}$ und $100\,\mathrm{Mbit\,s^{-1}}$. Der Hub weist in der Regel *Switching*-Funktionalität auf und kann Ethernet-Frames zielgerichtet an seinen jeweiligen Ports ausgeben.

- *Drahtloses Netzwerk (WLAN).* In gängigen Routern ist zudem eine WLAN-Schnittstelle integriert. Darüber können drahtlose Clients ebenfalls eine Verbindung zum Internet herstellen. Die Schnittstelle wird dazu im Access-Point-Modus betrieben. WLAN-Geräte können in einigen Modellen voneinander isoliert werden. Außerdem bieten manche Hardwaremodule die Möglichkeit, mehrere drahtlose Netzwerke mit unterschiedlichen Service Set Identifier (SSID) und verschiedenen Sicherheitseinstellungen zu erstellen.

- *Smart Devices – Netzwerkverbindung* Diese Aufgabe wurde in Kapitel 4.1 bereits beschrieben. Der Router bedient lokal über USB angeschlossene Gateways zur Kommunikation mit Smart Devices. Dies erfolgt über die Communication-Device-Class (CDC)-Geräteklasse der USB-Spezifikation und ist entweder über eine virtuelle serielle Schnittstelle oder ein RNDIS-Netzwerkgerät implementiert.

- *Smart Functionality – Anwendungsebene.* Innerhalb des Smart Home wird auf dem Router auch ein Teil der Anwendungslogik implementiert. Immer dann, wenn eine einfache Verknüpfung von Datenpunkten für eine gewünschte Funktionalität des Systems nicht ausreicht, wird der erforderliche Programmcode in einem dafür vorgesehenen Teil des Routers ausgeführt. Kapitel 4.6 beschäftigt sich mit einer beispielhaften Implementierung, die auf ressourcenbeschränkten eingebetteten Routerplattformen eingesetzt werden kann.

4.2.2 Hardware

Zwei unterschiedliche Plattformen des Herstellers TP-Link wurden im Rahmen der Arbeit verwendet. Sie sind mit vergleichbarer Funktionalität betreffend die Netzwerkeigenschaften ausgestattet; wesentliche Unterschiede liegen in der Ausstattung der Rechnerplattform und des typischen Straßenpreises. In Tabelle 4.1 werden beide Plattformen gegenübergestellt.

Im Rahmen der Evaluation wird in Kapitel 5 ein Vergleich der Performanz und Reaktionszeit der beiden Plattformen für die programmierte Anwendungssoftware angestellt. Insbesondere wird untersucht, ob und gegebenenfalls wie sich die unterschiedlichen Befehlssätze und Taktfrequenzen der CPUs auf die Reaktionszeit der programmierten Software auswirken.

		
	TL-WR1043ND [56, 57]	**TL-WDR4900** [58, 59]
Plattform: Befehlssatz	MIPS32 (Atheros)	PowerPC
Plattform: Hersteller	Qualcomm	Freescale Semiconductor
Plattform: SoC & CPU	AR9132, 24Kc V7.4	Freescale MPC8548, PPC P1014
Taktfrequenz:	400 MHz	800 MHz
RAM	32 MB	128 MB
Flash-ROM	8192 kB	16 384 kB
WLAN-Modul 2,4 GHz	Atheros AR9103, 3x3 MIMO	Atheros AR9381-AL1A, 3x3 MIMO
WLAN-Modul 5 GHz	*n/a*	Atheros AR9580-AR1A, 3x3 MIMO
USB-Ports	1x USB 2.0	2x USB 2.0
Straßenpreis (8/2013)	~30 €	~85 €

Tabelle 4.1: Gegenüberstellung der verwendeten Routerhardware. Beide Geräte besitzen für typische Heimanwendung gängige Peripherie wie Ethernet-Switches, WLAN-Funktionen und Interkonnektivität über ein WAN-Interface. Neben empfohlenem Verkaufpreis unterscheiden sie sich hauptsächlich in der Ausstattung der Rechnerplattform.

4.2.3 Grundlegende Software

Die originale Firmware des Herstellers wurde gegen eine Linux-basierte Open-Source-Variante ausgetauscht. Es ergibt sich hierdurch kein Nachteil, da das Betriebssystem speziell für die Hardware des Routers kompiliert wird und die selben grundlegenden Funktionen besitzt. Teilweise stellt es sogar erweiterte Funktionen zur Verfügung, etwa zur Einrichtung mehrerer WLAN-Netze oder durch direkten Zugriff auf die Firewall `iptables`. Eine übersichtliche Weboberfläche steht für Endbenutzer zur Verfügung.

Außerdem existiert eine Reihe von Kernelmodulen zur Unterstützung externer, USB-gebundener Hardware, die mit der originalen Firmware nicht verwendet werden könnten.

Ausschlaggebend für den Austausch ist allerdings die Möglichkeit, eigene Software für das Betriebssystem zu entwickeln. Das Management der virtuellen seriellen Schnittstelle sowie ein beispielhafter Regelinterpreter für Anwendungslogik machen hiervon Gebrauch.

4.2.3.1 Betriebssystem

Auf beiden Plattformen wird das für eingebettete Systeme optimierte OpenWRT ([60, I.3] und [61]) eingesetzt.

Für die ältere Plattform WR1043ND wird aufgrund der Hardwareversion 1.0 die stabile und nicht mehr weiterentwickelte OpenWRT-Variante „Backfire“ in Version 10.03.1 mit Kernel 2.6.32.27 verwendet. Sie wurde von der OpenWRT-Webseite geladen [62]. Bei Neukauf des Routers ist unbedingt die Hardware-Revision zu ermitteln; „Backfire“ kann nur auf Version 1.0 betrieben werden [57].

Erst mit der aktuellen Entwicklerversion 12.09 „Attitude Adjustment“ wurde eine Unterstützung für die MPC85xx Power-PC-Plattform hinzugefügt. Betriebssystemkern und nachinstallierbare Module sind nicht als „stable“ markiert. Bei Test und Langzeitbetrieb ist allerdings keine Fehlfunktion aufgetreten. Die längste Betriebsdauer betrug 118 Tage ohne Neustart. Die verwendete Entwicklerversion für den WDR4900 lautet r36602 mit Kernel 3.8.12; sie wurde von der OpenWRT-Webseite geladen [63].

Aus mehreren Derivaten (FreeWRT, DD-WRT) wurde OpenWRT der folgenden Gründe wegen ausgewählt:

- *Plattformunterstützung.* OpenWRT ist von den drei Varianten die einzige, die beide vorgesehenen Plattformen unterstützt.
- *Auswahl verfügbarer Erweiterungen.* Die Anzahl der verfügbaren Module ist bei OpenWRT mit weit über 2000 am größten. Die für diese Arbeit benötigten Schnittstellentreiber liegen bereits als installierbare binäre Komponenten vor, was bei Installation und Konfiguration Fehlerquellen verringert und Zeit spart.
- *Paketmanagement.* Alle Erweiterungen können über ein eigens implementiertes Paketmanagementsystem `opkg` [64] verwaltet werden.
- *Unterstützung durch die Community.* Am OpenWRT-Projekt beteiligen sich (der Einschätzung des Autors nach) äußerst qualifizierte Programmierer. Über Mailinglisten und Foren leisten sie zeitnah und umfangreich Hilfestellung bei Schwierigkeiten. Letztlich hat dies die Entscheidung für OpenWRT maßgeblich beeinflußt.
- *Geringe Größe des Betriebssystems.* OpenWRT belegt ab Installation nur eine Größe von ~3,5 MB auf dem internen Speicher der Geräte.

4.2.3.2 Installierte Hilfsmodule und -Software

Auf beiden Internet-Routern wurde vorbereitend eine Reihe von Zusatzmodulen installiert, die über das Paketmanagement `opkg` verfügbar ist. Die folgende Liste enthält den Namen der Module und ihre Beschreibung.

- `udp-broadcast-relay`. Das bereits erwähnte Softwaremodul ermöglicht ein Weiterleiten von UDP-Broadcasts auf einem bestimmten Port an definierte Schnittstellen. Die Absender- und Zieladressen werden dabei nicht verändert.
- `kmod-usb-core`. Bietet generische Kernelunterstützung für die USB-Hardware.
- `kmod-usb-net`. Bietet Kernelunterstützung für USB-Netzwerkgeräte (NICs, Modems, etc.).

- `kmod-usb-ohci`. Lowlevel-Treiber für das Open Host Controller Interface (OHCI). Je nach verwendeter USB-Hardware muß statt diesem Treiber `kmod-usb-uhci` installiert werden, der Geräte mit Universal Host Controller Interface (UHCI) steuert.
- `libc` und `libgcc`. Bieten plattformabhängige Unterstützung für Standardprogramme in *C* und müssen für selbsterstellte Software in der Regel vorhanden sein. Statisches Linken erzeugt größeren Code, was auf der Zielhardware unbedingt zu vermeiden ist.

4.2.3.3 Kerneltreiber für virtuelle serielle Schnittstellen

OpenWRT bietet Kerneltreiber für gängige Integrated Circuit (IC) für virtuelle serielle Schnittstellen. Die Grundvoraussetzung ist die Installation des Moduls `kmod-usb-serial`. Darauf aufbauend sind die passenden Treiber für die seriellen Wandler zu installieren. Im Rahmen der Arbeit wurden zwei verschiedene serielle ICs verwendet, der FT232R (FTDI) und der PL2303 (Prolific). Die entsprechenden Kerneltreiber lauten `kmod-usb-serial-ftdi` und `kmod-usb-serial-pl2303`.

4.2.3.4 Kerneltreiber für CDC-Ethernet und RNDIS-Protokoll

Für USB-Ethernetkarten und den RNDIS-Unterstützung sind ebenfalls zwei Kerneltreiber zu installieren. Zum einen ist das das Modul `kmod-usb-net-cdc-ether`. Es bietet Kernelunterstützung für USB-CDC-Geräte. Zum anderen muß das Modul `kmod-usb-net-rndis` installiert werden. Dieses bietet Kernelunterstützung für Microsofts RNDIS-Protokoll. Gebräuchlich ist es bei Mobilfunkmodems, deren Netzwerkschnittstelle über RNDIS angesprochen werden kann.

4.2.4 Daemon für serielle Schnittstellen

Um Ethernet mit den seriellen Schnittstellen zu verbinden, wird eine eigene, in *C* programmierte Software eingesetzt. Ihre Aufgabe besteht in der Terminierung der IP-Nachrichten und der Umsetzung deren Nutzdaten auf die seriellen Schnittstellen, sowie einer rudimentären Filterung etwa von nicht unterstützten HTTP-Headern.

Das Programm kann als Konsolenanwendung im Vordergrund laufen, wobei Debugmeldungen auf der Konsole in Echtzeit ausgegeben werden. Alternativ besteht die Möglichkeit, das Programm über `fork()` in einen Hintergrundprozeß auszugliedern. Informationen und Fehlermeldungen werden dann an das `syslog` gesendet.

4.2.4.1 Regeln für über Ethernet eingehende Pakete

Da jede serielle Schnittstelle nur eine einzige logische Verbindung unterstützt und durch das Anwendungsprotokoll im Halbduplex-Verfahren arbeitet (auch wenn die Schnittstelle an sich Vollduplex unterstützt), muß innerhalb des Programms ein Management der potentiell mehrfachen eingehenden Verbindungen von Clients und anderen Smart Devices sowie der Antworten der angeschlossenen seriellen Schnittstellen durchgeführt werden.

Auf der Ethernetseite können Steuerungen das Programm über TCP oder UDP ansprechen; mehrfache Verbindungen und UDP-Anfragen gleichzeitig sind möglich. Die folgenden Regeln wurden implementiert:

- *Über Ethernet eingehende TCP-Verbindungen.* Das Transportprotokoll behandelt die Verbindungen selbständig. Bei Erstellung des Sockets kann eine Warteschlangenlänge („Backlog“) im Systemcall `listen()` angegeben werden. Damit können mehrere Clients unicast auf alle Smart Devices zugreifen; eine Abarbeitung findet sequentiell in der Hauptschleife statt. So ist sichergestellt, daß eine Anfrage eines Clients stets vollständig vom Bus beantwortet wurde, bevor eine neue TCP-Verbindung angenommen wird. Dauert eine Abarbeitung entsprechend lange, so brechen die Clients in der Warteschlange nach einem Timeout ihre Verbindungsanfrage ab. Solange eine TCP-Verbindung existiert, wird die Annahme von UDP-Paketen unterbrochen; sie verbleiben im Puffer des Netzwerkstacks und können – bei Überschreiten dessen Kapazität – verworfen werden. Der Puffer kann für einen Socket über die Socketoption `SO_RCVBUF` angepaßt werden.

- *Über Ethernet eingehende UDP-Nachrichten.* Da hier keine Verbindung existiert, wird für die Abarbeitung der Anfrage die Absenderinformation (IP-Adresse und Quellport) gespeichert. Die Antworten der Smart Devices werden dann über ein oder mehrere UDP-Pakete an diese Adressdaten zurückgesendet. Solange eine UDP-Nachricht bearbeitet wird, können keine TCP-Verbindungen angenommen werden. Sie verbleiben im Backlog des Puffers und werden bei Überschreiten abgewiesen („reject“).

- *Event-Nachrichten.* Im Falle von Event-Nachrichten werden diese – unabhängig vom Transportprotokoll – auf allen seriellen Schnittstellen ausgegeben. Bei TCP wird die Verbindung dann beendet. Dazu ist eine Inhaltsanalyse des Netzwerkverkehrs implementiert, die einfach die ersten fünf Byte mit `EVENT` vergleicht.

Das Programm behandelt in seiner Hauptschleife stets nur *eine einzige* IP-Nachricht. So kann die Struktur des Programms einfach gehalten werden. Threads, Semaphore oder explizite Zustandsautomaten werden nicht benötigt.

Ein Routing zwischen Ethernet und mehreren seriellen Schnittstellen basierend auf dem Anwendungsprotokollinhalt findet nicht statt; die eintreffenden Nutzdaten werden auf allen seriellen Schnittstellen ausgegeben.

4.2.4.2 Regeln für Antworten der seriellen Schnittstellen

Die Antworten auf Anfragen von mehreren Smart Devices werden zusammenhängend versendet. Da jede serielle Schnittstelle allerdings ihr separates Smart Device Segment mit eigenen HTTP-Headern und umschließenden `<bus>`-Elementen aufspannt (siehe Listing 4.2), wird eine Kombination der Antworten mit Inhaltsanalyse durchgeführt.

```
>>> GET / HTTP/1.1
>>> Host: *

[Rohantwort der ersten seriellen Schnittstelle]
<<< HTTP/1.1 200 OK                                    -- wird ignoriert
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>        -- wird ignoriert
<<< <bus>                                              -- wird ignoriert
<<<   <fridge_00017FBA5>...</fridge_00017FBA5>         -- wird übernommen
<<<   <light_F31D>...</light_F31D>                     -- wird übernommen
<<< </bus>                                             -- wird ignoriert

[Rohantwort der zweiten seriellen Schnittstelle]
<<< HTTP/1.1 200 OK                                    -- wird ignoriert
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>        -- wird ignoriert
<<< <bus>                                              -- wird ignoriert
<<<   <waschmaschine_9123>...</waschmaschine_9123>     -- wird übernommen
<<<   <dishwasher_AD32>...</dishwasher_AD32>           -- wird übernommen
<<< </bus>                                             -- wird ignoriert
<<< EVENT /dishwasher_AD32/time/remaining              -- wird beibehalten
<<< Content-Length: 4                                  -- wird beibehalten
<<<
<<< 1801                                               -- wird beibehalten
```

Listing 4.2: Ursprüngliche Antwort der Smart Devices, in jeweilige Zwischenpuffer kopiert. Man beachte, daß im zweiten Puffer auch eine EVENT-Nachricht enthalten ist.

Zunächst wird in einem eigenen Thread jeder Puffer der Schnittstellen zyklisch alle 10 ms mittels `select()` auf neue Daten geprüft. Liegen im Empfangspuffer Daten vor, wird ein Timer, dessen Ablauf auf 200 ms festgelegt wurde, zurückgesetzt, und alle Daten (Rohantworten) in Zwischenpuffer – getrennt nach Schnittstelle – der Anwendung kopiert.

Sobald der Timer abgelaufen ist und damit keine Daten mehr in den *Empfangspuffern* der Schnittstellen vorliegen, kann von einer vollständigen Antwort aller Devices aller Schnittstellen ausgegangen werden. Bei den nun vorliegenden Antworten werden mittels Inhaltsanalyse initiale HTTP- und XML-Header verworfen und nur vollständige Antworten der einzelnen Devices (ohne `<bus>`-Element) in einen weiteren Sendepuffer der Anwendung verschoben, siehe Listing 4.3, unterer Teil. Die so erzeugte Antwort wird um passende HTTP- und XML-Header ergänzt. Dabei kommt auch der `Content-Length`-Header zur Anwendung.

```
<<< HTTP/1.1 200 OK
<<< Content-Length: 946
```

```
<<<
<<< <?xml version="1.0" encoding="iso-8859-1"?>
<<< <bus>
<<<   <fridge_00017FBA5>...</fridge_00017FBA5>
<<<   <light_F31D>...</light_F31D>
<<<   <waschmaschine_9123>-...</waschmaschine_9123>
<<<   <spuelmaschine_AD32>...</spuelmaschine_AD32>
<<< </bus>
```

Listing 4.3: Kombination der Antworten mehrerer Smart Devices aus den Zwischenpuffern der seriellen Schnittstellen für einen Ethernet-Client in einem einzigen Sendepuffer. Aus den ursprünglichen Antworten werden nur vollständige Device-Knoten extrahiert und an den Ethernet-Client zurückgesendet. Dabei kann auch der `Content-Length`-Header eingesetzt werden, da nun die Gesamtgröße der Anfrage bekannt ist.

Letztlich verbleiben in den jeweiligen Zwischenpuffern der seriellen Schnittstellen nur noch unbearbeitete Daten, wie in Listing 4.4 dargestellt.

```
[Inhalt des Zwischenpuffers der ersten seriellen Schnittstelle]
// leer

[Inhalt des Zwischenpuffers der zweiten seriellen Schnittstelle]
EVENT /spuelmaschine_AD32/zeit/verbleibend
Content-Length: 4

1801
```

Listing 4.4: Inhalt des Zwischenpuffers nach Entfernen der Antworten für die Weiterleitung über UDP oder TCP. Diese Daten werden im nächsten Durchlauf der Hauptschleife verarbeitet; hier etwa als `EVENT`-Nachricht.

Abschließend wird der Sendepuffer an den jeweils gültigen Client (TCP-Verbindung oder UDP-Quelldaten) zurückgesendet. So ist sichergestellt, daß stets nur eine einzige Anfrage an Smart Devices gleichzeitig bearbeitet wird.

4.2.4.3 Von Smart Devices ausgehende Event-Nachrichten

Zu jeder Zeit, auch direkt nach Übertragung aller Antworten auf Anfragen, können Smart Devices Event-Nachrichten auf den Bus absetzen, wie in Listing 4.2 für den ersten Bus bereits dargestellt. Mit Hilfe einer Inhaltsanalyse der Zwischenpuffer werden diese Nachrichten extrahiert und per UDP-Broadcast versendet.

Es wird davon ausgegangen, daß Smart Devices von sich aus keine anderen Nachrichten als Events versenden; deswegen können aus den Puffern alle Daten verworfen werden, die nicht dem Event-Schema entsprechen. Dazu wird der Puffer zeichenweise geleert, bis das Verb `EVENT` auftritt. Mit Hilfe des folgenden `Content-Length`-Headers wird die entsprechende Anzahl von Nutzdatenbytes aus dem Puffer entfernt und das gesamte Konstrukt versendet.

Dieser Mechanismus wird, genau wie bei TCP und UDP, in einer Schleife ausgeführt, bis keine Daten mehr in einem Zwischenpuffer vorhanden sind. Nicht-Event-Nachrichten werden in diesem Teil des Programms (keine gültige TCP- oder UDP-Verbindung) verworfen, da keine Ethernet-Zieladresse vorliegt.

4.2.5 Medienwandlung der virtuellen seriellen Schnittstellen

Wie beschrieben findet die Umsetzung der Daten aus dem Router heraus über Adapter der Firmen FTDI und Prolific statt. Sie stellen eine dem Electronic Industries Alliance (EIA) Standard 232 entsprechende serielle Schnittstelle zur Verfügung, an welche passende serielle Endgeräte verbunden werden.

Die verwendeten Produkte sind in Abbildung 4.2a und 4.2b dargestellt. Im August 2013 lag der Straßenpreis bei ungefähr 10 €.

Im Betriebssystem werden beide Geräte im Ordner `/dev/` als Dateien eingeblendet. Wie für serielle Schnittstellen üblich tragen sie den Namen `tty*`, wobei zur besseren Abgrenzung der Text `USB` zwischen die Schnittstellennummer eingefügt wird. Der vollständige Pfadname der ersten Schnittstelle, unabhängig vom Hersteller, lautet `/dev/ttyUSB0`.

Sie kann wie eine reguläre Datei gelesen und beschrieben werden; innerhalb einer Anwendung müssen aber Fehlerbehandlungsroutinen für I/O-Fehler vorhanden sein, ansonsten wird das Programm mit einer Fehlermeldung vom Betriebssystem beendet. Insbesondere das Entfernen des Adapters zur Laufzeit muß abgefangen werden.

Die beiden Treiber für OpenWRT weisen hinsichtlich ihrer Funktionalität keinen Unterschied auf. Lediglich die Konfigurationsmasken unter Windows bieten unterschiedliche Optionen, die an dieser Stelle nicht behandelt werden.

Bei Lasttests mit der nachfolgend beschriebenen Hardware waren keine Unterschiede zwischen den beiden Adaptern festzustellen; die Datenrate von 115,2 kbit s^{-1} konnte auch gehalten werden, wenn die Adapter nur über einen zwischengeschalteten USB-Verteiler verbunden waren.

(a) FT232RL-Chipsatz.

(b) PL2302-Chipsatz.

Abbildung 4.2: Abbildungen der verwendeten Adapter für virtuelle serielle Schnittstellen. Neben dem IC zur Kommunikation mit dem USB-Host befindet sich ein Pegelwandler zur Anpassung an die Schnittstelle EIA-232 im Gehäuse.

4.3 Konstruierte Smart Devices

Im Rahmen der Arbeit wurden zur Demonstration des Konzeptes einige Smart Devices entworfen und konstruiert. Ziel davon ist neben der Validierung des beschriebenen Softwarekonzepts auf möglichst kleinen MCU-Plattformen insbesondere die Demonstration der Praxistauglichkeit sowie eine Umsetzung der eingangs beschriebenen Anforderungen von potentiellen Geräteherstellern.

Dazu wird auf eine Verwendung von käuflichen Modulen (Sensorknoten, eingebettete Plattformen, Breakout-Boards etc.) bewußt verzichtet. Ein leichter und auch für Hobbyelektroniker nachvollzieh- und durchführbarer Herstellungsprozeß stellt eine wesentliche Anforderung an die Hardware dar.

Weiterhin sollen möglichst preiswerte und qualitativ „schlechte“ Bauteile eingesetzt werden, um ein Bad-Case-Szenario zu realisieren. Wenn gezeigt werden kann, daß das Konzept auf Plattformen mit bewußt minderwertig gewählten Bauteilen funktioniert und nach noch zu definierenden Kriterien akzeptabel läuft, kann daraus eine Funktion auf leistungsfähigerer Hardware inferiert werden.

Die im Einzelnen wurden entworfenen Module und Geräte werden im Folgenden beschrieben.

4.3.1 Vorbemerkungen

Zur Demonstration wurde eine Variante eines LED-Controllers für die drahtgebundene RS-232-Schnittstelle entworfen, während alle anderen Devices drahtlos über ein proprietäres Funkmodul im 2,4 GHz-Band kommunizieren. Die folgenden Ausführungen geben allgemeine Hinweise und Beschreibungen zu Konstruktion, Herstellung und verwendeten Modulen der Smart Devices.

4.3.1.1 Leiterplattenentwurf

Alle Leiterplatten (Printed Circuit Board, PCB) wurden mit Hilfe des Programms „Eagle“ der Firma Cadsoft [65] in Version 6.1 vom Autor selbst erstellt. Dabei wurde auf ein einseitiges Layout und möglichst große Gehäuseformen von Bauteilen (Widerständen, Kondensatoren, ICs) Wert gelegt, um Nachbauten für Hobbyelektroniker mit preiswerten Werkzeugen so einfach wie möglich zu halten. Wo möglich, werden Bauteile in Surface Mounted Technology (SMT) verwendet.

Die Fertigung der Leiterplatten wurde bei der Firma PCB-Pool [66] beauftragt. Durch breite Leiterbahnen kann ein geübter Hobbyelektroniker die Leiterplatten aber auch selbst mit Hilfe der Toner-Direktdruck-Methode [67] oder – einen Laserdrucker mit mindestens 1200 dpi vorausgesetzt – mit der Belichtungsmethode [68] herstellen; der Autor konnte das bei der RS-232-Variante des LED-Controllers selbst erfolgreich durchführen. An manchen Stellen konnte eine Drahtbrücke auf der Bestückungsseite nicht vermieden werden; das

ist bei geringen Stückzahlen aber stets preiswerter als eine zweiseitige Platine.

4.3.1.2 Micro Controller Unit (MCU)-Plattformen

Die verwendete MCU-Plattform ist die *XMega*-Reihe des Herstellers Atmel. Neben 8-bit Rechenwerk, statischem RAM und Programmspeicher (Flash-ROM) enthält sie auch Peripheriemodule wie SPI- und I²C-Buscontroller [69, Chapter 13] sowie flexibel programmierbare Timer/Counter-Einheiten. Für diese Arbeit relevante Daten sind in Tabelle 4.2 basierend auf den Datenblättern [70, 71] angegeben.

Weiterhin kommt die ältere MCU ATmega32U2 der MEGA-Reihe zum Einsatz, anhand derer die USB-Funktion auch mit geringeren Resourcen als auf der XMega-Reihe verifiziert wurde.

4.3.1.3 Toolchain zur Programmierung

Alle MCUs der beschriebenen Baureihen verfügen über die Möglichkeit zur In-System Programmierung (ISP). Ein Modul innerhalb der MCU wird mit einem externen Programmieradapter entweder über eine besondere SPI-Schnittstelle (megaAVR) oder über dedizierte Leitungen (xmegaAVR) verbunden. Neben der Betriebsspannung und den Datenleitungen ist insbesondere eine gemeinsame Masse (Bezugspotential) notwendig. Über die Schnittstelle kann eine bereits verlötete MCU nachträglich im Zielsystem mit neuer Firmware ausgestattet werden, wenn physikalischer Zugriff besteht. Im Rahmen der Arbeit werden so alle Smart Devices programmiert.

Abbildung 4.3: STK-600 Entwicklungsboard.

	mega32U2	xmega16A4U	xmega32A4U	xmega128A4U
ROM / RAM	32 kB / 1 kB	16 kB / 3,3 kB	32 kB / 4 kB	128 kB / 8 kB
Taktfrequenz	8 MHz	32 MHz	32 MHz	32 MHz
Gehäuseform	TQFP 32	TQFP 44	TQFP 44	TQFP 44
Anzahl 16 Bit PWM-Kanäle	2	16	16	16
SPI	2	7	7	7
TWI	0	2	2	2
USART	1	5	5	5
USB	1 (Full Speed)	1 (Full Speed)	1 (Full Speed)	1 (Full Speed)

Tabelle 4.2: Darstellung der verwendeten MCUs sowie ihrer Peripheriebausteine. Die ICs der XMega-Reihe unterscheiden sich nur in ihrer Flash-ROM- und RAM-Ausstattung. Für den Betrieb des USB-Peripheriemoduls ist eine Core-Taktfrequenz mit einem Vielfachen von 8 MHz notwendig, da durch Vervielfachen der nötige 48 MHz-Takt für das Modul erzeugt wird.

Der erwähnte Programmieradapter ist das Entwicklungsboard STK-600 des selben Herstellers, siehe Abbildung 4.3. Es wird mit dem Entwicklungsrechner per USB verbunden und erhält darüber auch seine Stromversorgung. Die Verbindung mit dem eingebetteten Zielsystem erfolgt über einen sechspoligen Stecker. Je nach Ziel-MCU besitzt er eine unterschiedliche Belegung [72, Kapitel 3].

Die Mikrocontroller können in Assembler oder *C* programmiert werden. Im Sinne einer geringeren Entwicklungszeit wird in dieser Arbeit ausschließlich in *C* programmiert. Dafür wird das Atmel-Studio in Version 6 eingesetzt, das neben einer Editieroberfläche auch einen aktuellen AVR-GCC Compiler, einen Assembler/Linker und eine Schnittstelle zum Programmieradapter bietet.

4.3.1.4 RFM70-Funkmodul (2,4 GHz-Band)

Anstelle von standardisierten Funktechniken wie 802.15.4 wird in dieser Arbeit bewußt ein preiswertes und proprietäres Funkmodul eines chinesischen Herstellers verwendet. Das RFM70 [73] des chinesischen Herstellers HopeRF [74] ist speziell für eingebettete Anwendungen entworfen und bietet folgende Eigenschaften:

- Verwendung des kostenfreien Frequenzbereichs von 2400 MHz bis 2483,5 MHz
- Bruttodatenrate von 1 Mbit s^{-1} oder 2 Mbit s^{-1} (im Burst-Betrieb)
- Programmierbare Sendeleistung von −35 dBm bis +5 dBm
- Nutzdatenlänge von 1 - 32 Byte
- Zwei unabhängige Empfangsadressen, jeweils 3 - 5 Byte lang
- Stromversorgung von 1,9 V bis 3,6 V
- Automatische Erstellung und Prüfung einer CRC-Prüfsumme
- Automatische Neuübertragung von Paketen bei Unicast-Betrieb
- SPI-Taktfrequenz bis zu 8 MHz

Das Modul wird generell bei Programmstart einmalig mit Informationen zur Datenrate, Paketlänge und Sendeleistung parametrisiert. Dabei werden auch die beiden Empfangsadressen – eine für Broadcast-, die andere für Unicast-Übertragungen – eingerichtet. Im EEPROM jedes Smart Devices ist dazu eine eineindeutige MAC-Adresse einprogrammiert.

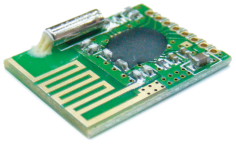


Abbildung 4.4: RFM70

An die MCU wird das RFM70 per SPI angeschlossen. Mit einer separaten Interruptleitung wird die MCU je nach Betrieb über ein versendetes oder erfolgreich empfangenes Paket informiert, das dann über SPI auszulesen ist.

Bis zu drei Pakete können im Modul gepuffert werden, falls die MCU gerade andere Aufgaben ausführt. Für schnellen Powerdown ist eine weitere Leitung („Chip enable“) eingerichtet. Liegt sie auf Low-Pegel, sind Sende- und Empfangsteile deaktiviert, womit sich der Stromverbrauch auf wenige Mikroampère reduziert. Das Paket-Format und die

konkreten Konfiguration werden später in Kapitel 4.4 beschrieben.

Inzwischen (05/2013) ist das Modul abgekündigt. Es ist durch einen pin- und funktionskompatiblen Nachfolger, das RFM73, ersetzt [75].

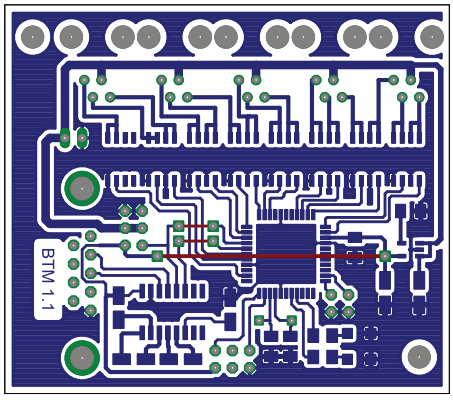
4.3.2 LED-Controller (RS-232)

Das erste entwickelte Smart Device basiert auf dem xmega16A4(U) und dient der Steuerung von bis zu 15 LEDs. Der Entwurf der Platine ist in Abbildung 4.5a, die aufgebaute Platine in Abbildung 4.5b dargestellt. Neben der MCU, mittig platziert, befinden sich ein Linearregler für 3,3 V, ein MAX3232-Pegelwandler und acht Leistungstreiber (IRF7341 N-Channel Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) [76]) auf der Platine. Die MCU wird mit einem externen Quarz auf 16 MHz Taktfrequenz betrieben.

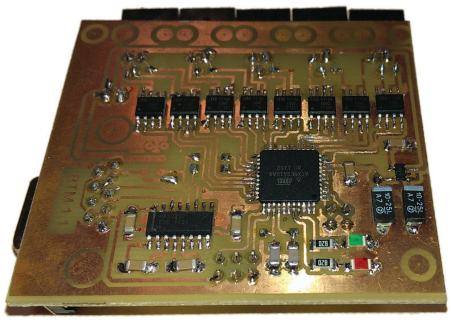
4.3.2.1 Funktionsbeschreibung der Hardware

Der Linearregler erzeugt dabei aus einer Eingangsspannung von bis zu 20 V die Versorgung für die MCU und den Pegelwandler. Eine Kühlung des Reglers findet über die Massefläche der Platine statt. Bei einem Strom von 40 mA und einer Spannung von 5 V ist keine Erwärmung spürbar.

Die LEDs werden in Gruppen zu jeweils drei Stück per RJ-11 Steckverbindung („Telefonkabel“) mit der Platine verbunden. Diese ist ihrerseits über einen bereits beschriebenen USB-zu-Seriell-Wandler an den Host (Internet-Router) angeschlossen. In Abbildung 4.6 ist die vollständige Verbindungskette dargestellt.



(a) Darstellung der vollständig gerouteten Platine aus EAGLE heraus. Die roten Linien verlaufen auf der Oberseite der Platine.



(b) Fertig bestücktes Smart Device. Auf der linken Seite sind die RS-232-Buchse, am oberen Rand die RJ-11 Verbinder erkennbar.

Abbildung 4.5: LED-Controller in Entwurfsstadium und gefertigtem Zustand.

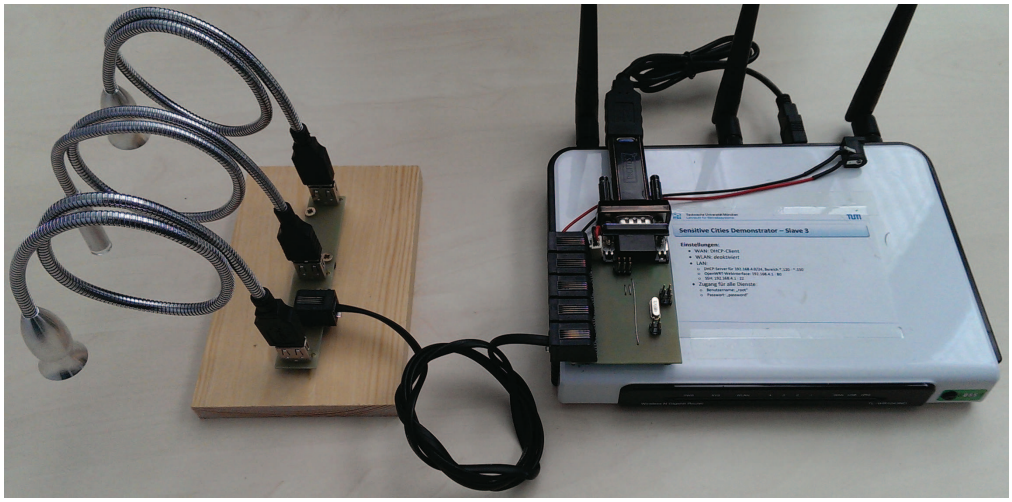


Abbildung 4.6: Abbildung des ersten Smart Devices, seiner Peripherie und des zugehörigen Internet-Routers.

Alle LEDs können separat in ihrer Helligkeit in einer Auflösung von 16 Bit gesteuert werden. Dazu sind die fünf in der MCU vorhandenen Timer/Counter-Module mit 15 unabhängigen Ausgängen auf PWM-Betrieb eingerichtet und steuern insgesamt 15 Leistungstransistoren an. Jede LED ist („gemeinsame Anode“) mit ihrer Masse mit einem Transistor verbunden und erhält die positive Versorgung direkt aus der Eingangsspannung der Schaltung. Dieses Schaltungsprinzip ist als „Low Side Switching“ bekannt und bietet den Vorteil, Bauteile mit unterschiedlichen Versorgungsspannungen an einem gemeinsamen Massepotential betreiben zu können.

Das Schaltprinzip der gemeinsamen Anode ermöglicht eine flexible und den LEDs angepaßte Stromversorgung. Prinzipiell kann jede Eingangsspannung bis 20 V entsprechend den Anforderungen der Einzel-LEDs oder in Reihe geschalteten LEDs („Streifen“) ohne Beeinflussung der restlichen Schaltung verwendet werden.

Zwei auf der Platine angebrachte SMT-LEDs dienen Diagnosezwecken und zeigen im normalen Betrieb Bereitschaft und Schnittstellenaktivität an. Letztere ist über den Pegelwandler MAX3232 mit einem USART der MCU verbunden.

4.3.2.2 Netzwerktopologie und Busarbitrierung bei mehreren Devices

In Abbildung 4.1 stellt diese Konfiguration die Variante mit „Herstellergateway C“ dar, wobei der LED-Controller als eines der Smart Devices 5, 6 oder 7 gelten kann.

Eine RS-232-Schnittstelle ist ursprünglich nur für Punkt-zu-Punkt-Betrieb zwischen zwei Geräten ausgelegt. Dank der Flexibilität der General Purpose Input/Output (GPIO) Pins der MCU kann allerdings eine Busarbitrierung durch die verbundenen Geräte selbst durchgeführt werden.

Dazu sind die Empfangs- und Sendeleitungen der USARTs der MCUs zu verbinden. Nachdem Eingänge prinzipbedingt hochohmig ausgeführt sind, können mit einem einzigen MAX3232 (RS-232 zu Smart Devices) bzw. von einer einzigen MCU (Smart Device zu Host) mehrere Smart Devices verbunden werden. Der Autor konnte so vier Geräte an einem einzigen Segment betreiben; die Signalform, gemessen mit einem TDS-2014 Oszilloskop, wies keinen Unterschied zu nur einem einzigen Device auf. Dabei betrug die Leitungslänge insgesamt ca. 80 cm.

Um den Bus nach CSMA/CA zu arbitrieren, wird eine „Trägerwellenerkennung“ aus Aktivität auf der Sendeleitung (Smart Devices zu USB-Host) abgeleitet: Alle MCUs, die nicht senden, trennen ihren Sende-GPIO-Pin von der USART-Peripherie und schalten ihn in den Interrupt-Betriebsmodus. Ein Interrupt wird auf jede Änderung des Pegels ausgelöst.

In der entsprechenden Interrupt Service Routine (ISR) wird ein Timer zurückgesetzt, bei dessen Ablauf wiederum der Backoff-Mechanismus anläuft. So können prinzipiell beliebig viele Teilnehmer am Bussegment betrieben werden. Hindernd steht dem der physikalische Layer entgegen, der nicht für Mehrfachbetrieb ausgelegt ist. In der Praxis funktionierte es aber hinreichend zuverlässig.

Für die erforderliche Zufallsquelle wird ein Pseudo-Zufallsgenerator verwendet, dessen Initialisierung aus dem Rauschen des internen Analog-Digital-Konverters mit maximaler Vorverstärkung generiert wird. In einer Schleife werden dazu 100 konsekutive Werte aufaddiert; die resultierende Zahl stellt zwar keine kryptographisch starke Zufallszahl dar, reicht für den Backoff-Mechanismus allerdings aus.

4.3.2.3 Konfiguration des Anwendungsprotokolls

Der LED-Controller ist über das entwickelte Anwendungsprotokoll ansprechbar. Eine Discovery-Anfrage liefert die Ausgabe in Listing 4.5.

```
<ledcontroller_0001>
  <meta ...>...</meta>
  <led01>
    <bit16   type="int" access="rw" min="0" max="65535"       >0</bit16>
    <bit8    type="int" access="rw" min="0" max="255"         >0</bit8>
    <percent type="int" access="rw" min="0" max="100" unit="%">0</percent>
  </led01>
  <led02>
    <bit16   type="int" access="rw" min="0" max="65535"       >0</bit16>
    <bit8    type="int" access="rw" min="0" max="255"         >0</bit8>
    <percent type="int" access="rw" min="0" max="100" unit="%">0</percent>
  </led02>
  <!-- ... -->
  <led15>
    <bit16   type="int" access="rw" min="0" max="65535"       >0</bit16>
    <bit8    type="int" access="rw" min="0" max="255"         >0</bit8>
    <percent type="int" access="rw" min="0" max="100" unit="%">0</percent>
  </led15>
</ledcontroller_0001>
```

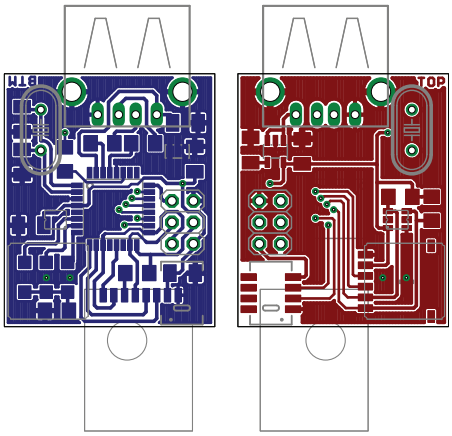
Listing 4.5: Datenpunkte des LED-Controllers (RS-232). Die Werte, die über 8 Bit oder 0 % bis 100 % schreibbar sind, werden intern durch Gammakorrektur der Wahrnehmung des menschlichen Auges angepaßt. Bei Änderung eines Datenpunktes eines Dienstes werden die anderen beiden Werte ebenfalls verändert.

4.3.3 Umweltsensor (2,4 GHz Funk)

Ein weiteres Smart Device stellt der Umweltsensor dar, der neben der MCU (ATmega32U2) mit einer USB-Schnittstelle und mehreren Sensoren zur Erfassung von Umweltgrößen (Temperatur, relative Luftfeuchte, Beleuchtungsstärke und Luftdruck) bestückt werden kann.

Je nach Firmware können drei unterschiedliche Aufgaben erfüllen werden:

1. *USB-Sensor.* In diesem Modus wird der Sensor als USB-Gerät mit virtueller serieller Schnittstelle betrieben und sendet jede Sekunde die aktuellen Meßwerte an den Host.
2. *USB-Funkgateway.* Hiermit werden Daten zwischen USB und dem Funkmedium übersetzt, wobei sich das Gerät als virtuelle Netzwerkkarte mit RNDIS-Protokoll anmeldet. Aufgrund des sehr geringen Speichers von nur 1 kB ist die MTU entsprechend begrenzt; der Router führt in diesem Fall eine IP-Fragmentierung aus. Auch wird hierbei nur Halbduplex unterstützt.



(a) Darstellung der finalen gerouteten Platine aus EAGLE heraus. Die rechte Seite (rot) Linien stellt die Oberseite der Platine dar.

(b) Vollständig und beidseitig bestückter Umweltsensor. Auf der Unterseite befindet sich die MCU, auf der Oberseite (rechts) die Sensoren.

Abbildung 4.7: Umweltsensor in Entwurfsstadium und gefertigtem Zustand.

3. *Funksensor.* Die dritte Firmware konfiguriert die Hardware als Funksensor, der seine Werte über EVENT-Nachrichten des Anwendungsprotokolls bei Änderung maximal einmal pro Sekunde per Broadcast versendet. Jeder Sensor erhält eine eigene ID auf Anwendungsebene und eine eineindeutige MAC-Adresse. Details zum Funkprotokoll werden in Kapitel 4.4 behandelt.

Aufgrund von zu geringem Flash-Speicher und zu wenigen USB-Endpunkten der MCU ist eine Kombination aller drei Funktionen nicht möglich.

4.3.3.1 Funktionsbeschreibung der Hardware

Die verwendete MCU bietet ein USB-Device Modul, das nur noch zwei externe Widerstände benötigt. Insgesamt stehen vier programmierbare Endpunkte zur Verwendung durch die Firmware zur Verfügung. Sie werden je nach Firmware für seriellen Betrieb oder als virtuelle Netzwerkkarte konfiguriert.

Die Stromversorgung der Platine kann über USB erfolgen. Der Mikrocontroller besitzt dazu einen internen Spannungsregler, der die USB-Spannung von 5,0 V auf zirka 3,3 V herabsetzt. Zwar könnte der Prozessor an sich auch mit der höheren Spannung betrieben werden, die verwendeten Sensoren aber nicht. Nachteil der niedrigeren Versorgungsspannung ist eine auf 8 MHz begrenzte Taktfrequenz. Sie wird über einen externen Quarz erzeugt.

Sofern der Sensor extern mit Spannung versorgt wird, kann auf der Oberseite der Platine (Lötpads in Abbildung 4.7b im rechten Teil knapp unter dem USB-Stecker sichtbar) ein passender Spannungsregler und der Abblockkondensator bestückt werden. Auf der Unterseite ist dann ein verbindender 0 Ω-Widerstand wegzulassen.

Die folgenden drei Sensoren wurden bestückt:

- *Temperatur- und Luftfeuchtesensor SHT11 von Sensirion.* Der mit knapp 80 µW sehr sparsame und damit für batteriebetriebene Anwendungen geeignete SHT11 [77] ist über zwei Leitungen an die MCU angeschlossen. Er beinhaltet Signalwandler und Analog-Digital-Konverter, deren Meßwerte digital über ein proprietäres Protokoll ähnlich dem Inter-Integrated-Circuit (I^2C)-Protokoll ausgelesen werden können. Die Auflösung beträgt hierbei typisch 12 bit für die Luftfeuchte und 14 bit bei der Temperatur. Eine Messung beider Werte wird mit einem Befehl gestartet; eine Sekunde später können die konvertierten Werte aus dem Speicher ausgelesen werden.

- *Luftdrucksensor HP03s von HopeRF.* Zur Messung des Luftdrucks kommt ein HP03s [78] der Firma HopeRF zum Einsatz. Der Sensor bietet über I^2C zum einen die bei der Herstellung ermittelten Kalibrierungswerte und zum anderen die Meßwerte zweier Analog-Digital-Konverter an. Mittels dieser Größen kann über eine in der MCU durchzuführende Berechnung der Luftdruck und die Umgebungstemperatur ermittelt werden.

- *Beleuchtungsstärkesensor APDS-9300 von Avago Technologies.* Zur Messung der Beleuchtungsstärke wird der Sensor APDS-9300 [79] verwendet. Er besitzt zwei Fotodioden in einem kompakten Gehäuse und kann ebenfalls über I²C angesprochen werden. Zur korrekten Ermittlung der Beleuchtungsstärke des sichtbaren Lichts mißt eine Diode nur den Infrarot-Anteil, während die zweite Diode die Stärke von Infrarot- und sichtbarem Licht ermittelt. Für Tageslichtzusammensetzung ist im Datenblatt eine Formel zur Berechnung der Beleuchtungsstärke in Lux gegeben.

Da die MCU kein Peripheriemodul für I²C besitzt, wird das Protokoll über zwei GPIO-Leitungen von der Firmware emuliert.

Drei farbige LEDs zur Debugausgabe und Zustandsmeldung runden die Hardware ab.

4.3.3.2 Konfiguration des Anwendungsprotokolls

Der Umweltsensor ist über das entwickelte Anwendungsprotokoll ansprechbar. Eine Discovery-Anfrage liefert die Ausgabe in Listing 4.6.

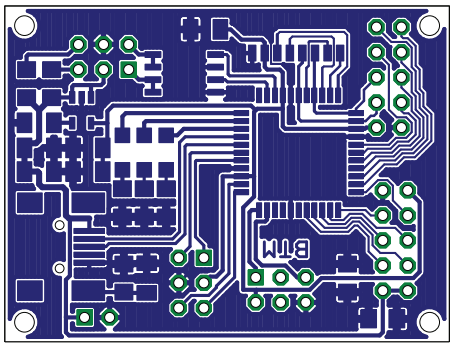
```
<envsensor_0001>
  <meta ...>...</meta>
  <Sensor>
    <relative_Luftfeuchte type="int" access="r" min="0" max="100" unit="%"
        >41</relative_Luftfeuchte>
    <Temperatur type="float" access="r" min="-40.0" max="120.0" unit="°C">
        21.4</Temperatur>
    <Lichteinfall type="float" access="r" min="0" max="500000.0" unit="lx"
        >5641</Lichteinfall>
    <Luftdruck type="float" access="r" min="900.0" max="1200.0" unit="hPa"
        >1012.2</Luftdruck>
  </Sensor>
  <Konfiguration>
    <minimales_Sendeintervall type="int" access="rw" min="1" max="3600"
        unit="s">1</minimales_Sendeintervall>
  </Konfiguration>
</envsensor_0001>
```

Listing 4.6: Datenpunkte des Umweltsensors.

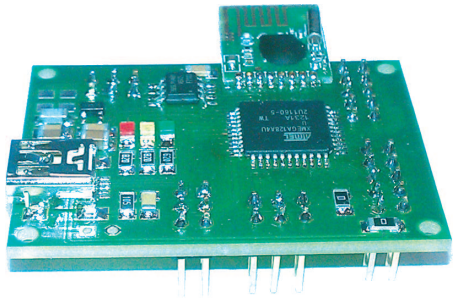
4.3.4 USB-Netzwerkadapter (2,4 GHz Funk)

Der Netzwerkadapter, dargestellt in Abbildung 4.8, basiert auf der ATxmega128A4U-MCU. Es wurde die größte Variante gewählt, da für das Weiterleiten von Ethernet-Frames ein großer Paketspeicher von 8 kB Static-RAM von Vorteil ist. Unterschiedliche Tasks können unabhängig voneinander Frames der USB-Schnittstelle und der Funkschnittstelle in beide Richtungen annehmen. Die MCU wird mit internem RC-Oszillator auf 32 MHz Taktfrequenz betrieben.

In Abbildung 4.1 stellt der Adapter das „Herstellergateway B“ dar.



(a) Darstellung der finalen gerouteten Platine aus EAGLE heraus.



(b) Vollständig und mit Through-Hole-Bauteilen beidseitig bestückter Netzwerkadapter.

Abbildung 4.8: USB-Netzwerkadapter in Entwurfsstadium und bestücktem Zustand.

4.3.4.1 Funktionsbeschreibung der Hardware

Im Unterschied zum Umweltsensor benötigt die USB-Schnittstelle keine externen Widerstände mehr, einzig eine möglichst kurze Verbindung zum Verbindungsstecker ist erforderlich. Am Stecker selbst, hier in Mini-USB-Variante, sorgt ein hochohmiges Tiefpaß-Filter für die Verbindung zwischen Abschirmung und Gerätemasse. Auf der verwendeten MCU stehen 32 Endpunkte zur freien Programmierung zur Verfügung. Mit 128 kB Flash-ROM kann die Hardware daher mehrere USB-Devices (Network Interface Card (NIC), serielle Schnittstelle, etc.) gleichzeitig implementieren.

Die gesamte Schaltung wird durch einen Linearregler mit 3,3 V versorgt. Die meisten unbenutzten GPIOs sind auf Pfostensteckern herausgeführt, welche in diesem Projekt aber nicht verwendet werden. Neben 3,3 V stehen auf diesen auch direkt 5,0 V der USB-Schnittstelle für externe Peripherie (Displays, Schnittstellen, Sensoren, etc.) zur Verfügung. Eventuell angeschlossene Peripherie darf eine Stromaufnahme von 350 mA nicht überschreiten. Der restliche Strom wird bereits für MCU, Sendemodul und LEDs benötigt.

Das RFM70-Funkmodul ist wie in allen Geräten an die MCU mit einer SPI-Schnittstelle verbunden und belegt zwei zusätzliche Leitungen für Chip-Enable und den Interrupt-Request (IRQ)-Kanal.

Drei LEDs zur Zustandsanzeige und Debugausgabe runden die Plattform ab.

4.3.4.2 Konfiguration des Anwendungsprotokolls

Der USB-Netzwerkadapter ist nicht über das Anwendungsprotokoll ansprechbar, sondern dient als reine Ethernetkarte. Ein Listing von Datenpunkten entfällt.

4.3.5 LED-Controller (2,4 GHz Funk)

Der in Kapitel 4.3.2 beschriebene LED-Controller wird für den Funkbetrieb mit einem 2,4 GHz Modul ausgestattet. Eine weitere Änderung ist, daß wegen des Platzbedarfs der neuen Firmware die verwendete MCU gegen die nächstgrößere Variante (ATxmega32A4U) getauscht wurde, welche mit dem internen RC-Oszillator auf 32 MHz Taktfrequenz betrieben wird.

In Abbildung 4.1 wird der LED-Controller durch Smart Device 3 oder 4 repräsentiert.

4.3.5.1 Funktionsbeschreibung der Hardware

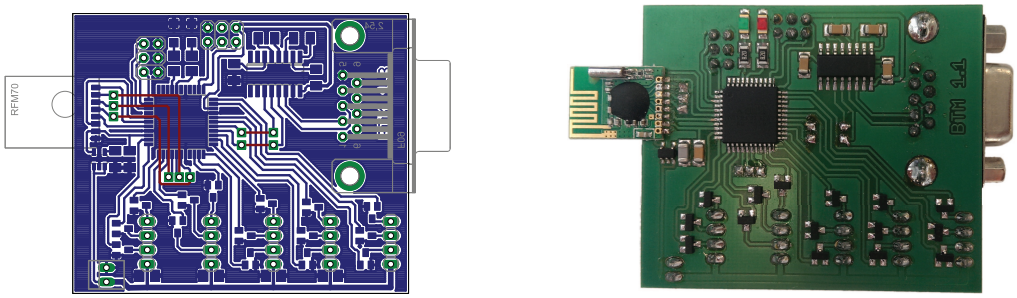
Die USB-Schnittstelle wird nicht verwendet, die serielle Schnittstelle bleibt zu Debugzwecken aber erhalten. Ob sie bestückt wird, kann von Device zu Device entschieden werden.

Weiterhin wurden die IRF7341-MOSFET wegen geringerer Baugröße gegen Einzeltransistoren, IRLML2502 [80], getauscht und die fehleranfälligen Telefonbuchsen durch 4-polige Pfostenleisten ersetzt. Der niedrigere maximale Strom und die geringere Wärmeabgabeleistung spielen für die Anwendung keine Rolle.

Die Stromversorgung kommt durch einen 3,3 V-Linearregler zustande, der in der Bauform SOT23-5 gefertigt ist. Bei 12 V Eingangsspannung, wie von einigen vollfarbigen LED-Streifen benötigt, ist die Erwärmung deutlich spürbar. In einer weiteren, für dieses Projekt allerdings nicht mehr zum Einsatz kommenden Variante wurde deswegen die größere Gehäuseform SOT223 für den Linearwandler gewählt.

4.3.5.2 Konfiguration des Anwendungsprotokolls

Die Beschreibung aus Listing 4.5 ist auch für diese Variante gültig.



(a) Darstellung der finalen gerouteten Platine aus EAGLE heraus.

(b) Bestückter LED-Controller.

Abbildung 4.9: LED-Controller mit RFM70-Funkmodul in Entwurfsstadium und bestücktem Zustand.

4.3.6 Generische Hardwareplattform für Smart Devices (2,4 GHz Funk)

Die bislang konstruierten Smart Devices wurden jeweils für eine spezifische Aufgabe entworfen. Abschließend für diese Beschreibung wird daher der in Kapitel 1 ermittelten Anforderung nach einer möglichst kurzen Time-to-Market Rechnung getragen und eine Vorlage für eine erweiterbare Hardware erstellt.

Dabei handelt es sich um ein einseitiges Layout einer Platine, die neben der MCU und ihrer Stromversorgung auch das Funkmodul, ein Grafikdisplay und einige Tasten als Benutzerschnittstelle beinhaltet. Weiterhin bietet die Platine freie Fläche zur Platzierung eigener Peripherie, die direkt an die MCU angeschlossen werden kann.

Ziel ist weniger ein fertiges Produkt, das externe Teile über Pfostenstecker anbindet (wie in der Industrie häufiger zu finden), sondern eine von der Benutzerschnittstelle und MCU-Plattform her vollständige Platine, die um anwendungsspezifische Bauteile erweitert und mittels Rapid-Prototyping in Kleinserien gefertigt wird. Da im Idealfall nur eine einziges Printed Circuit Board (PCB) zu fertigen ist, können so die Stückkosten geringer gehalten werden. Allerdings besteht auch hier weiterhin die Option auf Zusatzplatinen, wenn etwa große Bauteile für Leistungselektronik erforderlich sind.

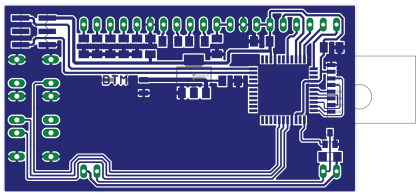


Abbildung 4.10: EAGLE-Entwurf der generischen Hardwareplattform.

Da sich die Verbindungen von auf der Plattform standardmäßig vorhandenen Bauteilen nicht ändern, kann ein Teil der Firmware für jedes auf der Plattform aufbauende Produkt verwendet werden. Lediglich für weitere Funktionen müssen Treiber bzw. Softwaremodule geschrieben werden. Somit verringert sich auch bei der Firmware die Entwicklungszeit.

Das Konzept wird an zwei unterschiedlichen Smart Devices in Hard- und Software demonstriert. Abbildung 4.10 zeigt die Plattform, auf der noch keine weitere Peripherie enthalten ist.

In Abbildung 4.1 wird die Plattform durch Smart Device 3 oder 4 repräsentiert.

4.3.6.1 MCU-Plattform und Stromversorgung

Neben einem Steckplatz für eine MCU der xmega-A4U-Baureihe im TQFP44-Gehäuse ist auf der Platine eine Stromversorgung über einen Linearregler im SOT223-Gehäuse integriert. Sie versorgt die gesamte Schaltung mit 3,3 V Spannung. Der Regler kann je nach Hersteller und Eingangsspannung bis zu 500 mA Strom ohne zusätzlichen Kühlkörper liefern; eine integrierte Übertemperaturabschaltung verhindert dauerhaften Schaden.

Die Ports *A*, *E* und *R* der MCU sind unbelegt und stehen für eigene Erweiterungen zur Verfügung. Da die Pins dieser Ports teilweise nicht mehr mit einseitigem Layout erreichbar sind, empfiehlt sich ggf. eine doppelseitige Platine. Für die Smart Devices in dieser Arbeit war das allerdings nicht notwendig.

Alle ICs wurden mit den erforderlichen Abblockkondensatoren ausgestattet, wobei für zweipolige SMT-Bauteile nach Möglichkeit die Bauform 1206 verwendet wird.

4.3.6.2 Programmierschnittstelle

Die MCU wird über das beschrieben ISP-Verfahren, unabhängig von der gewählten Variante, programmiert, wobei die Pinbelegung der Schnittstelle (siehe Abbildung 4.11) der offiziellen Empfehlung der Firma Atmel entspricht. Vom Programmieradapter erfolgt dabei die Versorgung der Schnittstelle mit Taktfrequenz und Daten.

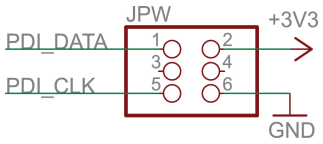


Abbildung 4.11: PDI-Programmierschnittstelle für xmega-AVRs

Sofern an der 3,3 V-Versorgung keine weitere Peripherie angeschlossen ist, kann die Programmierung des Smart Devices stromlos erfolgen; der Programmieradapter übernimmt dann die Spannungsversorgung. Bei größerer Last muß allerdings die Versorgung durch die Schaltung selbst erfolgen und am Programmieradapter deaktiviert werden. Das STK-600 bietet dafür eine Steckbrücke („Jumper“).

4.3.6.3 Grafikdisplay

Zur flexiblen Ausgabe von Informationen ist ein Grafikdisplay der Firma Electronic Assembly auf der Steuerung integriert. Es ist Teil der vielfältigen Serie DOG-M [81] mit verschiedenen äußeren Abmessungen und inneren Eigenschaften (Auflösung, Farbe, Text- oder Grafikdisplay).

Alle Liquid Crystal Display (LCD)s der Serie erfordern auf der Platine allerdings stets die selben bzw. nur in Y-Richtung versetzten Bohrlöcher (siehe Abbildung 4.12). Die Steuerung ist so mit unterschiedlich großen Displays an jeweilige Anwendungen anpassbar. Die Displays werden in Through-Hole-Technik auf der Platine montiert.

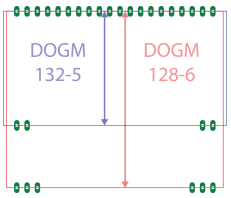


Abbildung 4.12: Unterschiedliche Abmessungen der EA-DOGM-LCDs.

Das LCD wird über SPI mit der MCU verbunden und belegt daneben zwei weitere Leitungen. Die eine davon dient der Unterscheidung von Befehlen und Daten am SPI-Bus, die zweite setzt den Controller des LCD zurück. Das ist insbesondere zur Beseitigung eines undefinierten Zustandes nach Einschalten der Stromversorgung notwendig. Die Taktrate der SPI-Clock kann bis zu 30 MHz betragen.

An das Display können Daten nur geschrieben werden; der Controller garantiert eine Verarbeitung innerhalb der zulässigen Taktfrequenz. Da der Controller keine Schriftsätze unterstützt, müssen geeignete Pixelzuordnungen („Schrift-Rendering“) im Mikrocontroller vorgenommen werden. Dazu hält die MCU das gesamte darzustellende Bild („Frame“) stets in einem RAM-Array bereit. Innerhalb der Hauptschleife werden Manipulationen des

Bildes direkt im RAM durchgeführt und am Ende der Schleife bei Bedarf („dirty“-Flag) auf das LCD geschrieben. Eine genauere Beschreibung findet in Kapitel 4.5 statt.

Für die Smart Devices dieser Arbeit wurde das EA DOGM132-5 gewählt [82]. Es ist ein einfarbiges Grafikdisplay mit einer Auflösung von 132x32 Pixel und kann in mehreren Varianten (hier: blau/weiß, orange/schwarz, weiß/schwarz, schwarz/rot) hintergrundbeleuchtet werden. Der Anschluß der Hintergrundbeleuchtung erfolgt direkt an die 3,3 V Spannungsversorgung, damit entfällt ein Anpassen des Vorwiderstandes bei verschiedenen Eingangsspannungen. Außerdem kann die Beleuchtung über den zwischengeschalteten Transistor BSS123 über einen PWM-Kanal der MCU in 16 bit Auflösung gedimmt werden. Eine Gammakorrektur ist in der Firmware bereits vorgesehen.

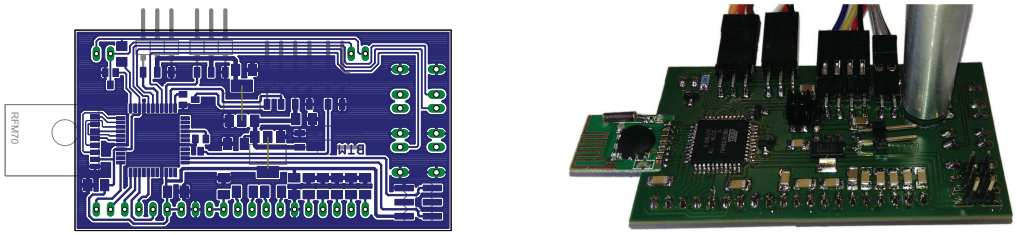
4.3.6.4 Benutzerschnittstelle – Eingabe

Als Benutzerschnittstelle bietet die Plattform drei neben dem LCD platzierte Kurzhub-Taster. Sie können unabhängig voneinander abgefragt werden und sind an Interrupt-fähigen GPIO-Pins angeschlossen. Bei entsprechender Programmierung können sie zum Wecken der MCU aus einem Sleep-Mode verwendet werden.

Ist die Steuerung für den Einbau in ein Gehäuse (z.B. hinter eine Frontblende) vorgesehen, so kann durch geeignete Wahl der Länge des Knopfes eine der Abstand zur Frontblende bis zu 1 cm betragen. Bei Tastenlängen ab 5 mm empfiehlt sich eine Fixierung der Taster durch Bohrlöcher, da sie seitlicher Belastung nicht standhalten.

4.3.7 Tischventilator (2,4 GHz Funk)

Das erste realisierte Smart Device ist ein Modell eines dreh- und schwenkbaren Tischventilators. Dazu wird ein regulärer PC-Lüfter in ein T-Gestell montiert, das von zwei Modellbauservos bewegt werden kann.



(a) Darstellung der finalen gerouteten Platine aus EAGLE heraus.

(b) Bestückte Steuerung des Ventilatormodells.

Abbildung 4.13: Ventilator-Controller mit RFM70-Funkmodul in Entwurfsstadium und bestücktem Zustand. An der oberen Seite sind die Steckverbinder für die externe Peripherie (Servos, Lüfter) erkennbar. Der silberne Zylinder auf der linken rechten Seite stammt aus einer USB-Schwanenhalslampe und dient der Befestigung der Platine auf dem Holzbrett.

Über die Funkschnittstelle können Parameter wie X/Z-Rotation und -Position gesetzt werden. Außerdem läßt sich die Geschwindigkeit des Lüfters in Prozentschritten verändern und die aktuelle Drehzahl auslesen. Weiterhin können Helligkeit und Kontrast des LCDs verändert werden.

Über LCD und Tasten können alle Parameter auch über ein Menüsystem direkt verändert werden. Abbildung 4.15 zeigt den Inhalt des Übersichtsbildschirms, in dem wichtige Geräteparameter gemeinsam angezeigt werden. Nur über das Menü sind Konfigurationswerte zur Kalibrierung der Positionen veränderbar. So wird das Konzept der getrennten Zugänglichkeit von Maschinenwerten am Gerät selbst und über das Anwendungsprotokoll demonstriert. Die Steuerung ist in Abbildung 4.13 dargestellt, das vollständige Gerät zeigt Abbildung 4.14 dargestellt.

Abbildung 4.15: LCD-Inhalt des Modells im Hauptbildschirm.

4.3.7.1 Funktionsbeschreibung der Hardware

Die Elektronik basiert auf der in Kapitel 4.3.6 vorgestellten generischen Steuerung. Sie wird erweitert, um die folgenden externen Hardwareteile ansteuern zu können:

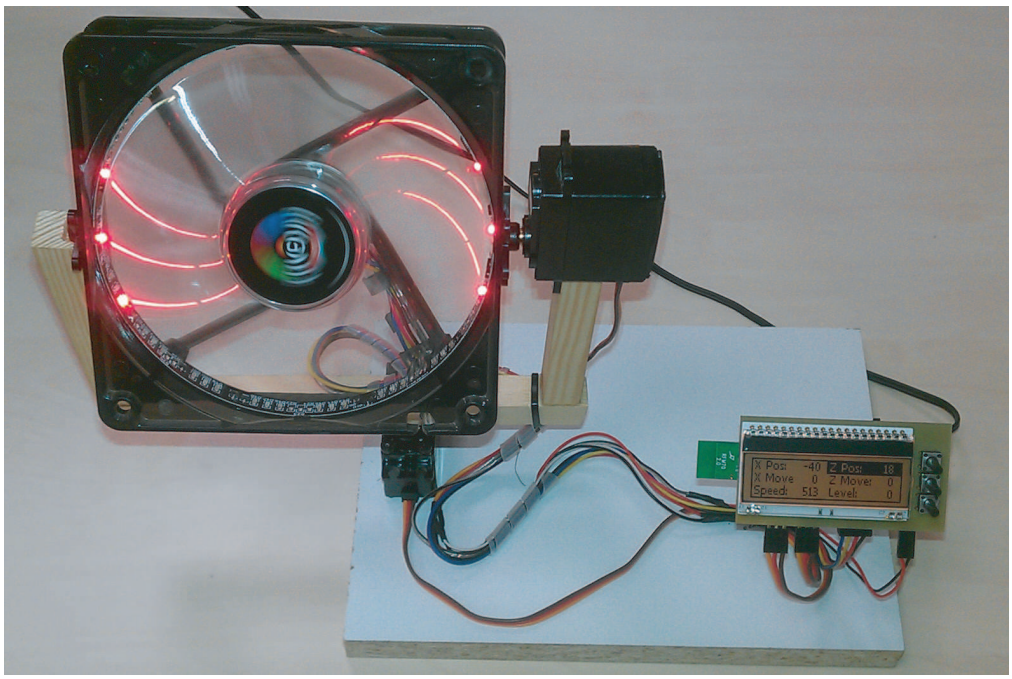


Abbildung 4.14: Gesamtdarstellung des Ventilatormodells mit Steuerung und Lüfteraufnahme. Gut erkennbar ist im Bild die eingeschaltete LED-Beleuchtung sowie die leicht in X-Richtung geschwenkte Halterung.

- Digitale Modellbauservos für X- und Z-Achse
- PC-Lüfter (120 mm) für Gehäuse
- Open-Drain-Ausgang zur Farbsteuerung des Lüfters

Wegen der für den Lüfter erforderlichen 12 V Betriebsspannung wird die gesamte Schaltung mit dieser Eingangsspannung betrieben. Der Linearregler für 3,3 V ist dabei bei aktivierter Hintergrundbeleuchtung mit bis zu 200 mA belastet und erwärmt sich nur unwesentlich. Der zweite Linearregler für die Betriebsspannung der Servos (5 V) dagegen benötigt einen externen aufgeklebten Kühlkörper bei Dauerbetrieb der beiden Servos, da sie Stromstärke in Spitzen bis zu 10 A beträgt.

Die Modellbauservos, die den Lüfter in waagrechter und senkrechter Achse bewegen, werden über PWM angesteuert. Sie besitzt eine Grundfrequenz von 50 Hz und eine Periodenlänge von 1 ms bis 2 ms. Geringere oder größere Werte sind möglich, beschädigen dann aber unter Umständen das Servo. In digitalen Schaltungen wird eine logische `1` in der Regel ab der Hälfte der Versorgungsspannung gewertet; im Falle von 5 V liegt die Schwelle bei 2,5 V. Da die Ausgangsspannung der GPIO-Pins der MCU 3,3 V beträgt, werden die hochohmigen Steuereingänge der beiden Servos direkt mit der MCU verbunden.

Der PC-Lüfter besitzt einen Open-Drain Tachometer-Ausgang, welcher pro Umdrehung zwei Impulse liefert. Aus Zeitmessung zwischen den Impulsen und der Taktfrequenz der MCU kann die Drehzahl bestimmt werden. Wegen des Open-Drain-Charakters des Ausgangs wird der GPIO-Pin als Eingang mit aktiviertem Pull-Up-Widerstand betrieben.

Der PWM-Steuereingang ist als Transistor-to-Transistor Logic (TTL)-Eingang für eine Grundfrequenz von 22 kHz bis 28 kHz ausgelegt. Er bietet je nach Hersteller einen Spannungspegel von 3,3 oder 5 V, welcher für eine logische `0` gegen Masse geschlossen wird. Wegen der Unsicherheit der Spannung und zur besseren Filterung von Rauschen wird der Eingang über einen BSS123-Transistor mit der MCU verbunden, wobei die MCU das Gate ansteuert. Die Strombegrenzung findet im Lüfter statt. Die Pulsbreite verändert bei dem verwendeten Modell *Enermax T.B. Vegas Trio* die Drehzahl in einem Bereich von ca. 500 1/min bis 1800 1/min; der Lüfter hält eine Mindestdrehzahl ein.

Weiterhin enthält der Lüfter mehrfarbige LEDs zur Beleuchtung der Rotorblätter. Im Original lassen sich diese durch einen Schalter, der eine 5 V Versorgungsspannung gegen Masse kurzschließt, aktivieren und in ihrem Blinkmuster ändern. Die Steuerung simuliert diesen Schalter durch einen BSS123-Transistor und kann die Blinkmuster ebenfalls ansteuern. Wegen fehlender Rückmeldung wird dabei auf ein großzügiges Timing für die „Tastendrücke“ zurückgegriffen.

4.3.7.2 Menüstruktur der Firmware

Neben dem ersten Bildschirm, auf dem aktuelle Betriebsparameter (Drehzahl, X-Position, Z-Position, etc.) angezeigt werden, bietet das Gerät die folgenden Einstellungsmöglichkeiten über eine Menüauswahl. Das Menü wird mit der oberen bzw. unteren Taste traversiert; Einträge werden mit der mittleren Taste bestätigt. In der folgenden Auflistung mit einem

Asterisk (*****) gekennzeichnete Punkte sind nur am Gerät selbst, nicht aber über das Anwendungsprotokoll zugänglich.

- *Servo X Optionen.* Bietet Zugriff auf die Konfiguration des Modellbauservos in X-Richtung.

  - Position. Setzt die Position auf einen bestimmten Wert zwischen −90 und 90.
  - Oszillation. Schreibt die Geschwindigkeit, mit der das Servo automatisch einen Schwenk zwischen den maximalen Werten abfährt.
  - Negativer Anschlag*****. Kalibriert den negativen Anschlag des Servos in einzelnen PWM-Schritten. Das Servo fährt zu Kontrolle den jeweils eingestellten Wert an. Der Lüfter sollte waagrecht positioniert ein, wobei der Luftstrom abwärts gerichtet ist.
  - Nullposition*****. Kalibriert die Neutralstellung des Lüfters. Er soll sich in dieser Position genau senkrecht befinden.
  - Positiver Anschlag*****. Kalibriert den positiven Anschlag des Servos in einzelnen PWM-Schritten. Das Servo fährt zu Kontrolle den jeweils eingestellten Wert an. Der Lüfter sollte waagrecht positioniert ein, wobei der Luftstrom aufwärts gerichtet ist.

  Anzumerken ist, daß selbst bei feinfühlig möglicher Einstellung der PWM-Pulsbreite insbesondere preiswerte digitale Modellbauservos nur grobe Stufen, z.B. ganze Grad, anfahren. Im ungünstigen Fall „zittert“ das Servo bei einer bestimmten Vorgabe durch die MCU. Abhilfe schafft hier nur ein anderes Servo.

- *Servo Z Optionen.* Bietet Zugriff auf die Konfiguration des Modellbauservos in Z-Richtung.

  - Position. Siehe oben.
  - Oszillation. Siehe oben.
  - Negativer Anschlag*****. Kalibriert den negativen Anschlag des Servos in einzelnen PWM-Schritten. Das Servo fährt zu Kontrolle den jeweils eingestellten Wert an. Der Lüfter sollte vom Betrachter aus gesehen vollständig nach links geschwenkt sein.
  - Nullposition*****. Kalibriert die Neutralstellung des Lüfters. Er soll sich in dieser Position genau dem Betrachter zugewandt befinden.
  - Positiver Anschlag*****. Kalibriert den positiven Anschlag des Servos in einzelnen PWM-Schritten. Das Servo fährt zu Kontrolle den jeweils eingestellten Wert an. Der Lüfter sollte vom Betrachter aus gesehen vollständig nach rechts geschwenkt sein.

- *Fan-Optionen.* Ermöglicht den Zugriff auf Attribute des Lüfters.

  - Level. Setzt die Drehzahl in Prozentschritten. Der verwendete Lüfter hält allerdings eine Mindestdrehzahl und läßt sich nicht vollständig stoppen.

  - LED-Programm. Wählt ein Programm der LED-Beleuchtung aus. Mögliche Werte sind 0 bis 3, wobei der Wert 0 die Beleuchtung ausschaltet, 1 für Dauerleuchten, 2 für Blinken und 3 für „Ventilatormodus“ steht.
  - LED-Farbe. Wählt bei aktivem Programm eine Farbe. Es kann zwischen rot, grün, blau und weiß gewählt werden.

- *LCD-Optionen.* Bietet Einstellungen für das LCD.
  - Helligkeit. Bietet die Option, die Helligkeit in Prozentschritten zu setzen. Eine Gammakorrektur findet im Gerät statt, so daß der subjektive Helligkeitsverlauf bei aufeinanderfolgenden Werten annähernd linear erscheinen.
  - Kontrast. Setzt den Kontrastwert des LCD.

- *Netzwerkinformationen*.* Gibt Informationen zur eigenen IP- und MAC-Adresse sowie der verbleibenden DHCP-Leasetime aus.

- *Einstellungen speichern*.* Speichert die aktuellen Werte der Servokalibrierung und LCD-Einstellungen dauerhaft im EEPROM der MCU.

4.3.7.3 Konfiguration des Anwendungsprotokolls

Das Lüftermodell ist über das entwickelte Anwendungsprotokoll ansprechbar. Eine Discovery-Anfrage liefert die Ausgabe in Listing 4.7.

```
<fancontrol_0001>
  <meta ...>...</meta>
  <X-Achse>
    <Winkel type="int" access="rw" min="-90" max="90" unit="°">
      0
      <label from="0"          >Neutralstellung</label>
      <label from="-90" to="-1">Luftstrom abwärts gerichtet</label>
      <label from="1"   to="90">Luftstrom aufwärts gerichtet</label>
    </Winkel>
    <Oszillationsstufe type="int" access="rw" min="0" max="2">
      0
      <label from="0">Keine Oszillation</label>
      <label from="1">Schwache Oszillation</label>
      <label from="2">Mittlere Oszillation</label>
    </Oszillationsstufe>
  </X-Achse>
  <Z-Achse>
    <Winkel type="int" access="rw" min="-90" max="90" unit="°">
      0
      <label from="0"          >Neutralstellung</label>
      <label from="-90" to="-1">Links geschwenkt</label>
      <label from="1"   to="90">Rechts geschwenkt</label>
    </Winkel>
    <Oszillationsstufe type="int" access="rw" min="0" max="2">
      0
      <label from="0">Keine Oszillation</label>
      <label from="1">Schwache Oszillation</label>
      <label from="2">Mittlere Oszillation</label>
```

```
29      </Oszillationsstufe>
30    </Z-Achse>
31    <Ventilator>
32      <Drehzahl type="int" access="r" min="0" max="2000" unit="1/min">512</
            Drehzahl>
33      <Stufe type="int" access="rw" min="0" max="100" unit="%">0</Stufe>
34      <Umschalteingang type="int" access="w" min="1" max="1" free-event-
            slots="2"></Umschalteingang>
35    </Ventilator>
36    <Display-Einstellungen>
37      <Helligkeit type="int" access="rw" min="0" max="100" unit="%">100</
            Helligkeit>
38      <Kontrast type="int" access="rw" min="1" max="10">6</Kontrast>
39    </Display-Einstellungen>
40  </fancontrol_0001>
```

Listing 4.7: Datenpunkte des Lüftermodells.

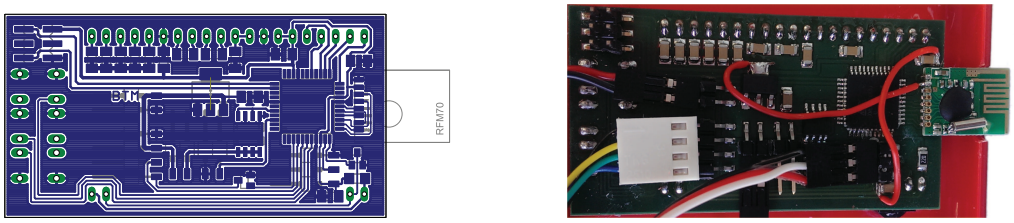
4.3.8 Modellkühlschrank (2,4 GHz Funk)

Das zweite konstruierte Smart Device modelliert die Funktionen eines Kühlschranks. Neben einer Kammer mit integriertem Kühlteil (Peltierelement, Kühlkörper und Ventilator) besitzt das Modell einen Schalter, der eine geöffnete Tür anzeigt. Abbildungen 4.17 und 4.16 zeigen Detailaufnahmen des Geräts.

Die Kühlung wird durch eine elektrische Wärmepumpe (Peltierelement) bewirkt, das im Boden des Gerätes montiert ist. Seine warme Seite zeigt nach unten und endet in einem Kühlkörper. Zur Wärmeabfuhr wird dieser von einem Lüfter mit Umgebungsluft versorgt; die erwärmte Luft wird hinten am Gehäuse ausgeführt.

Aus Kapitel 4.3.6 wird die generische Steuerung zur harmonischen Farbgebung mit einem schwarzen Grafik-LCD (rote Schrift) bestückt und direkt in die Tür montiert. Sie steuert die folgende externe Hardware an:

- Peltierelement, 9 V bis 15 V Eingangsspannung, bis zu 6 A Stromaufnahme

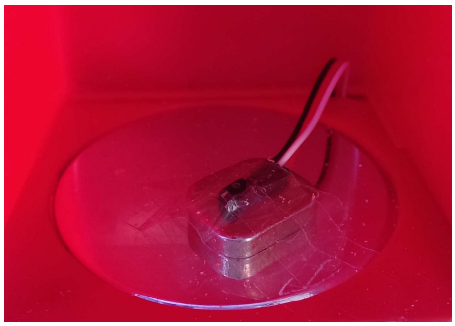


(a) Darstellung der finalen gerouteten Platine aus EAGLE heraus.

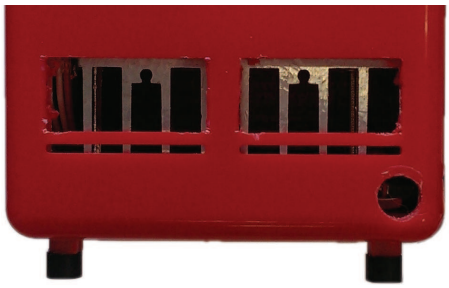
(b) Bestückter LED-Controller.

Abbildung 4.16: LED-Controller mit RFM70-Funkmodul in Entwurfsstadium und bestücktem Zustand.

(a) Vollständige Abbildung des Modells in der Frontansicht.



(b) Mit einem Metallklotz simuliertes Kühlgut. Der Temperaturfühler ist mit Klebefilm befestigt.



(c) Gehäuseausschnitte zur besseren Wärmeableitung auf der Rückseite des Modells. Der Kühlkörper ist im Hintergrund erkennbar.

Abbildung 4.17: Detailaufnahmen des Kühlschrankmodells.

- Lüfter zur Kühlung des Peltierelements, 12 V Betriebsspannung, Tachometerausgang, PWM-Steuereingang
- Temperaturfühler für Kühlkörper- und Kühlguttemperatur, angebunden über I^2C
- Digitaler Schalter zur Anzeige einer offenen Tür

Das verwendete Plastikmodell ist vom Hersteller ursprünglich für den Betrieb an USB vorgesehen und nimmt daher maximal 2,5 W auf. Der original montierte Lüfter wird deswegen gegen ein 12 V-Modell mit den bereits beschriebenen weiteren Merkmalen getauscht. Am Gehäuse werden mehrere Ausschnitte vorgenommen, um eine bessere Wärmeabfuhr bei höherer Leistung des Peltier-Elements zu ermöglichen.

4.3.8.1 Funktionsbeschreibung der Hardware

Die Eingangsspannung beträgt für den Betrieb von Lüfter und Peltierelement 12 V. Wie bereits beim Ventilator wird der Lüfter zur Geschwindigkeitsmessung direkt mit der MCU verbunden und erhält das PWM-Signal zur Drehzahlsteuerung über einen trennenden BSS123-MOSFET.

Das Peltierelement wird über einen bereits beschriebenen Hochstromtransistor (IRF7341) als Low-Side-Switch mit der Steuerung verbunden. Beide Transistoren im Gehäuse sind parallel geschaltet und können ohne Kühlkörper etwa mit insgesamt bis zu 8 A ohne nennenswerte Erwärmung belastet werden. Das verwendete Peltierelement hat bei 12 V eine gemessene Stromaufnahme von 3,7 A; die Dimensionierung ist ausreichend bemessen. Das Peltierelement kann über PWM mit bis zu 16 bit Auflösung gesteuert werden.

Im Modell sind zwei kalibrierte digitale Temperatursensoren (DS18B20 [83], Abbildung 4.18) mit 12 bit Auflösung des Digital/Analog-Wandlers integriert. Sie werden über den 1-Wire-Bus [84] angeschlossen und messen die Temperaturen des Peltierelement-Kühlkörpers sowie des Kühlguts.

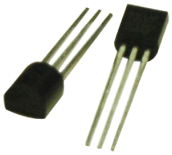


Abbildung 4.18: 1-wire DS 18B20-Temperatursensor

Beide Größen dienen der Regelung der Leistung des Peltierelements und der Drehzahl des Lüfters. Das Protokoll wird in Software durch aufeinanderfolgendes Umschalten des GPIO-Pins zwischen Eingangs- und Ausgangsbetriebsart emuliert; für die Sensoren steht zudem – wie im Datenblatt empfohlen – eine permanente Spannungsversorgung mit 3,3 V zur Verfügung.

Der Schalter zur Prüfung der offenen Tür ist direkt mit einem GPIO-Pin (aktivierter Pull-Up-Widerstand) verbunden. Er wird durch die Firmware zyklisch abgefragt, wobei auch ein Interrupt-Betrieb denkbar wäre.

4.3.8.2 Menüstruktur der Firmware

Neben dem ersten Bildschirm, auf dem aktuelle Betriebsparameter (Lüfterdrehzahl, Temperaturen, etc.) angezeigt werden, bietet das Gerät die folgenden Einstellungsmöglichkeiten über eine Menüauswahl. Das Menü kann mit der oberen bzw. unteren Taste navigiert werden; Einträge werden mit der mittleren Taste bestätigt. Mit einem Asterisk (*) gekennzeichnete Punkte sind nur am Gerät selbst zugänglich.

Abbildung 4.19: LCD-Inhalt des Modells im Hauptbildschirm.

- *Kühlschrank-Optionen.* Bietet Zugriff auf die Konfiguration des Kühlschranks und des Peltierelements.
  - Zieltemperatur. Setzt die gewünschte Temperatur für das Kühlgut.

  - Peltier, maximale Temperatur. Legt die maximale Temperatur des Peltierelements fest. Kann trotz voll aktiviertem Lüfter diese Temperatur nicht gehalten werden, so wird die Leistung des Peltierelements reduziert.
  - Peltier, maximale Lüfterdrehzahl. Beschränkt die Drehzahl des verwendeten Kühlkörperlüfters.

- *LCD-Optionen.* Bietet Einstellungen für das LCD.
  - Helligkeit. Bietet die Option, die Helligkeit in Prozentschritten zu setzen. Eine Gammakorrektur findet im Gerät statt, so daß die Werte annähernd linear erscheinen.
  - Kontrast. Setzt den Kontrastwert des LCD.

- *Netzwerkinformationen*.* Gibt Informationen zur eigenen IP- und MAC-Adresse sowie der verbleibenden DHCP-Leasetime aus.

- *Einstellungen speichern*.* Speichert die aktuellen Werte der Kühlschrankkonfiguration und LCD-Einstellungen dauerhaft im EEPROM der MCU.

4.3.8.3 Konfiguration des Anwendungsprotokolls

Das Lüftermodell ist über das entwickelte Anwendungsprotokoll ansprechbar. Eine Discovery-Anfrage liefert die Ausgabe in Listing 4.8.

```
<fridge_0001>
  <meta ...>...</meta>
  <Kuehlschrank>
    <Aktuelle_Temperatur type="float" access="r" min="-55.0" max="-125.0"
        unit="°C">15.3</Aktuelle_Temperatur>
    <Solltemperatur type="float" access="rw" min="4.0" max="18.0" unit="°C
        ">12.0</Solltemperatur>
    <Tuer type="int" access="r" min="0" max="1">
      100
      <label from="0">geschlossen</label>
      <label from="1">offen</label>
    </Tuer>
  </Kuehlschrank>
  <Peltier>
    <Aktuelle_Temperatur type="float" access="r" min="-55.0" max="-125.0"
        unit="°C">37.3</Aktuelle_Temperatur>
    <Maximale_Temperatur type="float" access="rw" min="-55.0" max="-125.0"
         unit="°C">40.0</Maximale_Temperatur>
    <Leistung type="int" access="r" min="0" max="100" unit="%">100</
        Leistung>
  </Peltier>
  <Ventilator>
    <Drehzahl type="int" access="r" min="0" max="10000" unit="1/min">3165
        </Drehzahl>
    <Maximale_Drehzahl type="int" access="rw" min="1000" max="6000" unit="
        1/min">4500</Maximale_Drehzahl>
  </Ventilator>
  <Display-Einstellungen>
```

```
22      <Helligkeit type="int" access="rw" min="0" max="100" unit="%">100</
            Helligkeit>
23      <Kontrast type="int" access="rw" min="1" max="10">6</Kontrast>
24    </Display-Einstellungen>
25  </fridge_0001>
```

Listing 4.8: Datenpunkte des Kühlschrankmodells.

4.4 IP über proprietäre Funkmodule

Neben eines Demonstrators (LED-Controller), der über die serielle Schnittstelle kommuniziert, wurde für die anderen Smart Devices eine Funkübertragung über das RFM70-Modul realisiert.

Ursprünglich war nur eine Übertragung alleine des Anwendungsprotokolls über die Pakete der Funkmodule vorgesehen; im Laufe der Arbeit wurde dies durch eine Implementierung mit vollständig übertragenen Ethernet-Frames und darauf aufsetzendem Internet Protocol, Version 4 (IPv4)-Protokoll realisiert. Die Implementierung wurde in [85] erfolgreich umgesetzt, siehe auch [86].

Von den in Abbildung 4.1 beschriebenen Varianten realisiert dieses Konzept die Variante *B*. Ziel ist zu zeigen, daß selbst über 30 Jahre alte Protokolle (IPv4 und darauf aufsetzende Techniken) noch immer Gültigkeit besitzen und mittlerweile auf kleinsten MCU ohne weitere Hardware implementiert werden können.

Zu untersuchen ist dabei die Performanz bezüglich Datenübertragungsrate und Latenzzeit von Smart Devices, die neben dem Anwendungsprotokoll auch einen vollständigen Ethernet- und IP-Stack implementieren.

Eine Skalierbarkeit wird im Rahmen der erstellten Hardware (fünf LED-Controller, zwei Kühlschränke, drei Ventilatoren) für die Implementierung mit RFM70-Modulen und den xmegaAVR-MCU in Kapitel 5 evaluiert.

4.4.1 Netzwerkkonfiguration und -Dienste

Jedes Smart Device agiert als eigenständiges IP-Endgerät mit eigener MAC- und IP-Adresse, die es über DHCP erhält. Im Internet-Router wird dazu für die Smart Devices ein eigenes IP-Netz (Zone) aufgespannt, was eine Trennung von den restlichen Geräten – insbesondere deren Broadcasts – zur Folge hat.

Weiter sind im Internet-Gateway Forwarding-Regeln zwischen den Zonen konfiguriert; Netzwerkclients und Smart Devices in verschiedenen Netzen können unicast miteinander kommunizieren. Broadcasts dagegen werden durch das beschriebene `udp-broadcast-relay` nur auf einem einzigen Port, 1045, zwischen den Segmenten sowie im Smart Device Segment selbst weitergeleitet.

4.4.1.1 Adressierung auf Schicht 2

Die 6 Byte lange Ethernet-MAC-Adresse wird jedem Smart Device in einem festen Bereich seines EEPROMs einmalig konfiguriert und ändert sich bei nachfolgenden Programmierungen nicht mehr. Durch das gesetzte „Locally-Administered"-Bit der Adresse[1] in einem Ethernet-Frame, siehe Abbildung 4.20, ist eine Kollision mit Geräten anderer Hersteller im Netzwerk nicht wahrscheinlich, da deren Adressen in der Regel und auch im vorliegenden Szenario global vergeben wurden [87, Kapitel 3].

Das Mapping auf die erste RFM70-Empfangsadresse erfolgt über ein 1:1-Kopieren der unteren 5 B der MAC-Adresse; dadurch können zwar Kollisionen zwischen Smart Devices auf Ebene des RFM70 auftreten, wenn sich die RFM70-MAC-Adressen nur im MSB unterscheiden. Sie würden aber durch die Schicht 2 (Tagged-MAC-Frame) abgefangen, da hier alle 6 B verglichen werden. Im Beispielszenario allerdings werden alle MAC-Adressen selbst vergeben, daher tritt ein solcher Fall nicht auf.

Die zweite Adresse des RFM70-Funkmoduls dient dem Empfang von Broadcast-Paketen und trägt daher in jedem Smart Device die Bytewerte `FF:FF:FF:FF:FF`. Multicast-Protokolle, die nur ein gesetztes Broadcast-Flag (erstes Bit des MSB) aufweisen, ansonsten aber für das jeweilige Protokoll spezifische MAC-Adressen erfordern (z.B. `01:00:5E:??:??:??` bei IPv4-Multicast), werden von der Funkhardware nicht unterstützt, da nur zwei unabhängige MAC-Adressen für den Empfang vergeben werden können[2].

Der USB-NIC verwirft nun bei vom Internet-Gateway eintreffenden Paketen das MSB zur Adressierung auf RFM70-Ebene und leitet das Frame über ein besonderes Protokoll (siehe Kapitel 4.4.2) an Smart Devices.

Von Smart Devices versendete Frames verwenden den selben Mechanismus und adressieren das Frame auf Schicht 2 an eine bestimmte MAC-Adresse (z.B. RFM70 – `00:00:00:00:12`, MAC-Frame – `02:00:00:00:00:12`) oder verwenden die Broadcast-MAC-Adresse (RFM70 – `FF:FF:FF:FF:FF`, MAC-Frame – `FF:FF:FF:FF:FF:FF`).

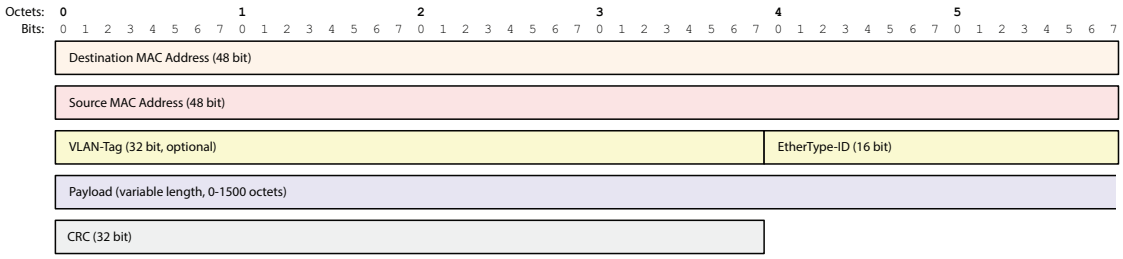


Abbildung 4.20: Darstellung eines Ethernet Tagged-MAC-Frames [87, Kapitel 4.3.2].

1 Anmerkung: Bit 2 im Most Significant Bit (MSB) bei 1-indizierter Zählweise
2 Anmerkung: Die Zieladresse kann für jedes Paket vor dem Sendevorgang frei vergeben werden.

4.4.1.2 Adressierung auf Schicht 3

Die Zuteilung von IPv4-Adressen wird über die dynamische Vergabe mittels DHCP [88] realisiert. Über einen mehrstufigen Nachrichtenaustausch wird einem Client von einem DHCP-Server eine IP-Adresse aus dem konfigurierten Subnetz zugewiesen. Außerdem erhält er den für die Event-Nachrichten erforderlichen Standardgateway, der den Versand an andere Subnetze sowie die Weiterleitung von Broadcasts behandelt.

Die DHCP-Funktion auf dem Server wird durch das Internet-Gateway bereitgestellt, das für jede Schnittstelle (`usb0`, `usb1`, ...) ein eigenes `/24`-Subnetz aufspannt. Das Präfix ist aus der Menge der privaten IP-Adressbereiche gewählt und lautet stets `10.0.`; im dritten Segment wird die Nummer des Subnetzes inkrementiert. Damit sind maximal 256 verschiedene Subnetze möglich; bei einer geschätzten Zahl von 200 Smart Devices in einem Smart Home reicht ein einziges Subnetz aus.

Es ergibt sich die folgende IPv4-Konfiguration für jedes Smart Device Netzwerksegment:

- `10.0.?.0`: Netzwerkadresse.
- `10.0.?.1`: IP-Adresse der Schnittstelle am Internet-Gateway, statisch konfiguriert.
- `10.0.?.[2-254]`: Verfügbare IP-Adressen für Smart Devices, per DHCP vergeben.
- `10.0.?.255`: Broadcast-Adresse des Netzwerksegments.

Weiterhin wird die Leasetime des DHCP-Servers in Sekunden (Option 51) ausgewertet. Sie beträgt im Beispiel 30 min; nach der halben verstrichenen Zeit wird ein erneuter Lease vom Server angefordert. Dies deckt sich mit der Empfehlung des Request for Comment (RFC) 2031 [88, Kapitel 4.4.5].

Der Code der Implementierung stammt aus Teilen der LUFA-Bibliothek [89]. Im ursprünglichen Code war eine statische ID („XID“) für die Kommunikation zwischen Client und Server vorgesehen, was bei mehreren Smart Devices, die gleichzeitig IP-Adressen anfragen, zu Fehlern führte. Der Code wurde korrigiert.

4.4.2 Simulation eines virtuell-zuverlässigen Mediums

Die Kommunikation zwischen Smart Devices untereinander und mit dem Gateway findet über die in Kapitel 4.3.1.4 beschriebenen RFM70-Funkmodule statt. Sie können maximal 32 B an Nutzdaten pro Paket über das in Abbildung 4.21 gezeigte Paketformat verschicken. Aufgrund dieser Beschränkung besteht keine Möglichkeit, einen gesamten Ethernet-Frame mit (unkomprimierten) IP-Nutzdaten und darauf aufsetzenden Protokollen in einem einzigen Paket zu übermitteln, wie in [90] und [91] spezifiziert.

Daher werden die Ethernet-Frames der jeweiligen Größe nach in bis zu 30 B lange Fragmente unterteilt (siehe Abbildung 4.22) und diese sequentiell übertragen. Ein vergleichbares Konzept findet – wenn auch mit anderer Motivation – beim Asychronous Transfer Mode (ATM) Anwendung, bei dem die Paketgröße allerdings 48 B bei 5 B Headerinformation beträgt.

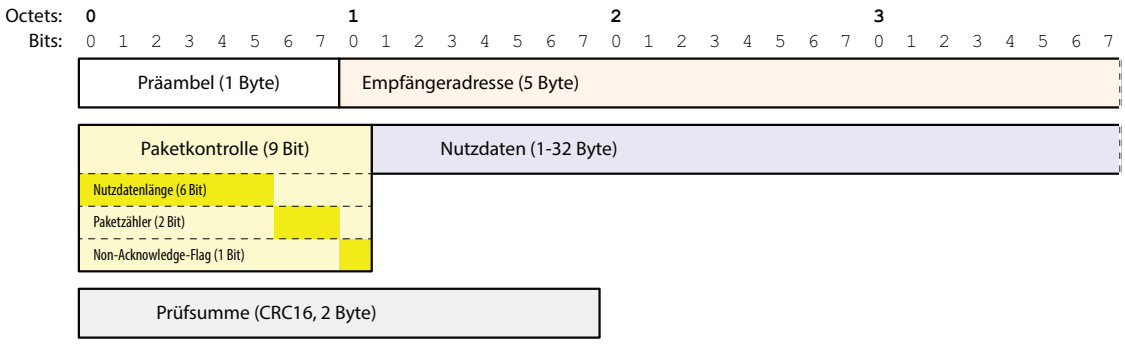


Abbildung 4.21: Aufbau eines RFM70-Paketes in der im Szenario verwendeten Konfiguration. Nach einer Präambel zur Synchronisation der Empfänger folgt aus der Ethernetadresse des Frames abgeleitete Zieladresse des oder der Smart Devices. Da die dynamische Paketlänge verwendet wird, geben 6 bit die Länge der Nutzdaten an; die beiden folgenden Features, die der automatischen Neu-Übertragung von Paketen dienen, werden nicht verwendet. An die bis zu 32 B Nutzdaten schließt sich eine CRC16 an, die automatisch von den Empfängern verifiziert wird.

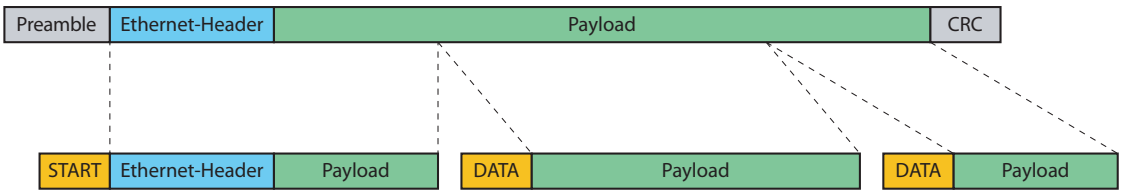


Abbildung 4.22: Aufteilen von Tagged-MAC-Frames als Grundlage für das RFM70-Funkprotokoll. Jedes Frame wird in entsprechend viele RFM-Pakete zu maximal 30 B geteilt, wobei das letzte Paket nur die erforderliche Anzahl an Bytes überträgt. Präambel und CRC über das Frame werden nicht übertragen, da sie von der Implementierung auf der MCU ignoriert werden und zudem das Medium als zuverlässig angesehen wird. [85, Kapitel 5.2]

4.4.3 Übertragungssicherheit des Funkkanals

Die erzeugten RFM-Pakete werden wegen der Unzuverlässigkeit des Funkmediums, insbesondere auf dem durch WLAN mitbenutzten 2,4 GHz-Band, gegen Verlust gesichert.

Das RFM70 bietet dafür den Mechanismus des „Auto-Acknowledge“, bei dem der Empfänger ein korrekt empfangenes Paket anhand der CRC identifiziert und dem Sender durch ein ACK-Paket bestätigt. Da keine Absenderadressen verwendet werden, akzeptiert der Sender ein beliebiges korrekt formatiertes Acknowledge-Paket. Gegen ein verlorenes ACK schützt ein Paketzähler mit vier Zuständen (2 bit).

Wenn ein ACK ausbleibt, wiederholt der Sender das im Speicher befindliche Paket bis zu 15 Mal, die Verzögerung zwischen den Paketen ist konstant und von 0,25 ms bis 4 ms konfigurierbar.

Mit der vorliegenden Hardware ergeben sich mehrere Schwierigkeiten:

- Fehlerhafte Pakete werden von der Funkhardware automatisch anhand einer unpassenden CRC-Quersumme erkannt und für die MCU unbemerkt verworfen.
- Der vorhandene Mechanismus zur automatischen Neuübertragung von RFM-Paketen („Auto-Acknowledge“) kann im Broadcast-Betrieb nicht verwendet werden. Ein einziges korrekt empfangenes und bestätigtes Paket eines beliebigen Devices würde den Versand des nächsten Teilpaketes auslösen. Eventuell für andere Devices verlorene Pakete werden nicht mehr wiederholt. Das gesamte Frame ist für diese Devices dann unbrauchbar.
- Die Protokolle höherer Schichten werden aufgrund ihrer Ethernet-Paketgröße aus mehreren RFM-Paketen bestehen. Fehlt eines davon, wird das gesamte Frame verworfen und eine Kommunikation kommt auf höheren Schichten nicht zustande.

Im Gegensatz zu anderen Techniken wie WLAN oder ZigBee, die bereits auf Schicht 2 eine Verwaltung der an einer Basisstation verbundenen Clients einführen, ist im vorliegenden Konzept die Menge der Clients nicht zu jedem Zeitpunkt bekannt. Smart Devices können jederzeit dem Netzwerk beitreten oder dieses verlassen.

Um den Verwaltungsaufwand dieses Konzepts gering zu halten und die Übertragungskapazität des Nachrichtentransports nicht zusätzlich mit JOIN- und LEAVE-Nachrichten zu belasten, wird daher ein Broadcast-Medium („Ad-Hoc-Netzwerk“) simuliert. Clients nehmen automatisch an der Kommunikation teil, sobald sie sich in Funkreichweite befinden und mit Strom versorgt sind.

Durch diese Annahme vereinfacht sich die Kommunikation zwischen den Teilnehmern, da kein Konzept zum Bootstrapping eines neuen oder zum Verwalten eines bestehenden Netzes implementiert werden muß, wie etwa bei 802.15.4-Netzen vorgesehen.

Auch ist das Konzept auf Kommunikationsebene unempfindlich gegen Ausfälle von Teilnehmern. Nicht mehr vorhandene Smart Devices stören die Kommunikation nicht; bei ausgefallenem Gateway können Smart Devices zumindest noch untereinander kommunizieren.

Obwohl bestimmte Merkmale der IT-Sicherheit (Verschlüsselung, Authentizität) implementiert werden können, ist dies kein Bestandteil der Arbeit; es würde für die Demonstration des Anwendungsprotokolls keinen Zusatznutzen bedeuten. Kapitel 4.4.6 beschäftigt sich allerdings mit Umsetzungsmöglichkeiten, die im Rahmen der vorhandenen Hardware realisiert werden können.

4.4.4 Reliable Broadcast-Kommunikation

Verschiedene Ansätze zur adressierten („Multicast“) und nicht-adressierten („Broadcast“) Mehrfachkommunikation sowie ihre Skalierung bei steigender Anzahl von Teilnehmern werden in [92], [93] und [94] evaluiert. Neben Single-Hop-Verbindungen werden in den Arbeiten insbesondere Verbindungen über mehrfache Hops in regulären IP-Netzen mit mehreren 1000 Teilnehmern simuliert.

Es wird die Ausbreitungsgeschwindigkeit, die Zuverlässigkeit und die Belastung für den Sender und die Empfänger sowie der Nachrichtentransportlösung untersucht. Insgesamt kommen die Autoren in [94] zu dem Schluß, daß mit steigender Anzahl von Teilnehmern und gleichbleibender Rechen- sowie Nachrichtenkapazität eine Non-Acknowledge (NACK) basierte Kommunikationslösung den geringsten Overhead erfordert.

4.4.4.1 Motivation

Daher wird ein zuverlässiges Unicast- und Broadcast-Funkprotokoll (explizit *nicht* Multicast!) in Software ohne Verwendung des Auto-Acknowledge-Features der RFM-Hardware implementiert, das den zuverlässigen Versand von Paketen an die anderen Teilnehmer übernimmt und im Broadcast-Modus Non-Acknowledge (NACK) basiert arbeitet: Smart Devices fordern fehlende Pakete über ein gesondertes Paket vom Sender an.

Mit der softwarebasierten Lösung wird so eine Abstrahierung von der Funk-Paketgröße und der verwendeten Funkhardware erreicht. Letztere kann auch gegen andere Module ausgetauscht werden.

4.4.4.2 Pakettypen

Für die zuverlässige Kommunikation sind die in Abbildung 4.23 gezeigten Pakettypen definiert. Der einfacheren Implementierung ist eine Codierung von Informationen in ganzen Byte anstatt nur einzelnen Bit geschuldet.

Die Aufgaben der einzelnen Typen umfassen:

- `START`. Das Startpaket markiert den Beginn eines Ethernet-Frames und enthält die Länge in Byte der Nutzdaten in allen RFM-Paketen insgesamt. Da ein Ethernet-Frame (ohne Jumbo-Frames) bis zu 1522 B lang sein kann, ist der Längenzähler als

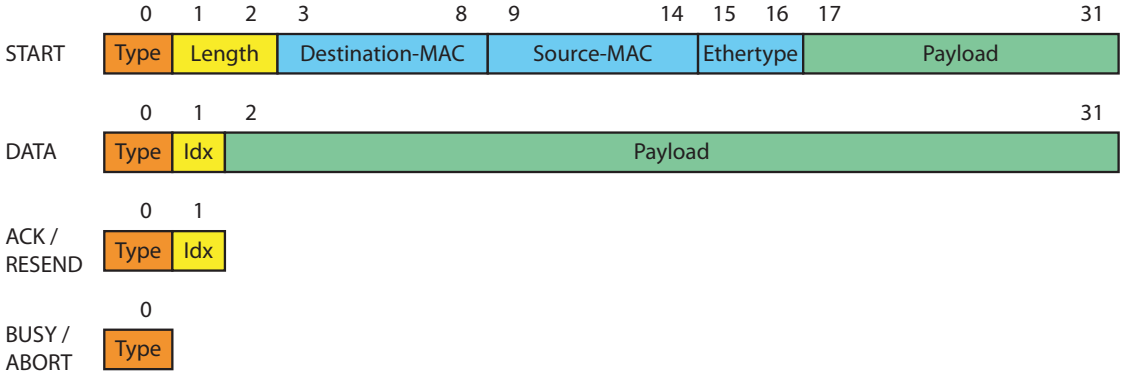


Abbildung 4.23: Definierte Pakettypen bei Uni- und Multicast-Kommunikation über das Funkprotokoll. Die erste Zeile gibt die nicht-linear skalierte Anzahl an Bytes für jedes Paket an. [85, Kapitel 5.2]

16 bit-Zähler ausgeführt. Das Startpaket erhält implizit durch seinen Typ den Index 0 und kann bis zu 15 B Nutzdaten enthalten.

- DATA. Auf ein Start-Paket folgen in der Regel ein oder mehrere Datenpakete, die die weiteren Daten des Ethernet-Frames, siehe auch Abbildung 4.22 beinhalten. Der Sender inkrementiert den Index ("Idx") für jedes neue Teilpaket um einen Wert. Ein Datenpaket kann bis zu 30 B Nutzdaten enthalten.

- ACK. Im Unicast-Modus dient dieser Pakettyp zur Bestätigung eines korrekt empfangenen Pakets und wird vom einzelnen Empfänger für jedes korrekt empfangene Paket mit einem bestimmten Index gesendet.

- RESEND. Bei Broadcast-Übertragungen dient das Paket der Anforderung einer erneuten Übertragung des RFM-Pakets mit dem bezeichneten Index („Non-Acknowledge"). Alle vom Sender bis dahin schon weiter übertragenen Pakete werden erneut gesendet.

- BUSY. Im Unicast-Modus dient das Paket dazu, einem zweiten Sender ein „Beschäftigt"-Signal zukommen zu lassen und ihn zum Abbruch seines Kommunikationsversuchs zu bewegen.

- ABORT. Mit dem letzten Pakettyp wird ein Kollisionsmanagement ermöglicht. Im Broadcast-Modus müssen bei Empfang des Pakets alle Sender ihre Übertragung abbrechen, da hier aufgrund fehlender Absenderadressen

Zur Kommunikation existieren die Betriebsmodi *Unicast* und *Broadcast*, die direkt mit dem Ethernet-Flag „Group-Address" korrelieren: Ist dieses gesetzt, so wird im Broadcast-Modus übertragen, ansonsten im Unicast-Modus. Beide Modi werden in RFM-Teilpaketen durch das unterschiedliche Typ-Byte am Anfang jedes Teilpakets unterschieden. Tabelle 4.3 listet die verwendeten Byte-Werte der Pakettypen und eine Beschreibung auf.

Bytewert	Bezeichnung	Modus	Beschreibung
0x10	START	unicast	Beginn der Übertragung eines Ethernet-Frames
0x11	START	broadcast	Beginn der Übertragung eines Ethernet-Frames
0x12	DATA	unicast	Fortsetzung eines Ethernet-Frames
0x13	DATA	broadcast	Fortsetzung eines Ethernet-Frames
0x14	ACK	unicast	Bestätigung eines Pakets
0x15	RESEND	broadcast	Aufforderung eines beliebigen Smart Devices, ein Paket zu wiederholen
0x16	BUSY	unicast	Empfänger kann das Frame nicht annehmen: Zurückstellen des Sendeversuchs
0x17	ABORT	broadcast	Auftreten eines Multicast-Sendekonflikts: Abbruch aller Sendeversuche jedes Smart Devices

Tabelle 4.3: Ethernet-Pakettypen des RFM70-Funkprotokolls.

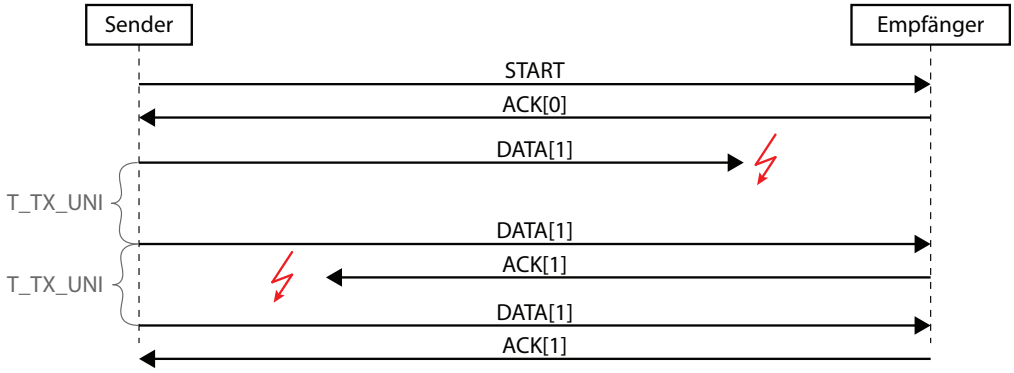


Abbildung 4.24: Neuübertragung von RFM-Paketen im Unicast-Modus bei fehlerhaftem Empfang entweder des zu sendenden Pakets oder der Bestätigung. Die Hardwareunterstützung der Funkmodule wird hierfür nicht verwendet.

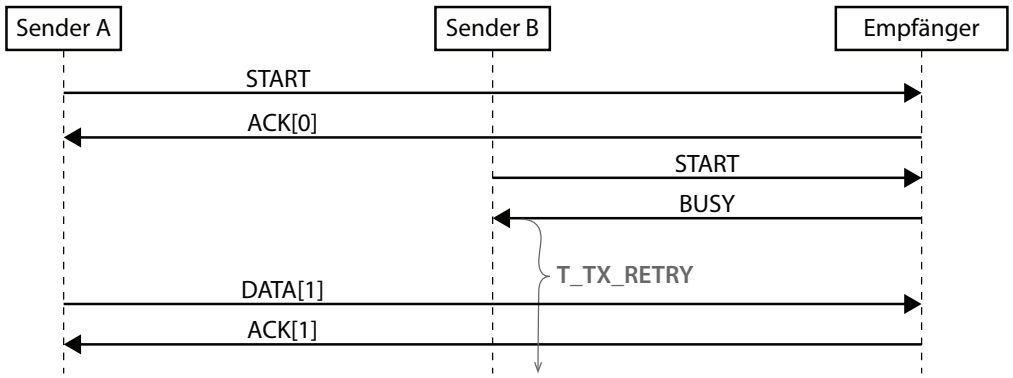
4.4.4.3 Unicast-Kommunikation

Bei Unicast werden Pakete nur an das adressierte Gerät gesendet und von diesem durch software-basierte ACK-Nachrichten bestätigt, siehe Abbildung 4.24. Die Retransmit/Auto-ACK-Funktion der RFM70-Module wird nicht verwendet, um die Funkhardware austauschbar zu halten. Außerdem ist sie durch eine Besonderheit des RFM-Hardwaredesigns bei mehreren bidirektional kommunizierenden Geräten unbrauchbar: Da keine Absenderadressen existieren, müssen beide Geräte die selbe Empfangsadresse einstellen; Konflikte könnten nicht mehr aufgelöst werden.

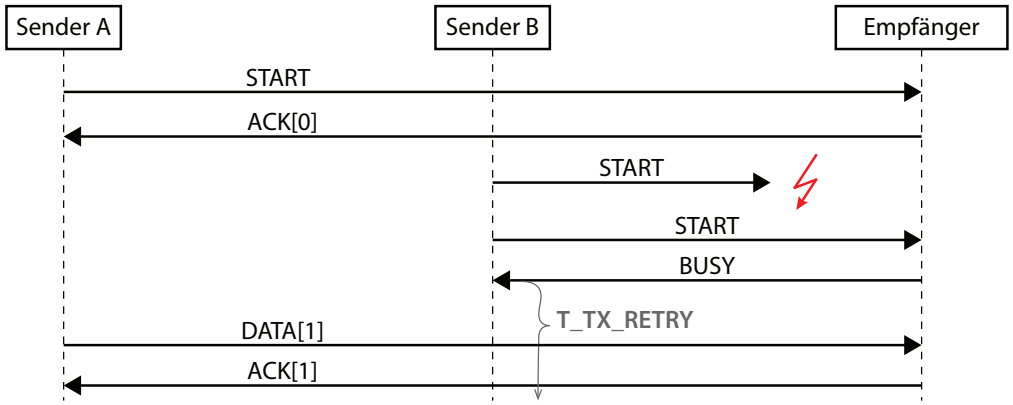
Der Sender benötigt daher einen Neuübertragungszähler und einen Bestätigungstimer in Software, um den Kommunikationsfluß zu überwachen. Ist ein RFM-Teilpaket durch fehlendes Bestätigen (Paket oder Bestätigung verloren) nicht sicher gesendet worden, so wird die Übertragung alle 5 ms (`T_TX_UNI`) bis zu 25 mal wiederholt. Falls danach kein ACK empfangen wurde, wird die Übertragung abgebrochen und das gesamte Frame als verloren gewertet.

Die verwendeten Funkmodule bieten keine (zuverlässige) Trägerwellenerkennung und können daher eine Belegung des Mediums nicht feststellen. Neben dem Verlust von RFM-Paketen, der durch den beschriebenen Bestätigungsmechanismus so weit wie möglich kompensiert wird, können allerdings Kollisionen bei Übertragung von Teilpaketen auftreten.

Im Unicast-Modus ist das der Fall, wenn ein weiterer Sender während der Kommunikation versucht, eine Übertragung mittels `START`-Paket zu initiieren (siehe Abbildung 4.25a). Der Fall kann auftreten, da nicht an der Kommunikation beteiligte Geräte aufgrund verschiedener RFM-Empfangsadressen keine Kommunikation feststellen können. Der Empfänger muß deswegen dem zweiten Sender ein Abbruchsignal übermitteln. Es wird per Unicast an die im ursprünglichen Start-Paket enthaltene Absenderadresse übertragen. Der zweite Sender stellt daraufhin seine Übertragung für eine zufällig bestimmte Zeitspanne zwischen 25 ms und 50 ms (`T_TX_RETRY`) zurück.



(a) Der Sendeversuch erfolgt während erfolgreicher Übertragung.



(b) Zweites START- oder BUSY-Paket gehen verloren.

Abbildung 4.25: Unterschiedliche Fälle von Kollisionen im Unicast-Modus des Funkprotokolls bei zwei Sendern. [85, Kapitel 5.4]

Würde das zweite START- oder das BUSY-Paket verlorengehen, würde der zweite Sender sein START-Paket erneut übermitteln und abermals ein BUSY erhalten, wie in Abbildung 4.25b dargestellt.

4.4.4.4 Broadcast-Kommunikation

Bei Broadcast-Kommunikation werden, wie in Abbildung 4.26 dargestellt, die RFM-Teilpakete an alle Empfänger gleichzeitig übermittelt. Auf RFM-Ebene wird dafür die Broadcast-Adresse FF:FF:FF:FF:FF verwendet, woran auf Empfängerseite auch zwischen Unicast-Kommunikation unterschieden wird. Der Sender wartet die Zeit T_TX_MULTI (5 ms) nach jedem Paket auf negative Rückmeldungen (NACKs). Sofern diese ausbleiben, gilt eine Übertragung des Teilpakets zunächst als erfolgreich.

Da im Broadcast-Modus stets die angegebene Zeit auf NACKs gewartet wird und keine adaptive Verkürzung der Zeit stattfindet, findet eine Übertragung zum einzelnen Empfänger in der Regel mit geringerem Datendurchsatz statt als bei einem Unicast-Vorgang.

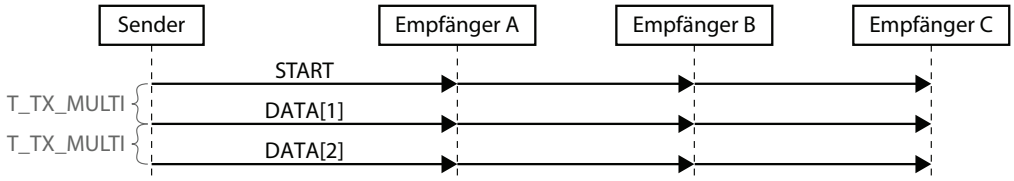


Abbildung 4.26: Darstellung der Übertragung von Paketen zu mehreren Empfängern ohne Fehler. Es werden keine ACK-Nachrichten generiert, da die Menge der Empfänger nicht bekannt ist. Es wird stets eine konstante Zeit `T_TX_MULTI` auf NACK-Pakete gewartet. [85, Kapitel 5.4]

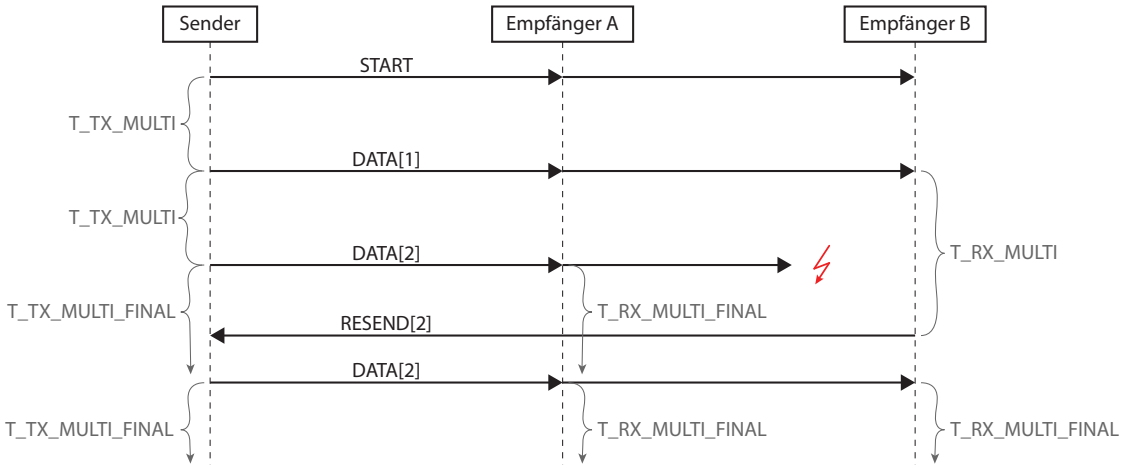


Abbildung 4.27: Neuübertragung eines Pakets im Broadcast-Modus. Ein `RESEND`-Befehl veranlaßt die erneute Übermittlung des Teilpakets mit dem darin bezeichneten Index. Ein Empfänger stellt den Fall anhand des abgelaufenen Timeouts `T_RX_MULTI` fest, innerhalb dessen er schon ein Teilpaket hätte empfangen müssen. [85, Kapitel 5.4]

Der Verlust eines Teilpakets kann, wie in Abbildung 4.27 gezeigt, an einem Empfänger nicht sofort erkannt werden, da die Funkhardware fehlerhafte Pakete ohne Benachrichtigung der MCU verwirft. Daher wird der Timeout `T_RX_MULTI` für die Erkennung eines fehlenden Teilpakets an der MCU verwendet. Mit Auslösen wird das fehlende Paket erneut angefordert.

Weicht der empfangene Index vom erwarteten nächsten Wert aufgrund mehrerer konsekutiver Paketverluste wie in Abbildung 4.28 ab, so fordert der Empfänger die Übertragung des ersten verlorenen RFM-Pakets mit einem `RESEND`-Paket an. Der Sender wiederholt daraufhin die Übertragung ab dem verlorenen Frame. Empfänger, die die wiederholten RFM-Pakete bereits korrekt erhalten hatten, ignorieren die Übertragung.

Das letzte RFM-Teilpaket eines Ethernet-Frames wird gesondert behandelt. Ein Empfänger kann aufgrund fehlender weiterer Pakete den Verlust des abschließenden Teilpaketes nicht mehr über den Index erkennen. Stattdessen wird ein Timer (`T_RX_MULTI`, 12 ms) für den Empfang des letzten Teilpakets verwendet, dessen Ablauf eine Neuanforderung dieses letzten Teilpakets auslöst. Der Sender wartet am Ende eines Broadcast-Frames eine

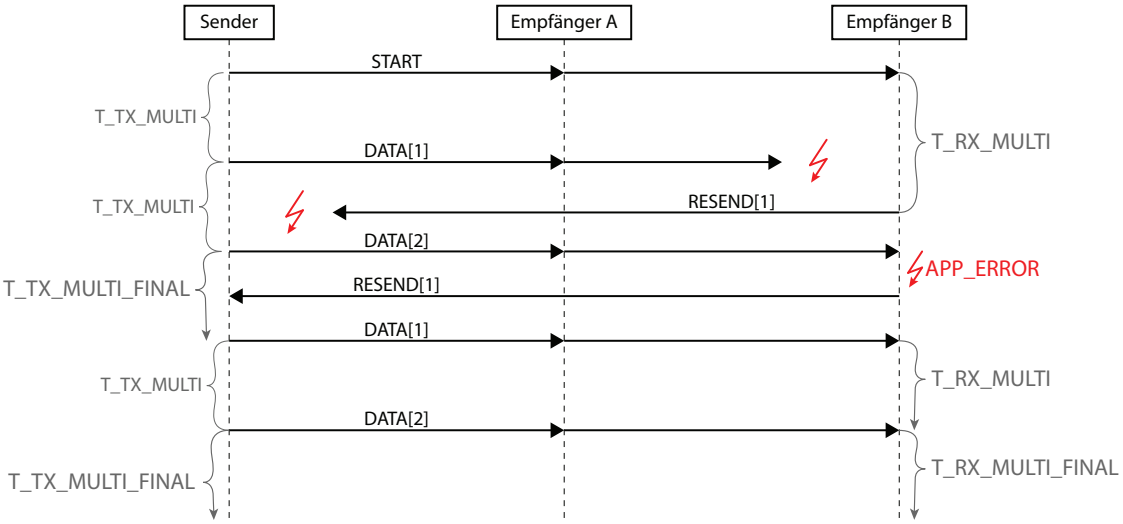


Abbildung 4.28: Neuübertragung mehrerer Pakete im Broadcast-Modus. Unter Umständen werden mehrere konsekutive Pakete verloren und der Sender nimmt irrtümlich ein erfolgreiches Versenden des Pakets an. Der Empfänger stellt den Fall anhand eines abweichenden Indexes fest und veranlaßt die erneute Übermittlung aller Teilpakete mit einem `RESEND`-Paket ab dem darin bezeichneten Index. [85, Kapitel 5.4]

längere Zeitspanne (`T_TX_MULTI_FINAL`, 30 ms), um eventuelle NACKs der letzten Teilpakete bearbeiten zu können. Während dieser Zeit dürfen keine neuen Übertragungen gestartet werden, da Empfänger sonst die Teilpakete nicht mehr einem einzigen Vorgang zuordnen könnten.

Da bei Broadcast-Übertragungen alle Geräte beteiligt sind, bestehen nur zwei Kollisionsszenarien. Zum einen können mehrere Smart Devices ein Teilpaket verloren haben, was mehrere NACK-Anforderungen zur Folge hat. Dem Sender reicht allerdings ein einziges NACK-Paket aus, um die Übertragung zu wiederholen, daher ist dieses Szenario abgedeckt.

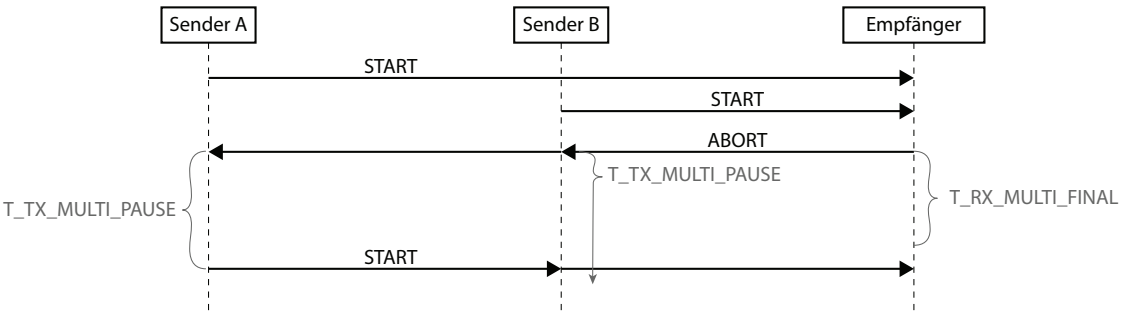


Abbildung 4.29: Darstellung der Kollision bei Broadcast-Übertragung. Zwei Geräte versuchen einen nahezu gleichzeitigen Start der Übertragung. Ein `ABORT`-Paket veranlaßt beide Sender zum Abbruch und erneuten Versuch nach einer zufälligen Backoff-Zeit (`T_TX_MULTI_PAUSE`). [85, Kapitel 5.4]

Zum zweiten könnten, wie auch in Abbildung 4.29 aufgezeigt, zwei Sender gleichzeitig eine Broadcast-Kommunikation initiieren. In diesem Fall werden beide Sendeversuche abgebrochen und jeweils nach einer zufälligen Backoff-Zeit erneut gestartet.

4.4.5 Details der Implementierung

Zur Behandlung von Ethernet-Frames, der IP-Pakete und den aufgesetzten Protokollen UDP und TCP wird der uIP-Stack des Betriebssystems Contiki [48] verwendet. Die verwendeten xmegaAVR MCUs besitzen ausreichend RAM-Speicherkapazität, um einen statischen Puffer für ein gesamtes Ethernet-Frame mit bis zu 1522 Byte bereitzustellen.

Ein Smart Device kann daher entweder senden oder empfangen. Der USB-Dongle dagegen besitzt zwei getrennte unabhängige Puffer zum Senden und Empfangen, somit kann ein Puffer von der USB-Logik bedient werden, während der andere vom Funkmodul bearbeitet wird.

4.4.5.1 Versenden eines Ethernet-Frames

Um ein Ethernet-Frame zu versenden, wird dieses zunächst durch den uIP-Stack in einem Puffer im Speicher der MCU konstruiert. Zustandsautomaten ermöglichen dabei, auch lange Antworten des Smart Devices, welche mehrere konsekutive Frames erfordern, zuverlässig zu versenden.

Nach Erstellen eines Frames durch das uIP-Modul stehen drei Funktionen zur Verfügung, die das Versenden steuern:

1. `int_fast8_t rfm_eth_send_frame()`. Mit dieser Funktion fordert die Firmware das Versenden an und übergibt die Länge des Ethernet-Frames. Kann wegen belegtem Medium oder stattfindender Übertragung von anderen Geräten das Versenden nicht gestartet werden, wird ein Fehler zurückgegeben; die Anwendung muß das Versenden nach einer Backoff-Zeit erneut versuchen.

2. `uint8_t * rfm_eth_cb_request_data()`. Diese Callback-Funktion ermittelt und retourniert einen Zeiger auf den Teil des Ethernet-Frames, der durch Index und Länge in ihren Parametern angegeben ist. Der RFM-Stack liest daraufhin die hier schon angegebene Menge an Bytes von der angegebenen Adresse und versendet sie.

3. `void rfm_eth_cb_send_frame_done()`. Nach Beendigung des Sendevorgangs wird die Anwendung über Erfolg oder Mißerfolg benachrichtigt und kann über Verwerfern oder einen erneuten Sendeversuch des Frames im Fehlerfalle entscheiden.

4.4.5.2 Empfang eines Ethernet-Frames

Der Empfang wird im Code über drei Callback-Funktionen implementiert, mit welchen das Ethernet-Frame Stück für Stück der Anwendung zur Verfügung gestellt wird. Damit es vom uIP-Stack verarbeitet werden kann, schreibt die Anwendung die Teilpakete zunächst in einen Puffer und rekonstruiert so das vollständige Frame.

1. `bool rfm_eth_cb_received_frame()`. Die Funktion wird zu Beginn des Empfangs mit der Gesamtlänge des nachfolgenden Frames aufgerufen. Implementierungen, die `malloc()` verwenden, können einen entsprechend großen Speicherbereich reservieren. Die Funktion muß der RFM-Empfangslogik über den Rückgabewert einen möglichen Empfang des Frames signalisieren. Damit kann ein BUSY-Paket generiert werden, falls gerade eine andere Übertragung läuft.

2. `void rfm_eth_cb_received_data()` Die Funktion wird mit einem Zeiger auf die Adresse zu den Nutzdaten des Teilpakets, dem null-indizierten Index innerhalb des Ethernetframes und der Länge der des Teilpakets aufgerufen. Die Anwendung kopiert die bezeichneten Daten mittels `memcpy()` in den richtigen Teil des Ethernet-Framebuffers.

3. `void rfm_eth_cb_receive_complete()` Über diese Funktion wird die Anwendung abschließend über den vollständigen Empfang des Ethernet-Frames benachrichtigt und erhält einen Statusbericht über Erfolg oder Mißerfolg. Die Verarbeitung durch den uIP-Stack wird hierin ausgelöst, sofern das Frame erfolgreich empfangen wurde.

4.4.5.3 Maximale Anzahl der Versuche einer Neuübertragung

Sowohl Sender wie auch Empfänger besitzen Zähler, die die Anzahl der Neuübertragungen überwachen. Ihre Maximalwerte sind wie folgt definiert:

- `RX_RETRY_NUM`. Legt die maximale Anzahl fest, die ein RFM-Teilpaket mit einer bestimmten Sequenznummer im Unicast-Modus durch ACK bestätigt und im Multicast-Modus mit einer NACK-Meldung erneut angefordert wird. Bei Überlauf wird das aktuelle Frame verworfen. Der Wert beträgt 25.

- `TX_RETRY_NUM`. Legt die maximale Anzahl fest, die ein RFM-Teilpaket vom Sender übertragen wird. Bei Überlauf bricht der Sender in beiden Modi das Versenden des Frames ab und benachrichtigt die Anwendung. Der Wert beträgt ebenfalls 25.

Ist die Übertragung nicht erfolgreich, definiert das Transportprotokoll das weitere Verhalten: Im Falle von TCP kümmert sich die Transportschicht um ein erneutes Versenden des Pakets und damit des Ethernet-Frames. Bei UDP ist das Paket verloren; die Host-Anwendung (auf MCU oder IP-Client im lokalen Netz) wird darüber nicht benachrichtigt.

4.4.5.4 Definierte Timer

Zusammenfassend existieren die folgenden Timer zur Steuerung der Kommunikation. Die Werte wurden in Versuchsreihen empirisch ermittelt und sind mit großzügigen Margen beaufschlagt.

- `TIMEOUT_TX_UNICAST`. Nach dieser Zeit wird bei ausbleibendem Unicast-ACK das aktuelle Teilpaket erneut übertragen und der Retransmit-Counter inkrementiert. Der Timeout ist auf 5 ms gesetzt.

- `TIMEOUT_RX_UNICAST`. Dieser Timeout dient zur Feststellung einer final fehlgeschlagenen Unicast-Übertragung, wenn `TX_RETRY_MAX * TIMEOUT_TX_UNICAST` lang keine Pakete empfangen wurden. Der Timeout ist auf daher 250 ms gesetzt.

- `TIMEOUT_TX_RETRY_MIN, TIMEOUT_TX_RETRY_MAX`. Diese beiden Werte beschreiben das Intervall, in welchem ein Smart Device seine zufällige Backoff-Zeit wählt, die vor einem erneuten Unicast-Sendeversuch verstreichen muß. Das Intervall liegt zwischen 25 ms und 50 ms.

- `TIMEOUT_TX_MULTICAST`. Im Multicast-Modus wird stets eine feste Zeit lang auf NACK-Pakete gewartet. Die Länge dieses Intervalls beträgt 5 ms.

- `TIMEOUT_RX_MULTICAST` Im Multicast-Modus stellt ein Empfänger mit Auslösen dieses Timeouts ein fehlendes Datenpaket fest und fordert es mittels `RESEND` erneut an. Der Timeout ist geringfügig größer als das Sendeintervall und liegt bei 12 ms. Dadurch liegt es genau im Senderaster zwischen zwei Paketübertragungen durch den Sender.

- `TIMEOUT_TX_MULTICAST_FINAL` Wie beschrieben wartet der Sender beim letzten Paket eine größere Zeitspanne auf eintreffende NACK-Pakete. Der Wert dieses Timeouts muß deutlich größer sein als `TIMEOUT_RX_MULTICAST` und beträgt 30 ms.

- `TIMEOUT_RX_MULTICAST_FINAL` Dieser Zähler bezeichnet das Ende einer Broadcast-Übertragung und ist auf 35 ms gesetzt. Solange er noch nicht abgelaufen ist, werden alle eingehenden Pakete als Neu-Übertragungen eines bisherigen Frames gewertet. Erst mit Ablauf beginnt implizit ein neues Frame. Jedes innerhalb dieser Zeit eintreffende RFM-Paket setzt den Zähler zurück.

- `TIMEOUT_TX_MULTICAST_PAUSE_MIN, TIMEOUT_TX_MULTICAST_PAUSE_MAX` Ist ein Konflikt beim Senden von Multicast-Paketen aufgetreten, oder ist die Übertragung beendet, so wählen die Sender eine zufällige Periode zwischen diesen Zeitpunkten, die sie mindestens abwarten, bevor eine erneute Sendung gestartet wird. Da Intervall liegt zwischen 50 ms und 75 ms und ist in jedem Fall länger als `TIMEOUT_RX_MULTICAST_FINAL`.

4.4.6 IT-Sicherheit

Das implementierte Protokoll ist gegen Angriffe wie Sniffing, Spoofing oder Denial-of-Service (DoS) [95, Kapitel 3] auf den OSI-Schichten nicht geschützt. Mit Standardhardware kann der Datenverkehr mit geringem Aufwand (Hardwarekosten ~30 €) sowohl abgehört wie auch manipuliert werden.

Da das implementierte Szenario vorrangig der Demonstration des Anwendungsprotokolls sowie der Validierung des Ethernet-Konzepts rein in Software dient, wurde der Mangel an Sicherheit wissentlich in Kauf genommen. Er hat keine weitere Auswirkung auf das Anwendungsprotokoll.

Abschließend für dieses Kapitel wird daher eine – nicht implementierte – Software-Erweiterung der Smart Devices hinsichtlich einer symmetrischen Verschlüsselung [95, Kapitel 7.5] beschrieben, die auf der verwendeten Hardware eingesetzt werden kann und zumindest die Nutzdaten der RFM-Pakete sichert. Angriffe auf den physikalischen Nachrichtentransport, z.B. durch Störung des Funkmediums, bleiben weiterhin möglich.

4.4.6.1 Hardware-Unterstützung

Die verwendete MCU-Familie der xmegaAVR besitzt zwei Hardwaremodule, die die Verschlüsselungsoperationen Data Encryption Standard (DES) mit 64 bit Schlüssellänge und Advanced Encryption Standard (AES) mit 128 bit Schlüssellänge durch eigene Operationen im Rechenwerk unterstützen. Die stärkere Triple-DES (3DES) Verschlüsselungsmethode wird durch mehrmaliges Ausführen der DES-Sequenz mit unterschiedlichen Teilen des Schlüssels realisiert.

Die Ver- bzw. Entschlüsselung eines Datenblocks benötigt bei DES 16 Taktzyklen, wobei die MCU blockiert ist. Die AES-Einheit ist nicht-blockierend ausgelegt und benötigt für eine Operation 375 Taktzyklen [70]. Dabei kann die MCU entweder im Polling-Betrieb aktiv auf die Beendigung der Operation warten oder sich über einen Interrupt mit programmierbarer Priorität benachrichtigen lassen.

Der Hersteller Atmel bietet in den Application Notes AVR1317 [96] und AVR1318 [97] ausführliche Beschreibungen der Hardware und modulare Treiber in *C*.

4.4.6.2 Verschlüsselung

Die von den Funkmodulen zu versendenden Pakete (Ethernet-Frames mit Nutzdaten und Reliable-Broadcast-Paketinformationen) werden, bevor sie per SPI in das Funkmodul zum Versand geladen werden, durch das AES-Modul verschlüsselt. Aufgrund der Paketgröße von 32 B sind dafür zwei konsekutive Operationen sowie Speicherbewegungen der beiden Teile des jeweils zum Versand bereiten Teilpakets der Ethernet-Frames erforderlich. Außerdem muß auch das letzte Paket auf 32 B expandiert werden, wodurch sich ein Overhead (mehr übertragene Daten als notwendig) ergeben kann.

Alle Geräte am Segment arbeiten die Ver- bzw. Entschlüsselung in vergleichbar schneller Zeit ab, damit Timeouts für die Neuanforderung von Paketen nicht ablaufen und eine Übertragung unmöglich wird.

Um Replay-Angriffe abzuwehren, muß zudem an geeigneter Stelle ein Rolling-Code Mechanismus eingeführt werden. Insbesondere eine Synchronisation zwischen allen Geräten stellt bei einer unbekannten Zahl von Teilnehmern eine Herausforderung dar, auf die an dieser Stelle nicht näher eingegangen wird.

4.4.6.3 Schlüsselaustausch

Da grundsätzlich die Eigenschaft der verwaltungslosen Broadcastkommunikation erhalten bleiben soll, kommt bei symmetrischer Verschlüsselung mit mehreren Teilnehmern ein gemeinsam genutzter Schlüssel in Betracht, der jedem Teilnehmer vorab bekannt ist. Zu beachten ist, daß Teilnehmer zu jedem Zeitpunkt das Netz betreten können und dennoch den selben Schlüssel benötigen.

Bei Geräten mit Grafik-LCD etwa kann dazu eine Passphrase in einem speziellen Menüpunkt eingetragen werden, aus der der Schlüssel abgeleitet wird („key derivation“). Der Austausch findet dabei über den als sicher erachteten Kanal der Benutzereingabe statt.

Bei Geräten ohne Benutzerschnittstelle wird für die gleiche Funktion ein Datenpunkt des Anwendungsprotokolls definiert. Das Gerät wird zunächst unverschlüsselt angesprochen und per `POST` mit der Passphrase konfiguriert. Ab dann verwendet es die Daten, um den Schlüssel abzuleiten und die Pakete zu ver- und entschlüsseln. In diesem Falle ist ein sicherer Schlüsselaustausch allerdings nicht möglich; der initiale Kommunikationsvorgang dürfte nicht mitgeschnitten werden.

Eine bessere Lösung involviert ein zweistufiges Verfahren, bei dem zunächst über Diffie-Hellmann ein Sitzungsschlüssel zwischen zwei oder mehreren „anlernbereiten“ Geräten ausgetauscht wird und dieser temporär für einen symmetrisch verschlüsselten Kanal verwendet wird. Über diesen wird daraufhin der richtige Schlüssel oder die Passphrase übermittelt.

Schlüsselherleitung und -austausch stellen insbesondere auf eingebetteter Hardware komplexe Vorgänge dar. In der Arbeit ist insofern keine weitere Spezifikation für Algorithmen oder Prozesse gegeben. In jedem Falle muß beachtet werden, daß statische Sitzungsschlüssel stets ein Sicherheitsrisiko bedeuten.

Insgesamt wird der kryptographische Layer als Schicht unterhalb der Ethernet-Frames und unterhalb des definierten Reliable-Broadcast-Protokolls angesehen; das Anwendungsprotokoll und die anderen OSI-Schichten arbeiten unabhängig davon.

4.5 Beschreibung der Firmware

Eine der aufgestellten Anforderungen war eine möglichst geringe Time-to-Market sowie eine möglichst geringe Einarbeitungszeit für Entwickler beim Design von Smart Devices. Für die Hardwareseite wurde im Kapitel 4.3 bereits ein mögliches Konzept zur Erfüllung dieser Anforderung vorgestellt. Es wird nun um Erläuterungen zur in *C* erstellten Firmware, insbesondere zur Parametrisierung für Smart Devices, ergänzt. Das Konzept wurde in den Arbeiten [98] und [99] erfolgreich umgesetzt.

Für eingebettete Plattformen (AVR-, MSP430- und ARM-MCUs) existieren mehrere Betriebssysteme (z.B. TinyOS [100], Contiki [48]). Sie bieten dem Entwickler eine Reihe von Bibliotheken zur Wahrnehmung von Aufgaben eines Betriebssystems [101] der eingebetteten Programmierung (Tasks, Hardwareabstraktion/Treiber, Netzwerk-Stacks, etc.) und übersichtlichen Kapselung von Code in Quelltexten.

In der vorliegenden Implementierung wurde aus folgenden Gründen bewußt auf die Verwendung von Betriebssystemen verzichtet:

- Entwickler von Smart Devices sollen nicht auf die Verwendung eines einzigen Betriebssystems festgelegt werden. Eine Abschätzung, welches der beiden genannten, oder ob überhaupt eines davon, für ein bestimmtes Smart Device am besten geeignet ist, könnte im Rahmen der Arbeit nicht gegeben werden.
- Die Hardwareabstraktion muß für externe Peripherie in jedem Fall durchgeführt werden; zumindest für die in dieser Arbeit verwendeten Module existieren keine vordefinierten stabilen Bibliotheken für die genannten Betriebssysteme. Stattdessen müßte bei Erstellung der Abstraktionsschicht größerer Aufwand betrieben werden, um die Konventionen des jeweiligen Betriebssystems einzuhalten.
- Die Verwendung eines Betriebssystem bietet für die vorliegenden eingebetteten Systeme keine Vorteile hinsichtlich der Modularisierung und Kapselung von Code. Für Entwickler erscheint stattdessen eine Kapselung von Code in eigenen Makros besser verständlich.

Bei Verwendung von Standardhardware wie Sensorknoten, auf welche die jeweiligen eingebetteten Betriebssysteme zugeschnitten sind, wird der Einsatz eines solchen durchaus für sinnvoll erachtet. Bei Eigenentwicklung der Hardware, so auch in der vorliegenden Arbeit, ist die Anpassung des Betriebssystems daran allerdings aufwendiger als ein für die Plattform spezifischer *C*-Code. Eine pauschale Aussage, „ab wann“ sich der Einsatz eines Betriebssystems lohnt, kann an dieser Stelle nicht getroffen werden.

4.5.1 Programmablauf

Bis auf wenige Callback-Funktionen bei Interrupts der Hardware wird ein Smart Device daher an die Plattform angepaßt single-threaded programmiert. Nicht verwendete CPU-Zyklen werden zugunsten einer einfacheren Implementierung des Demonstrators mit

Leerlaufinstruktionen („busy waiting“ oder `NOP`) verbracht.

Die Abarbeitung von Aufgaben („Tasks“) findet in der Hauptschleife statt; dem Entwickler ist eine Unterteilung des Codes in Tasks zur Modularisierung auferlegt. Da es sich bei der verwendeten AVR-Architektur um Einzelkern-MCUs handelt, werden die Tasks zu keiner Zeit parallel ausgeführt.

Im Rahmen dieses Kapitels werden die drei Punkte der Parametrisierung eines Gerätes, der Definition von Callback-Funktionen für Datenpunkte und die Parametrisierung der Menüstruktur bei Devices mit Grafik-LCD besprochen.

4.5.2 Parametrisierung

Um die statischen Elemente des virtuellen XML-Dokumentes zu definieren, wurden Präprozessor-Makros [102, Kapitel 3] innerhalb der Entwicklungsumgebung definiert. Sie sorgen für einen einheitlichen und lesbaren Code, der in allen Smart Devices den gleichen Aufbau besitzt.

In der jeweiligen `main.c` wird die geräte-spezifische Konfiguration (Hauptschleife, XML-Parameter) angegeben. Sie bindet außerdem per `#include` eine Reihe von plattformabhängigen und auch -unabhängigen Headerdateien ein, die zum einen weitere Konfigurationen und Treiber für die jeweilige gerätespezifische Peripherie beinhalten. Zum anderen werden Headerdateien eingebunden, die stets gleiche Programmteile, wie Netzwerkstack und Funkmodultreiber, implementieren.

4.5.2.1 Globale Definitionen von Callback-Makros

Die Makros zur Definition von XML-Elementen basieren auf den folgenden *C*-Definitionen. Im folgenden Listing 4.9 wird generell die Struktur für lesende und schreibende Callback-Funktionen festgelegt.

Zur Erkennung des betroffenen Datenpunktes innerhalb der Callback-Funktion werden Zeiger auf die jeweiligen Instanzen der (erst in einem späteren Kapitel erklärten) jeweiligen Service- und Datenpunktknoten übermittelt. Damit muß für gleichartige Datenpunkte jeweils nur eine einzige Callback-Funktion für lesende und schreibende Werte implementiert werden.

Schreibende Funktionen bekommen den Wert, der in einen Maschinenzustand zu überführen ist, mitgeliefert; lesende Funktionen erhalten einen Pointer, welcher auf das jeweilige Ergebnis zu setzen ist. Die Unterscheidung der drei definierten Datentypen des Anwendungsprotokolls ist in den unterschiedlichen Datentypen der Funktionen und ihrer Funktionsnamen bereits berücksichtigt. Zu beachten ist die doppelte Dereferenzierung bei Zeichenketten.

```
struct _DATA_NODE;
struct _SENSOR_NODE;

// Callback definitions
typedef void(* SENSOR_WRITE_INT_CB)   (int32_t   iValue, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
typedef void(* SENSOR_READ_INT_CB)    (int32_t *piValue, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
typedef void(* SENSOR_WRITE_FLOAT_CB) (float     fValue, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
typedef void(* SENSOR_READ_FLOAT_CB)  (float   *pfValue, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
typedef void(* SENSOR_WRITE_STR_CB)   (const char  *str, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
typedef void(* SENSOR_READ_STR_CB)    (char      **pStr, const struct
    _SERVICE_NODE *pServiceNode, const struct _DATA_NODE *pDataNode);
```

Listing 4.9: Typ-Definition von Callback-Funktionen.

Für jeden Datentyp werden wie in Listing 4.10 gezeigt sechs Instanzen über `#define`-Makros definiert, die jeweils direkt eine Speicherzelle, eine Funktion ohne Argumente oder eine Funktion mit mehreren Argumenten erwarten. Über Funktionspointer werden so Aliase für gleiche Callback-Funktionen, allerdings mit unterschiedlichen Maschinenzustandsargumenten definiert.

```
// Easier pre-written definitions
#define WEBSERVER_CB_NODE_PARAM_ const SERVICE_NODE *sn, const DATA_NODE *
    pn

#define WEBSERVER_CB_GET_INT_VAL(   name, var)          static void name(
    int32_t *val, WEBSERVER_CB_NODE_PARAM_) {*val = var;}
#define WEBSERVER_CB_GET_INT_FUN(   name, fun)          static void name(
    int32_t *val, WEBSERVER_CB_NODE_PARAM_) {*val = fun();}
#define WEBSERVER_CB_GET_INT_FUN_VA(name, fun, ...)     static void name(
    int32_t *val, WEBSERVER_CB_NODE_PARAM_) {*val = fun(__VA_ARGS__);}
#define WEBSERVER_CB_SET_INT_VAL(   name, var)          static void name(
    int32_t  val, WEBSERVER_CB_NODE_PARAM_) {var = val;}
#define WEBSERVER_CB_SET_INT_FUN(   name, fun)          static void name(
    int32_t  val, WEBSERVER_CB_NODE_PARAM_) {fun(val);}
#define WEBSERVER_CB_SET_INT_FUN_VA(name, fun, ...)     static void name(
    int32_t  val, WEBSERVER_CB_NODE_PARAM_) {fun(__VA_ARGS__, val);}

#define WEBSERVER_CB_GET_FLOAT_VAL(   name, var)        static void name(
    float   *val, WEBSERVER_CB_NODE_PARAM_) {*val = var;}
#define WEBSERVER_CB_GET_FLOAT_FUN(   name, fun)        static void name(
    float   *val, WEBSERVER_CB_NODE_PARAM_) {*val = fun();}
#define WEBSERVER_CB_GET_FLOAT_FUN_VA(name, fun, ...)   static void name(
    float   *val, WEBSERVER_CB_NODE_PARAM_) {*val = fun(__VA_ARGS__);}
#define WEBSERVER_CB_SET_FLOAT_VAL(   name, var)        static void name(
    float    val, WEBSERVER_CB_NODE_PARAM_) {var = val;}
#define WEBSERVER_CB_SET_FLOAT_FUN(   name, fun)        static void name(
    float    val, WEBSERVER_CB_NODE_PARAM_) {fun(val);}
#define WEBSERVER_CB_SET_FLOAT_FUN_VA(name, fun, ...)   static void name(
    float    val, WEBSERVER_CB_NODE_PARAM_) {fun(__VA_ARGS__, val);}
```

```
15 #define WEBSERVER_CB_GET_STRING_VAL(   name, var)      static void name(
       char    **str, WEBSERVER_CB_NODE_PARAM_) {*str = var;}
16 #define WEBSERVER_CB_GET_STRING_FUN(   name, fun)      static void name(
       char    **str, WEBSERVER_CB_NODE_PARAM_) {*str = fun();}
17 #define WEBSERVER_CB_GET_STRING_FUN_VA(name, fun, ...) static void name(
       char    **str, WEBSERVER_CB_NODE_PARAM_) {*str = fun(__VA_ARGS__);}
18 #define WEBSERVER_CB_SET_STRING_VAL(   name, var)      static void name(
       char     *str, WEBSERVER_CB_NODE_PARAM_) {strcpy(var, str);}
19 #define WEBSERVER_CB_SET_STRING_FUN(   name, fun)      static void name(
       char     *str, WEBSERVER_CB_NODE_PARAM_) {fun(str);}
20 #define WEBSERVER_CB_SET_STRING_FUN_VA(name, fun, ...) static void name(
       char     *str, WEBSERVER_CB_NODE_PARAM_) {fun(__VA_ARGS__, str);}
```

Listing 4.10: Makrodefinitionen für Callback-Funktionen und -Aliase. Zu beachten ist bei schreibenden Zeichenkettenoperationen direkt auf einen Speicherbereich das manuelle Kopieren des Wertes mit `strcpy()`, da Gleichsetzungsoperationen nur für numerische Datentypen unterstützt werden. Beim Lesen wiederrum ist das nicht erforderlich; die aufrufende Funktion verwendet den selben Pointer.

4.5.2.2 Datenpunkte

Datenpunkte können über das Anwendungsprotokoll gelesen oder beschrieben werden. Zur Definition eines Datenpunktes existieren verschiedene statische Makros (siehe Listing 4.11), die entweder eine Angabe von Getter- und Setter-Callback-Funktionen erlauben oder eine (zur Laufzeit statische) Adresse einer Variablen und ihrer Dimension in Bytes erwarten. Dies vereinfacht den Zugriff auf reine Speicherstellen und reduziert den Overhead (Flash-Speicher, Rechenzyklen und RAM-Belegung) für einen Datenpunkt erheblich, wenn Werte nicht mehrfach behandelt oder transformiert werden müssen. Eine Prüfung der übergebenen Werte gegen Beschränkungen findet automatisiert in jedem Falle, unabhängig von einer definierten Call-Back-Funktion, statt.

Die Makros besitzen stets die gleiche Grundstruktur, abhängig vom vorgesehenen Ziel (Variable/Speicherzelle oder Funktion) erwarten sie aber eine unterschiedliche Anzahl und Typisierung der letzten Parameter. Mit Hilfe von `union`-Statements werden die Definitionen kompakt gehalten.

```
1  #define FEATURE_NONE      (0)
2  #define FEATURE_READABLE  (1 << 0)
3  #define FEATURE_WRITEABLE (1 << 1)

4  typedef enum {
5    NT_INTEGER, // Callback erwartet int32_t
6    NT_FLOAT,   // Callback erwartet float
7    NT_STRING,  // Callback erwartet string
8  } NODE_TYPE;

9  typedef struct _DATA_NODE {
10   const char      *name;     // Name des Knotens
11   const NODE_TYPE  nodetype; // Typ, siehe oben
```

```
12   const char       *unit;      // Zeichenkette für unit-Attribut
13   const uint8_t     feature;   // bit-weise OR Verknüpfung von FEATURE_*
14   union {
15     struct {
16       const int32_t                iMin;
17       const int32_t                iMax;
18       const SENSOR_READ_INT_CB     pReadIntCB;
19       const SENSOR_WRITE_INT_CB    pWriteIntCB;
20     } intNode;
21     struct {
22       const float                  fMin;
23       const float                  fMax;
24       const SENSOR_READ_FLOAT_CB   pReadFloatCB;
25       const SENSOR_WRITE_FLOAT_CB pWriteFloatCB;
26     } floatNode;
27     struct {
28       const size_t                 iLen;
29       const SENSOR_READ_STR_CB     pReadStrCB;
30       const SENSOR_WRITE_STR_CB    pWriteStrCB;
31     } strNode;
32   } node;
33   bool bChanged; // Dirty-Flag, das bei der Ausgabe ausgewertet wird.
34 } DATA_NODE;

35 // Instanziierung der union-Varianten für einfache Verwendung
36 #define INTEGER_NODE(name, unit, feature, iMin, iMax, readCB, writeCB) {
       name, NT_INTEGER, unit, feature, {.intNode = {iMin, iMax, readCB,
       writeCB} }, false}
37 #define FLOAT_NODE(name, unit, feature, fMin, fMax, readCB, writeCB)   {
       name, NT_FLOAT, unit, feature, {.floatNode = {fMin, fMax, readCB,
       writeCB} }, false}
38 #define STRING_NODE(name, unit, feature, length, readCB, writeCB)      {
       name, NT_STRING, unit, feature, {.strNode = {length, readCB, writeCB}
       }, false}
```

Listing 4.11: Makrodefinitionen der Datenpunkt-Ebene.

Durch die Definitionen über Makros ist noch keine Aussage über den verwendeten Speicherplatz getroffen. Je nach Anwendung können die späteren instanziierten Definitionen im ROM oder zusätzlich auch im RAM der MCU abgelegt werden.

Abschließend ist in Listing 4.12 ein vollständiges Beispiel zur Definition von Datenpunkten angegeben. Für die spätere Service-Definition wird eine Kollektion der einzelnen Datenpunkte benötigt, daher ist dies bereits im Beispiel berücksichtigt.

```
1 // Die Callback-Funktionen erhalten Referenzen auf den aufrufenden
      Datenpunkt und Service.
2 static void read_led_node_rgb(int32_t *val, const SENSOR_NODE *sn, const
      DATA_NODE *dn) {
3   // Unterscheidung des Datenpunktes innerhalb der Callback-Funktion.
        Damit muß nicht für jeden Datenpunkt eine eigene Funktion
        implementiert werden, wenn ansonsten die Funktionalität ähnlich ist.
4   if (dn->name == s_red) { /*...*/ }
5   else if (dn->name == s_green) { /*...*/ }
```

```
6   else if (dn->name == s_blue) { /*...*/ }
7 }
8 static void read_led_node_hsv(int32_t *val, const SENSOR_NODE *sn, const
      DATA_NODE *dn) { /*...*/ }
9 static void write_led_node_rgb(int32_t val, const SENSOR_NODE *sn, const
      DATA_NODE *dn) { /*...*/ }
10 static void write_led_node_hsv(int32_t val, const SENSOR_NODE *sn, const
      DATA_NODE *dn) { /*...*/ }

11 // Definition einer Datenpunktkollektion
12 static DATA_NODE dn_led[] = {
13   INTEGER_NODE("red"   , "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_rgb , write_led_node_rgb),
14   INTEGER_NODE("green", "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_rgb , write_led_node_rgb),
15   INTEGER_NODE("blue" , "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_rgb , write_led_node_rgb),
16   INTEGER_NODE("hue"   , "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_hsv , write_led_node_hsv),
17   INTEGER_NODE("sat"   , "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_hsv , write_led_node_hsv),
18   INTEGER_NODE("value", "", FEATURE_READABLE|FEATURE_WRITEABLE, 0, 255,
         read_led_node_hsv , write_led_node_hsv),
19   // [...]
20 };
```

Listing 4.12: Beispielhafte Datenpunkt-Definitionen des LED-Controllers.

Datenpunkte können Teile mehrerer unterschiedlicher Listen sein, um verschiedene Teilmengen in unterschiedlichen Services anzubieten. Allerdings ist eine Instanz eines Datenpunktes nur Mitglied einer einzigen Liste.

4.5.2.3 Service-Knoten

Serviceknoten können eine beliebige positive Anzahl an Datenpunkten aufnehmen und tragen selbst einen Namen. Zur Einrichtung müssen im Quelltext zunächst die passenden Datenpunkte deklariert werden, wie in Listing 4.12 bereits geschehen. Um einen Serviceknoten zu deklarieren, werden Hilfsmakros aus Listing 4.13 verwendet.

```
1 typedef struct _SERVICE_NODE {
2   const char    *name;
3   bool           bChanged;
4   DATA_NODE     *node;
5   const uint8_t  nodecount;
6 } SERVICE_NODE;

7 #define _countof(x) (sizeof(x)/sizeof(x[0]))
8 #define SERVICE(name, nodes) { name, false, nodes, _countof(nodes) }
```

Listing 4.13: Makrodefinitionen der Service-Ebene.

In Listing 4.14 ist die eigentliche Deklaration von Datenpunkten abgebildet.

```
// Definition einer Service-Kollektion
static SERVICE_NODE service_node[] = {
  SERVICE("led0", dn_led),
  SERVICE("led1", dn_led),
  SERVICE("led2", dn_led),
  SERVICE("led3", dn_led),
  SERVICE("led4", dn_led),
};
```

Listing 4.14: Beispielhafte Service-Definitionen des LED-Controllers.

4.5.2.4 Metadaten und Gerätedefinition

Die Liste der definierten Serviceknoten wiederum wird für die Device-Deklaration verwendet. Auch hierfür existieren Makros, die neben der Liste der Service-Namen auch die Deklaration der Meta-Elemente als Schlüssel-Werte-Paare ermöglichen (siehe Listing 4.15).

```
typedef struct {
  const char    *id;
  SERVICE_NODE  *pServices;
  const uint8_t  servicecount;
  const char    *additionalData; // Makros DESC_TUPEL und END_DESC oder
      NULL benutzen
} DEVICE_DESCRIPTION;

#define BEGIN_DESC(name, services) name, services, _countof(services),
#define DESC_TUPEL(key, iValue) key "\0" iValue "\0"
#define END_DESC() "\0"
```

Listing 4.15: Makrodefinitionen der Device-Ebene.

Instanziiert wird ein Gerät wie im folgenden Listing 4.16.

```
static DEVICE_DESCRIPTION device_desc = {
  BEGIN_DESC(NULL, service_node) // Device ID kommt aus dem EEPROM, daher
      NULL
  DESC_TUPEL("deviceType", "LED-Lighter")
  DESC_TUPEL("deviceDescription", "This is a LED controller with 5 RGB LED
       channels. It is designed to drive up 5 LED floodlights with up to
      500mA current on each channel.")
  DESC_TUPEL("vendor", "TUM Informatik - F13")
  DESC_TUPEL("vendorUrl", "http://www.os.in.tum.de/")
  DESC_TUPEL("hardwareRevision", "1.1")
  DESC_TUPEL("firmwareRevision", "r24")
  END_DESC()
};
```

Listing 4.16: Beispielhafte Geräte-Deklaration des LED-Controllers.

4.5.3 Menüstruktur und Konfiguration

Auch bei der Konfiguration der Menüstruktur der Firmware wird mit Makros und Funktionspointern ein lesbarer Code gewährleistet. Da Texte im Programmspeicher liegen, wird in den folgenden Listings an entsprechender Stelle stets das Makro `PROGMEM` verwendet.

Ab der `avr-gcc` Compilerversionen 4.7 kann stattdessen auch die Direktive `__flash` verwendet werden. Da dies ein relativ neues Feature des Compilers darstellt, wurde die Firmware noch auf dem `PROGMEM`-Makro basierend erstellt und müßte für den Einsatz mit anderen Plattformen und Compilern portiert werden.

4.5.3.1 Beschriftungselemente mit Unterstützung mehrerer Sprachen

Durch statische Listen ist auch eine Mehrsprachlichkeit der Beschriftungen des Displays gegeben, indem vorhandene Texte in den gewünschten Sprachen ASCII-codiert im ROM abgelegt werden. Für jedes Gerät ist dafür eine Zeichenkettendefinition wie in Listing 4.17 erforderlich.

```
1  // Definition einer Liste aller vorhandenen Strings. Das letzte Item dient
       der Bestimmung der Anzahl der Strings und muß enthalten sein.
2  enum { STR_NULL = 0, STR_LANGUAGE, STR_ITEM1, STR_ITEM2, STR_EXIT,
       STR_MAIN, /* [...] */ STR_NUM };

3  // Definition von Zugriffskonstanten
4  enum { LANGUAGE_ENGLISH, LANGUAGE_GERMAN };
5  uint8_t currentLanguage = LANGUAGE_GERMAN;

6  // Deklaration der Strings; implizit werden sie mit abschließendem "\0" im
       ROM abgelegt. Die Längen der Beschriftungen dürfen sich unterscheiden
       .
7  static const PROGMEM char str_null[]      = "";

8  static const PROGMEM char str_eng_lang[]  = "English";
9  static const PROGMEM char str_eng_item1[] = "Item One";
10 static const PROGMEM char str_eng_item2[] = "Item Two";
11 static const PROGMEM char str_eng_exit[]  = "Exit";
12 static const PROGMEM char str_eng_main[]  = "Main Menu";

13 static const PROGMEM char str_ger_lang[]  = "Deutsch";
14 static const PROGMEM char str_ger_item1[] = "Punkt Eins";
15 static const PROGMEM char str_ger_item2[] = "Punkt Zwei";
16 static const PROGMEM char str_ger_exit[]  = "Verlassen";
17 static const PROGMEM char str_ger_main[]  = "Hauptmenü";

18 // Erstellen von statischen Arrays der Strings pro Sprache
19 static const PROGMEM char* const str_array_eng[STR_NUM] = { str_null,
       str_eng_lang, str_eng_item1, str_eng_item2, str_eng_exit, str_eng_main
       };
20 static const PROGMEM char* const str_array_ger[STR_NUM] = { str_null,
       str_ger_lang, str_ger_item1, str_ger_item2, str_ger_exit, str_ger_main
```

```
    };

// Kombinieren der Sprach-Teile zu einem einzigen Array
static PROGMEM char const * const * language_array[] = {
  str_array_eng,
  str_array_ger
};

// Zugriff auf ein Sprach-Element
printf("%S", language_array[currentLanguage][STR_LANGUAGE]);
```

Listing 4.17: Deklaration für Beschriftungen des LCD-Inhalts. Das `PROGMEM`-Makro ist spezifisch für die AVR-Plattform und den AVR GNU Compiler `avr-gcc`. Zeilenumbrüche sind aus Platzgründen entfernt. Das großgeschriebene „S“ im `printf()`-Befehl liest einen String aus dem Programmspeicher.

Nachteilig wirkt sich eine unübersichtlichere Definition von Elementen bei einer großen Anzahl aus. Neben den eigentlichen Sprachelementen müssen auch die jeweiligen Adress-Pointer-Arrays korrekt mit Werten belegt werden. Es empfiehlt sich das Verwenden mehrerer Dateien, bei denen eine Korrelation über die Zeilennummer möglich ist, um den Überblick zu behalten.

Eine Alternative besteht in der Definition eines einzigen mehrdimensionalen Arrays für jedes Beschriftungselement, dessen einzelne Sprachelemente stets die gleiche Länge aufweisen (siehe Listing 4.18). Damit würden allerdings nicht verwendete Bytes mit überflüssigen `\0`-Bytes befüllt und Speicherplatz bleibt ungenutzt.

```
// Definition der Anzahl der Sprachen, auch für andere Programmteile
    erforderlich, z.B. bei der Auswahl der Sprache
#define NR_LNGS 2

// Definition von Zugriffskonstanten
enum { LANGUAGE_GERMAN, LANGUAGE_ENGLISH };
uint8_t currentLanguage = LANGUAGE_GERMAN;
const char str_lang[NR_LNGS][8]   PROGMEM = {"Deutsch","English"};
const char str_item1[NR_LNGS][11] PROGMEM = {"Punkt Eins","Item One"};

// Zugriff auf ein Sprach-Element, zur Compilezeit statisch aufgelöst
printf("%S", str_lang[currentLanguage]);
// Zugriff auf ein Sprach-Element, u.U. erst zur Laufzeit ausgewertet
printf("%S", str_lang[currentLanguage+1]);
```

Listing 4.18: Alternative Deklaration für Beschriftungen des LCD-Inhalts, die nicht verwendet wird. Nachteilig wirken sich die fest anzugebenden Array-Grenzen sowie der verschwendete Speicherplatz bei kürzeren Elementen aus. Von Vorteil ist aber die direkte Adressierbarkeit von Elementen, die auch zur Laufzeit erfolgen kann, da die Elemente stets gleich lang sind. Das großgeschriebene „S“ im `printf()`-Befehl liest einen String aus dem Programmspeicher.

4.5.3.2 Displayinhalte

Auf dem LCD wird stets ein einziger logischer Inhalt („Bildschirm“) dargestellt. Zur

Navigation innerhalb des Bildschirms dienen die Aufwärts-/Abwärtstasten („Auf/Ab“) sowie die Bestätigungstaste („OK“), jeweils kurz oder länger (ab ca. 0,3 s) betätigt.

Da eine Reihe von LCD-Inhalten gleich aufgebaut ist und dem Ändern von nur einer einzigen Variable im RAM dient, konnten die Inhalte in vier einzelnen Bildschirmtypen implementiert werden. Jeder Tastendruck wird in jeder Bildschirminstanz separat ausgewertet, wobei je nach Bildschirmtyp immer die selbst Logik implementiert ist.

Grundsätzlich sind zunächst sogenannte Items (siehe Listing 4.19) zu definieren, die aus einer Kombination von Callback-Funktion und Beschriftung bestehen. Die Callback-Funktion muß dabei einen `int8_t` als einzigen Parameter akzeptieren, der den Tastendrücke von Auf/Ab codiert. Die Callback-Funktion wird für jeden Tastendruck einmal aufgerufen, modifiziert dann den Maschinenzustand entsprechend und druckt den geänderten Wert mit `printf()` oder einer Variante davon aus.

```
1  // Deklaration der Callback-Funktion für Item 1
2  static void item_fun_item1(int8_t change) {
3    if (change) {
4      // Modifiziere Maschinenzustand, Überlauf beabsichtigt möglich.
5      test_val += change;
6    }
7    glcd_printf_P(PSTR("%4u"), test_val);
8  }

9  // Deklaration eines einzelnen Items (siehe auch vorherige
       Stringdefinitionen). Ein Item besteht aus dem Funktionspointer der
       Callbackfunktion und dem Index des Textes.
10 \static const PROGMEM item_t item_item1 = {item_fun_item1, STR_ITEM1};
11 \static const PROGMEM item_t item_item2 = {item_fun_item2, STR_ITEM2};

12 // Deklaration eines Mehrfachitems als Array von Items. Der letzte
       Parameter gibt einen Schreibschutz an, Item 2 kann hier nicht geändert
        werden.
13 static const PROGMEM item_t multi_item_main_array[] = {
14   {item_fun_item1 , STR_ITEM1, 0},
15   {item_fun_item2 , STR_ITEM2, 1},
16   // Bis zu vier weitere Elemente
17 };
18 static const PROGMEM multi_item_t multi_item_main = {multi_item_main_array
       , 2};
```

Listing 4.19: Deklaration von Menü-Elementen und Callback-Funktionen.

Dabei wird vor Aufruf der Callback-Funktion der Bildschirm schon entsprechend vorbereitet, damit der `printf()`-Aufruf an der richtigen Stelle mit der richtigen Farbe und der gewünschten Schriftart geschieht. Ist eine eigene Definition notwendig, wird der Bildschirm innerhalb der Callback-Funktion mit Grafikfunktionen selbst erstellt.

Ein Bildschirm besitzt somit drei Komponenten: Die aufzurufende Callback-Funktion, das betroffene Element (Item, Liste von Items oder Menü, siehe später), sowie ein Rücksprungziel, wenn der Bildschirm wieder verlassen wird, dargestellt in Listing 4.20.

```
1 // Deklaration eines Screens für mehrere Elemente
2 static const PROGMEM screen_t screen_main_screen = {screen_fun_multi_item
      , &multi_item_main, &screen_main_menu};
3 // Deklaration eines Screens für ein Menü mit mehreren Elementen (siehe
      später)
4 static const PROGMEM screen_t screen_main_menu   = {screen_fun_menu
      , &menu_main      , &screen_main_screen};
5 // Deklaration zweier Screens jeweils ein Element
6 static const PROGMEM screen_t screen_item1       = {screen_fun_single_item
      , &item_item1, &screen_main_menu};
7 static const PROGMEM screen_t screen_item2       = {screen_fun_single_item
      , &item_item2, &screen_main_menu};
```

Listing 4.20: Deklaration von Bildschirmen.

Wie im Listing erkennbar, sind drei Typen von Bildschirm-Funktionen, die den Aufbau und die Tastenbehandlung übernehmen, vorgegeben:

1. *Hauptbildschirm*, `screen_fun_multi_item`. Dieser Typ stellt bis zu sechs beliebige Variablen, übergeben in einer Liste von Items, dar. Mit Hilfe der `OK`-Taste werden nicht schreibgeschützte Variablen ausgewählt und dabei invertiert dargestellt. Durch `Auf`/`Ab` wird der ausgewählte Wert bis zur jeweiligen oberen oder unteren Grenze verändert oder läuft über. Ein langer Druck auf `OK` wechselt die Anzeige zum im Rücksprungfeld definierten Menü (in der Regel „Hauptmenü“).

2. *Einzelwertdarstellung*, `screen_fun_single_item`. Dieser Bildschirmtyp zeichnet den Namen des Items in der jeweiligen Sprache in den oberen Bereich und gibt den aktuellen Wert darunter aus. Mit Hilfe von `Auf`/`Ab` wird der Wert geändert und bei Druck auf `OK` der jeweils angegebene Rücksprungbildschirm aufgerufen.

3. *Animierte Menüdarstellung*, `screen_fun_menu`. Dieser Bildschirmtyp rendert ein Menü und erwartet eine Liste von (Bildschirm - Beschriftung)-Tupeln. Mit `Auf`/`Ab` wird durch die einzelnen definierten Menü-Elemente (Beschriftungen) navigiert, der Druck

(a) Multi-Item-Bildschirm. **(b)** Menü-Ansicht.

(c) Einzel-Item-Ansicht. **(d)** Eigendefinition.

Abbildung 4.30: Abbildung der unterschiedlichen Bildschirmtypen.

auf OK ruft den zugehörigen Bildschirm auf.

Die Deklaration von Menüs erfolgt wie in Listing 4.21 gezeigt.

```
1 // Menü-Elemente-Definition, Liste von (Screen::Beschriftung)-Tupeln
2 static const PROGMEM menu_entry_t menu_main_entry_array[] = {
3   {&screen_item1       , STR_ITEM1},
4   {&screen_item2       , STR_ITEM2},
5   {&screen_main_screen, STR_MAIN },
6 };
7 // Menü-Definition, Zuordnung von Menü-Elementen, Länge und Beschriftung
8 static const PROGMEM menu_t menu_main = {menu_main_entry_array, 3,
    STR_MAIN_MENU};
```

Listing 4.21: Deklaration von Menüs.

Durch die Verwendung der beschriebenen Struktur wird mit einfachen Definitionen ein Zugang zu Maschinenzuständen über das grafische LCD ermöglicht. Abbildung 4.30 zeigt die drei in der Bibliothek enthaltenen sowie einen individuell gestalteten Bildschirmtyp.

4.5.3.3 Speicherbelegung

Das Kapitel wird mit einer Angabe über die statische Speicherplatzbelegung in Tabelle 4.4 beschlossen.

	MCU	Ausstattung		ROM belegt		SRAM belegt	
Gerät	Bezeichnung	ROM	SRAM	kB	%	kB	%
LED-Controller v1	ATxmega16A4U	16 kB	3,3 kB	12,5 kB	78,1 %	1,2 kB	36,4 %
Umweltsensor	ATmega32U2	32 kB	1 kB	10,2 kB	31,0 %	0,5 kB	49,2 %
LED-Controller v2	ATxmega32A4U	32 kB	4 kB	28,9 kB	78,6 %	3,1 kB	76,2 %
USB-Dongle	ATxmega128A4U	128 kB	8 kB	14,7 kB	10,6 %	5,9 kB	72,9 %
Kühlschrankmodell	ATxmega128A4U	128 kB	8 kB	43,4 kB	31,2 %	3,5 kB	42,2 %
Ventilator	ATxmega128A4U	128 kB	8 kB	42,2 kB	30,3 %	3,4 kB	41,7 %

Tabelle 4.4: Speicherbelegung der erstellten Smart Devices.

Aus dem Vergleich zwischen USB-Dongle, dem LED-Controller v2 und den beiden anderen Geräten mit xmega128 MCU ist erkennbar, daß vor allem die Behandlung des grafischen LCD mit den notwendigen Textkonstanten das Gros an Speicherplatz beansprucht, während das Anwendungsprotokoll und RFM-Protokoll auch in einer MCU mit nur 32 kB ROM unterkommen.

4.6 Universeller Interpreter für Anwendungslogik

Abschließend wird eine beispielhafte Implementierung einer universell parametrisierbaren

Anwendungslogik für das Smart Home vorgestellt. Ziel ist zu demonstrieren, daß Anwendungslogik auch ohne Kenntnis von Programmier- oder Scriptsprachen in einer benutzerfreundlichen grafischen Darstellung erstellt werden kann. Insbesondere die begrenzten Ressourcen auf den Internet-Routern stellen dabei eine wesentliche Herausforderung dar.

Die Software wurde in [103] erfolgreich umgesetzt und konnte auf beiden Modellen des Internet-Gateways eingesetzt werden.

4.6.1 Funktionsbeschreibung

Neben dem vorgestellten Anwendungsprotokoll wird über eine flexible Protokolldefinition auch eine Zahl weiterer IP-basierter Produkte und Gateways unterstützt. Durch Auswahl von Transportprotokoll und Festlegen von Interpretationsregeln für Nutzdaten werden einfache Byte- sowie komplexere Text- und XML-basierte Protokolle auf Anwendungsebene innerhalb des Programms unterstützt.

Die so im Programm verfügbaren Werte („Kanäle“) können über statische Transformationen verändert und über Aktionen wiederum als Byte-, Text- oder XML-Nachrichten an andere Geräte versendet werden. Ein- und Ausgangsformat sind dabei unabhängig voneinander; eine Übersetzung zwischen den Protokollen kann realisiert werden.

Die konzipierte Software besitzt folgende Eigenschaften:

- Auf den Internet-Gateways steht nur sehr begrenzter Speicherplatz für Programme (~3,5 MB); das Programm ist deswegen in effizientem *C* geschrieben und soweit wie möglich gegen Shared Libraries gelinkt.
- Kanäle können auf einen externen USB-Speicher mit Wert und Zeitstempel in eine SQlite-Datenbank protokolliert werden; eine Auswertung erfolgt dann durch separate Programme.
- Für den Endanwender bietet das Programm eine eigene, auf JavaScript, XML und XSLT basierende Weboberfläche, mit der die Konfiguration festgelegt wird. Durch Verwendung dieser Techniken wird größtmögliche Kompatibilität mit einer Reihe von Endgeräten erzielt.

Ein Diagramm der Funktionsblöcke und ihrer Verbindungen untereinander ist in Abbildung 4.31 dargestellt. Die einzelnen Komponenten werden in den folgenden Unterkapiteln beschrieben.

4.6.1.1 IP-Endgeräte

Zunächst muß eine Liste der zu beteiligenden Endgeräte vorliegen, die über IP-Fähigkeiten verfügen, bereits am Netzwerk angebunden sind oder zumindest über ein konfiguriertes Routing erreichbar sind und eines der beiden Transportprotokolle UDP oder TCP verwenden. Die Implementierung unterstützt derzeit nur IPv4.

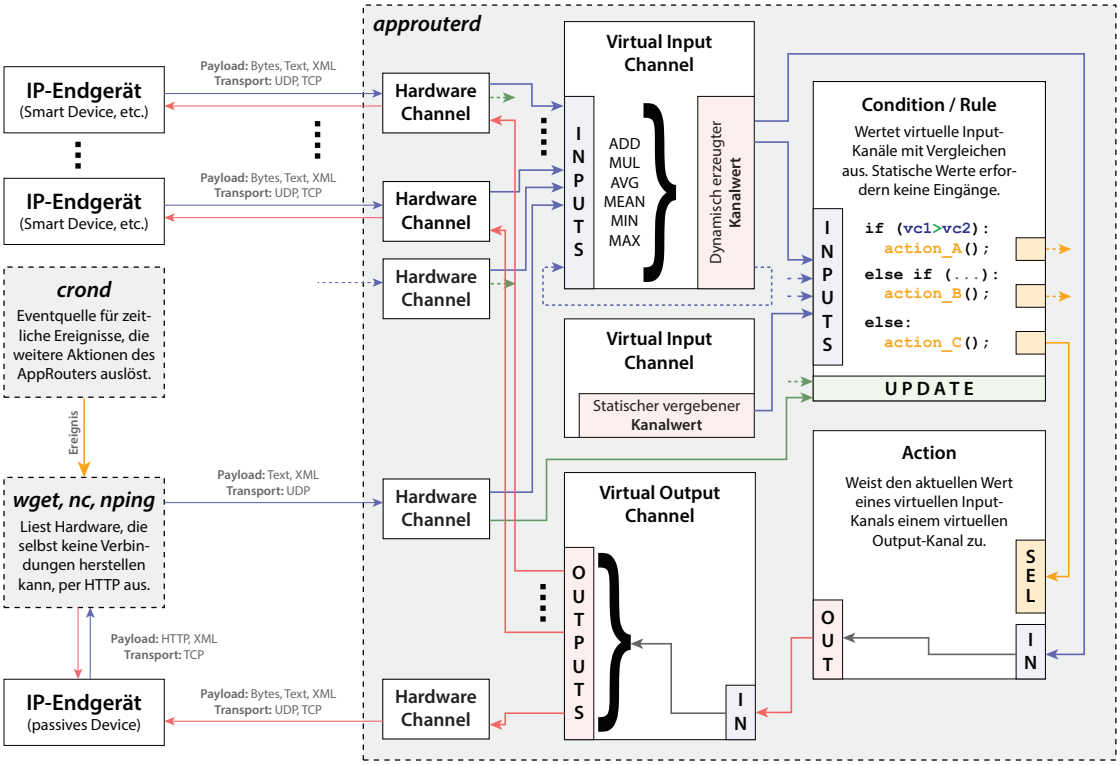


Abbildung 4.31: Übersicht über die Routerkomponente der Anwendungsschicht.

Das Programm erwartet von Geräten eine aktive Teilnahme, d.h., Geräte müssen geänderte Werte selbständig über etwa UDP-Broadcasts oder TCP/UDP-Unicast-Nachrichten melden („Events“). Sie können dabei auf Anwendungsprotokollebene weitgehend frei definierte Format verwenden (siehe Unterkapitel 4.6.1.4).

Durch ein Hilfsprogramm, in Unterkapitel 4.6.1.3 beschrieben, werden auch rein passive, nur zur Steuerung vorgesehene Geräte unterstützt, die keine Ereignisse versenden. Die Abfrage geschieht über HTTP. Sie werden periodisch auf neue Werte geprüft und die Ergebnisse an den Anwendungsrouter ohne Änderung weitergereicht.

4.6.1.2 Konzept des Hilfsprogramms *crond*

Um Ereignisse zeitgesteuert auslösen zu können, wird das Hintergrundprogramm („daemon“) `crond` konzipiert. Seine Aufgabe besteht im Versenden von bestimmten Inhalten, in der vorliegenden Implementierung statischer Text, zu vordefinierten konstanten Zeitpunkten an den Anwendungsrouter.

Die Konfiguration der zeitlichen Ereignisse erfolgt nach dem Schema der `cron`-Tabelle [104]. Die minimale zeitliche Auflösung beträgt damit eine volle Minute.

Das Programm sendet einen Text, z.B. `EREIGNIS:08:00UHR`, an die lokale IP-Adresse `localhost`

auf einen per Kommandozeile konfigurierbaren UDP-Port. Innerhalb des Anwendungsrouters kann dieses Ereignis dann zum Triggern von Aktionen verwendet werden, indem der übertragene Wert mit Hilfe eines einzelnen Hardware-Kanals innerhalb einer Regel durch einen Vergleich ausgewertet wird.

Sollen verschiedene Ereignisse ausgewertet werden, werden innerhalb der Regel einfach mehrere Vergleichsoperationen eingetragen. Eine entsprechende Zahl virtueller Eingangskanäle mit entsprechenden Textkonstanten ist dafür erforderlich.

Da die `crontab`-Syntax verwendet wird, kann auf Internet-Routern, auf denen `crond` installiert ist, dieses Programm verwendet werden, das allerdings auf das Hilfsprogramm `nc` (netcat [105]) zum Versand von Nachrichten angewiesen ist. Eine passende Kommandozeile für den Zielport 8080 ist in Listing 4.22 angegeben.

```
1 echo "EREIGNIS:08:00UHR" | nc -uc localhost 8080
```

Listing 4.22: Kommandozeile zum Versand von UDP-Nachrichten mit `nc`. Der Text ist mittels `echo` auf `stdin` auszugeben und wird über den Pipe-Operator an netcat übermittelt. Mit dem Parameter `-u` legt man UDP als Protokoll fest, `-c` veranlaßt ein Beenden des Prozesses nach Absetzen des Datagramms.

Sowohl `crond` wie auch `nc` stehen unter OpenWRT zur Verfügung. Auf eine eigenständige Implementierung des Hilfsprogramms konnte damit verzichtet werden. Auch grafische Benutzeroberfläche zur Konfiguration des `cron`-daemons wurde nicht implementiert. Sie ist zur Demonstration des Konzepts auch nicht erforderlich und könnte einfach in die vorhandene browserbasierte Oberfläche (siehe Unterkapitel 4.6.2.4) integriert werden.

4.6.1.3 Konzept der Hilfsprogramme *wget*, *nc* und *nping*

Unter Umständen versenden Smart Devices keine eigenen Nachrichten, sondern erwarten eine periodische Abfrage durch Clients. Bei Verwendung des konzipierten Anwendungsprotokolls ist das der Fall, wenn ein Smart Device keine Events versendet.

Es wird angenommen, daß Smart Devices direkt oder über Gateways per HTTP erreichbar sind. Sie werden dann zu bestimmten Zeitpunkten, siehe voriges Unterkapitel, mit dem Hilfsprogramm `wget` abgefragt, das wie in Listing 4.23 gezeigt konfiguriert wird.

```
1 root@gateway$ wget -qO- http://10.0.0.138:1045/ventilator_4/Ventilator/
    Drehzahl
2 root@gateway$ wget -qO- http://ventilator_4:1045/Ventilator/Drehzahl
3
3 <ventilator_4><fan><speed>506</speed></fan></ventilator_4>
```

Listing 4.23: Kommandozeile zum Abfragen eines Smart Devices mit `wget`. Das Argument `-qO-` veranlaßt die Ausgabe der Payload ohne HTTP-Header auf `stdout`. Wenn keine statischen IP-Adressen verwendet werden, kann auch der Hostname wie in der zweiten Zeile gezeigt verwendet werden. Hierbei wird von `wget` auch der `Host`-Header automatisch gesetzt.

Die Ausgabe des Gerätes wird wiederum über den Pipe-Operator an den Anwendungsrouter versendet, dargestellt in Listing 4.24.

```
wget -qO- http://ventilator_4:1045/Ventilator/Drehzahl | nc -uc localhost
    8080
```

Listing 4.24: Kommandozeile zum Versenden der Abfrage an den Anwendungsrouter. Die Ausgabe von `wget` wird per Pipe-Operator an `nc` weitergereicht und an den Anwendungsrouter versendet.

Die beschriebene Variante mit `netcat` hat den Nachteil, daß stets die Adresse des Internet-Routers als Absende-IP gesetzt wird. Um die IP-Adresse des Smart Devices zu verwenden, wird das Werkzeug `nping` [106] eingesetzt, das auch für OpenWRT verfügbar ist.

Mit Hilfe von IP-Adress-Spoofing [95, Kapitel 3.3.1] wird das UDP-Datagramm versendet, als würde es direkt vom jeweiligen Smart Device stammen (Listing 4.25). Innerhalb des Hardwarekanals ist die Parser-Regel entsprechend der Ausgabe des Smart Devices zu konfigurieren.

```
nping -S ventilator_4 -g 1045 -H --udp localhost -p 8080 -c 1 --quiet --
    data-string "$(wget -qO- http://ventilator_4:1045/Ventilator/Drehzahl)
    " # "$()" öffnet eine Sub-Shell.
```

Listing 4.25: Kommandozeile zum Versenden der Abfrage mit gefälschter Absenderadresse an den Anwendungsrouter. Die Ausgabe von `wget` wird über eine Sub-Shell `$()` an `nping` weitergereicht und von diesem an den Anwendungsrouter versendet. Die Optionen `-S` und `-g` bestimmen die Absenderangaben des Pakets, `-H` unterdrückt eine Ausgabe, `localhost` und `-p` bestimmen die Zieldaten, und mit `-c 1` wird nur ein einziges Paket versendet.

Im Anwendungsrouter wird so nur ein einziger Hardware-Kanal zur Abbildung eines Smart Devices benötigt.

4.6.1.4 Hardwarekanäle

Hardwarekanäle stellen innerhalb des Programms die Schnittstelle zu externen IP-Endgeräten dar. Jeder Kanal wird mit der IP-Adresse, dem Transportprotokoll und der Port-

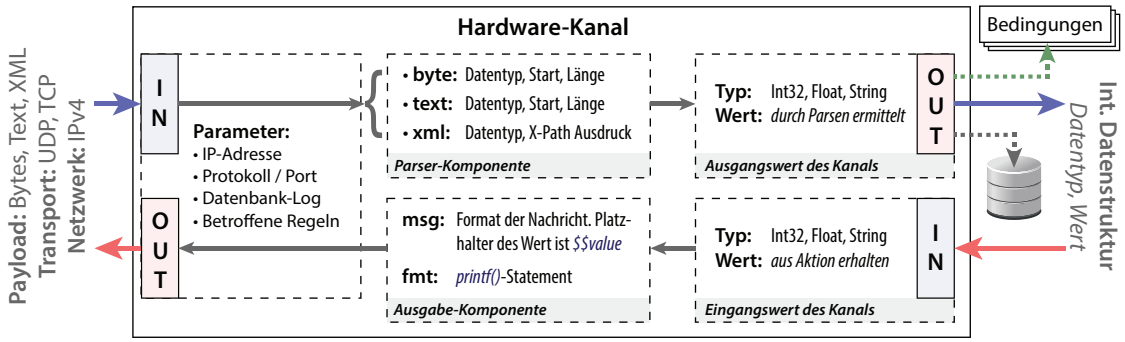


Abbildung 4.32: Anwendungsrouter: Schematische Darstellung eines Hardwarekanals. Im rechten Teil sind die optionalen Parameter der auszuführenden Regel(n) und des Speicherns des interpretierten Wertes in der SQlite-Datenbank abgebildet.

Nummer parametrisiert und besitzt einen Namen, die `channel_id`. Diese Daten werden zur Zuordnung von eintreffenden Nachrichten zu einem Hardwarekanal und gleichzeitig als Ziel-Informationen für auf dem Kanal ausgehende Pakete verwendet, siehe Abbildung 4.32.

Weiterhin ist optional eine Liste der Regeln enthalten, die bei Änderung des Hardware-Kanals zu aktualisieren sind. Die Aktualisierung wird stets nach Auswertung der betroffenen virtuellen Eingangskanäle durchgeführt, damit geänderte Werte in den Ausgangswert übernommen werden.

In Eingangsrichtung übersetzen sie von Smart Devices eintreffende IP-Nachrichten auf ein anwendungsinternes Interface, das nur den Datentyp (Ganzzahl, Fließkomma oder Zeichenkette) und den empfangenen Wert beinhaltet. Dazu wird die eintreffende Nachricht anhand der Konfiguration des Kanals ausgewertet, wobei drei verschiedene Inhaltsformate gewählt werden können:

1. *Byte-Interpretation.* Hierbei werden die durch Start und Länge angegebenen Bytes als Wert interpretiert. Ist mehr als ein Byte angegeben, so wird automatisch die Byte-Reihenfolge des Betriebssystems verwendet. Diese Auswahl dient der Anbindung kleinster Smart Devices, deren Nachrichten von Medienwandlern ohne eigene Interpretation im Netzwerk verteilt werden.

2. *Text-Auswahl.* Die Nachricht wird anhand eines gespeicherten Beginns und einer Länge für ausgewertet. Ein konstantes Nachrichtenformat und eine feste Position des Wertes wird dieser Option vorausgesetzt.

3. *XML-Inhalt.* Die Nachricht wird anhand eines gespeicherten X-Path-Ausdrucks ausgewertet. Der Ausdruck sollte einen einzigen Knoten zurückliefern; falls mehrere Knoten retourniert werden, wird entweder der in der Hierarchie am weitesten unten liegende Knoten, oder – bei gleichen Hierarchiestufen – ein zufälliger Knoten automatisch ausgewählt. Zur Verarbeitung von XML und X-Path wird eine reduzierte Version der `libxml`, nämlich die Bibliothek `libroxml` [107] eingesetzt. Das Binärobjekt belegt knapp 50 kB an Speicherplatz.

Optional wird ein geänderter Wert mit Zeitstempel in einer externen SQlite-Datenbank, siehe Unterkapitel 4.6.2.3, gespeichert und kann von dort für eine weitere Verarbeitung ausgelesen werden. Die beiden Internet-Router verfügen nur über wenig Speicherplatz, daher ist ein externes Speichermedium vorab einzubinden. Dies kann beispielsweise über USB („USB-Stick“) oder das Mounten eines Netzwerklaufwerks geschehen. Dem Anwendungsrouter ist der Pfad zur Datenbank zu konfigurieren, das Schema wird im genannten Unterkapitel beschrieben.

In Ausgangsrichtung wird ein Wert mit Hilfe von `printf()` in eine textuelle Repräsentation übersetzt und danach in eine Textvorlage anstelle der Zeichenkette `$$value` eingefügt. Die Textvorlage kann etwa aus einer HTTP-`POST`-Anweisung des Anwendungsprotokolls bestehen. Dank des `printf()`-Statements kann die Präzision von Fließkommazahlen an die erforderliche Zahl von Nachkommastellen angepaßt werden.

4.6.1.5 Virtuelle Eingangskanäle

Durch einen virtuellen Eingangskanal werden mehrere Hardware- oder weitere virtuelle Eingangskanäle, deren Werte v im Folgenden mit i indiziert und ihre gesamte Anzahl mit n bezeichnet sind, aggregiert. Ihre Datenstruktur besteht aus einer Liste der Quellkanäle und einer der folgenden Aggregationsfunktionen, die den Ausgangswert x berechnet:

- *ADD*. Addiert die Kanäle: $x = \sum_{i=1}^{n} v_i$
- *MUL*. Multipliziert die Kanäle: $x = \prod_{i=1}^{n} v_i$
- *AVG*. Arithmetisches Mittel der Kanäle: $x = \frac{1}{n} \sum_{i=1}^{n} v_i$
- *MEAN*. Geometrisches Mittel der Kanäle mit positiven Werten: $x = \sqrt[n]{\prod_{i=1}^{n} v_i}$
- *MIN*. Minimaler Wert der Kanäle: $x = \min\{v_1, v_2, \ldots, v_n\}$
- *MAX*. Maximaler Wert der Kanäle: $x = \max\{v_1, v_2, \ldots, v_n\}$

Auch das Ergebnis x der Aggregationsberechnung wird in der Datenstruktur im jeweiligen Datentyp gespeichert. Wie beschrieben dient der Ausgangswert anderen virtuellen Eingangskanälen als Quellwert und wird auch als Operand in Vergleichsoperationen von Regeln verwendet.

Virtuellen Eingangskanälen müssen keine Hardware-Kanäle zugeordnet sein; sie können auch einfach nur zum Speichern eines konstanten Wertes zur Verwendung als Operanden in Vergleichen oder zum Setzen eines konstanten Wertes in Hardware-Kanälen dienen.

4.6.1.6 Bedingungen

Jede Bedingung besteht aus einer Reihe von Vergleichsoperationen mit binärem Ergebnis (wahr/falsch), die sequentiell abgearbeitet werden. Trifft ein Vergleich zu, so wird die zugeordnete Aktion ausgelöst, ansonsten die nächste Operation in der Kette ausgeführt.

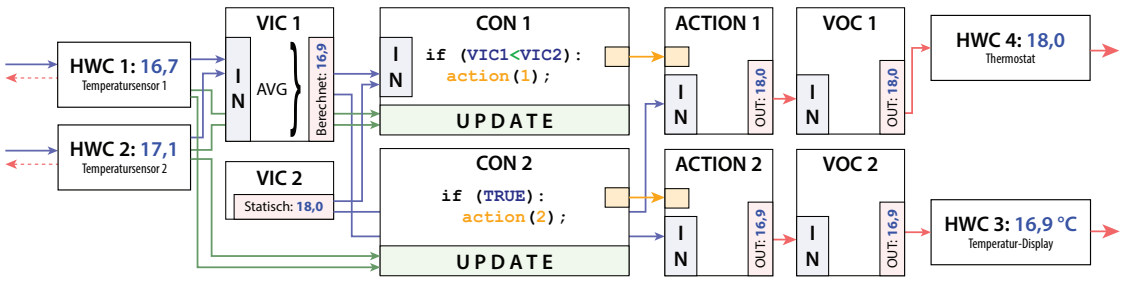


Abbildung 4.33: Darstellung einer beispielhaften Konfiguration mit vier Smart Devices. Die Temperatursensoren *1* und *2* melden ihre Werte an den virtuellen Eingangskanal 1 und veranlassen eine Aktualisierung der beiden Bedingungen. Ein Vergleich mit einem statischen Wert entscheidet daraufhin über das Senden der konstanten Temperatur an ein Thermostat, während das Temperatur-Display die Durchschnittstemperatur der beiden Sensoren bei jeder Änderung übermittelt bekommt.

Die Vergleichsoperation an sich besteht dabei aus zwei Operanden, $VC1$ und $VC2$, und einem der folgenden Operatoren:

- $>$: Wert von $VC1$ ist größer als der von $VC2$.
- $\geq$: Wert von $VC1$ ist größer oder gleich dem von $VC2$.
- $=$: Werte von $VC1$ und $VC2$ sind gleich (Ganzzahlen, Zeichenketten).
- $<$: Wert von $VC1$ ist kleiner als der von $VC2$.
- $\leq$: Wert von $VC1$ ist kleiner oder gleich dem von $VC2$.
- *true* Immer erfüllt, keine Operanden notwendig.

Pro erfüllter Bedingung kann nur eine einzige Aktion ausgeführt werden.

4.6.1.7 Aktionen und virtuelle Ausgangskanäle

Eine Aktion bildet die Verknüpfung zwischen einem virtuellen Eingangskanal und einem virtuellen Ausgangskanal. Über eine 1:1-Beziehung wird der momentane Wert des Eingangskanals in den Wert des Ausgangskanals übernommen und damit eine Aktualisierung aller Hardwarekanäle angestoßen. Abbildung 4.33 zeigt ein Beispiel.

Der von einer Aktion erhaltene Wert wird durch den virtuellen Ausgangskanal auf einen oder mehrere Hardware-Kanäle umgesetzt. Jeder Hardware-Kanal erzeugt die bereits beschriebene Nachricht und versendet sie an den konfigurierten IP-Adressaten. Die Abarbeitung der konfigurierten Kanäle geschieht sequentiell.

4.6.2 Konfiguration und Bedienung

Nach Beschreibung der Funktionen und internen Strukturen des Anwendungsrouters werden in diesem Teil einige externe Eigenschaften, wie Eckdaten zur Implementierung und Konfiguration, die Datenbankanbindung und die Benutzerschnittstelle erläutert. Auf eine detaillierte Diskussion des *C*-Codes wird wegen geringer Relevanz allerdings verzichtet.

4.6.2.1 Eckdaten der Implementierung

Der Anwendungsrouter wurde in *C* implementiert. Maßgeblich für der Entscheidung der Programmiersprache war die auf den Internet-Gateways anzutreffende Variante des Linux-Betriebssystems OpenWRT, auf der die Bibliothek `libc.so` für die bei Auslieferung installierten Programme zwingend erforderlich ist. Sie belegt je nach OpenWRT-Version ~1,4 MB, die bereits vom ohnehin geringen Speicherplatz abgezogen werden.

Der Verbrauch an Arbeitsspeicher variiert mit Anzahl der konfigurierten Bestandteile, liegt aber an der Untergrenze bei zirka 40 kB. Nachdem beide Internet-Router, sowie das Gros der käuflichen Produkte, stets ein Vielfaches ihres Festspeichers als Arbeitsspeicher zur Verfügung haben, treten hierbei keine Einschränkungen auf.

4.6.2.2 Konfigurationsdatei

Die gesamte Konfiguration des Anwendungsrouters wird auf Dateisystemebene in einer einzigen, XML-basierten Konfigurationsdatei festgehalten. Der wesentliche Grund für die Wahl dieser vom Speicherplatzverbrauch her ineffizienten Methode wird in Unterkapitel 4.6.2.4 ersichtlich. Für das Parsen kommt erneut die bereits genannte `libroxml` zum Einsatz.

Der Aufbau der XML-Datei ist linear gestaltet und führt im Basisschema die fünf Komponenten *Hardware-Kanal*, *virtueller Eingangskanal*, *Bedingung*, *Aktion* und *virtueller Ausgangskanal* linear nacheinander auf. Dies ist in Listing 4.26 dargestellt.

```
<approuter>
  <hardware_channels>
    <hwc><!-- 1. Hardware channel --></hwc>
  </hardware_channels>
  <virtual_input_channels>
    <vci><!-- 1. Virtual input channel --></vci>
  </virtual_input_channels>
  <virtual_output_channels>
    <vco><!-- 1. Virtual output channel --></vco>
  </virtual_output_channels>
  <rules>
    <rule><!-- 1. Rule --></rule>
  </rules>
  <actions>
    <action><!-- 1. Action --></action>
  </actions>
</approuter>
```

Listing 4.26: Basisaufbau der Konfigurationsdatei.

Jede Instanz der Unterknoten bildet dann den jeweiligen Datentypen mit einem eigenen Schema ab. Beispielhaft ist das für die Hardware-Kanäle in Listing 4.27 abgebildet.

```
<hwc>
  <id> <![CDATA[hw1]]> </id>
  <datatype> INT32 </datatype>
  <value> 123 </value>
  <address>
    <IP> 127.0.0.1 </IP>
    <port> 22222 </port>
    <transport_protocol> TCP </transport_protocol>
  </address>
  <rx>
    <application_protocol> XML </application_protocol>
    <xpath_selector> <![CDATA[/params/param/value]]> </xpath_selector>
  </rx>
  <tx>
    <output_msg> Neuer Wert: $$VALUE </output_msg>
    <printf_selector> %05i </printf_selector>
  </tx>
  <update_rules>
```

```
19      <rule> 0 </rule>
20      <rule> 2 </rule>
21    </update_rules>
22    <log_values> TRUE </log_values>
23  </hwc>
```

Listing 4.27: Detailaufbau der eines einzigen Hardwarekanals.

4.6.2.3 SQlite-Protokollierung für Hardware-Kanäle

Die Werte von Hardware-Kanälen können bei Änderung zur weiteren Verarbeitung durch externe Programme, z.B. Visualisierungen, in einen Datenspeicher kopiert werden. Häufig werden für die Verarbeitung nicht alle Werte aller Kanäle gleichzeitig benötigt, und möglicherweise kann auch eine Einschränkung hinsichtlich des gewünschten Zeitraums des Werteverlaufs getroffen werden.

Damit stellt eine im Umfang reduzierte Variante der Structured Query Language (SQL) [108, Part I.1] eine passende Umsetzung der Anforderung zur Abfrage der betroffenen Daten dar. Im Gegensatz zu einer reinen Textdatei, die Werte durch Trennzeichen separiert speichert und außer grundsätzlichen `seek()`-Befehlen keine Beschränkung der Auswahl der Daten bietet, können bei SQL detaillierte Kriterien zur Definition der gewünschten Rückgabemenge in einer Anfrage formuliert werden.

Im Anwendungsrouter wird die im Funktionsumfang reduzierte Implementierung *SQlite* [109] verwendet, die als reine *C*-Bibliothek verfügbar ist. Aus ihr werden nur die Features zum Erstellen einer Tabelle (`CREATE TABLE ...`) und zum Einfügen von neuen Datensätzen (`INSERT ...`) verwendet. Verarbeitende Programme implementieren auch Funktionen zur Selektion von Daten.

4.6.2.4 Benutzerschnittstelle im Webbrowser

Eine browserbasierte Bedienung ist durch Einsatz von JavaScript und XSLT möglich. Der Anwendungsrouter implementiert dazu einen rudimentären Webserver, der lediglich die Befehle `GET` und `POST` akzeptiert und statische URLs erwartet:

- `/` Diese URL wird von jedem Browser initial angefragt und liefert den HTML-Rahmen aus. Innerhalb des Rahmens werden die nächsten Dateien angefragt.
- `/config.xml` Diese Datei enthält die gesamte Konfiguration des Anwendungsrouters und wird vollständig im Browser als DOM-Objekt vorgehalten. Alle Änderungen der Konfiguration im Browser werden zunächst gegen das im RAM befindliche Dokument ausgeführt.
- `/translate.xlt` In dieser Beschreibung wird die Konfigurationsdatei in vom Browser lesbares HTML umgewandelt und dann gerendert.

- `/translate.js` Für die Benutzerinteraktion sind bestimmte clientseitige Browserfunktionen notwendig. Sie werden in dieser Datei implementiert.

Konzeptionell lädt also ein Clientbrowser vom Anwendungsrouter nur die in Unterkapitel 4.6.2.2 beschriebene Konfigurationsdatei, eine XSLT-Transformationsbeschreibung und ein Stück JavaScript-Code herunter. Mit deren Hilfe wird die Konfigurationsdatei vom Browser automatisch in eine HTML-Repräsentation umgewandelt, wobei die jeweiligen HTML-Elemente mit passenden `class=""`-Attributen versehen werden.

Weiterhin verarbeitet dann das clientseitige JavaScript-Programm die Benutzeraktionen im Browserfenster. Es stützt sich dabei auf die im vorigen Absatz genannten Attribute, um verschiedene Aktionen für verschiedene Elemente ausführen zu können: Ein Klick auf einen Kanalnamen etwa wandelt ein `<div>`-Element in ein `<input>`-Element mit passend vorbelegten Werten um, während ein Doppelklick auf den Datentyp diesen in ein DropDown-Menü verwandelte.

Mit Hilfe von Seitenreitern für die unterschiedlichen Komponenten des Anwendungsrouters wird die Oberfläche übersichtlich gehalten.

Beim Speichern der Konfiguration wird der umgekehrte Weg beschritten. Die vorhandenen oder durch JavaScript neu erzeugten Elemente werden im Webbrowser zu einer Konfigurationsdatei im XML-Format zusammengestellt. Die so erzeugte Beschreibung wird per `POST` an den Anwendungsrouter übermittelt, der die bisherige Konfigurationsdatei nur überschreibt. Anschließend wird, wie beim Starten des Programms auch, einfach die Konfigurationsdatei neu eingelesen und so die neue Parametrisierung übernommen.

4.6.3 Schnittstelle zu Stromversorgungsunternehmen

An den Anwendungsrouter können Stromversorgungsunternehmen über die in Kapitel 3.6.2 beschriebenen Wege (aktive Abfrage eines Webdienstes oder Benachrichtigung durch den Lieferanten mit HTTP oder im eigenen VPN) angebunden werden.

Zu beachten ist, daß der Anwendungsrouter und auch der Internet-Gateway potentiell als unsicher einzustufen sind. Eine Portfreigabe in das öffentliche Internet einzurichten, die die Steuerung der elektrischen Verbraucher im Smart Home ermöglicht, sollte gegen die Sicherheitsrisiken abgewogen werden. Insbesondere bieten weder der Anwendungsrouter noch das potentiell mit geringen Rechen- und Firewallressourcen ausgestattete Internet-Gateway ausreichend Schutz gegen Angriffe; insbesondere, da im Anwendungsrouter keine IT-Sicherheit implementiert ist

Beide Methoden werden allerdings der Vollständigkeit halber vorgestellt.

4.6.3.1 Aktives Abfragen der Energiesituation

Bei diesem Konzept wird ein Webdienst den Energielieferanten, möglicherweise im Internet, oder auch im Smart Meter – eingebunden in das LAN – mit bestimmten Parametern abgefragt. In der Regel umfassen diese eine eineindeutige Benutzerkennung und ein zugehöriges Kennwort zur Personalisierung der Antwort des Webdienstes. Anhand der Benutzerdaten sind server-seitig wichtige Parameter wie geographische Lage und Kundendaten bekannt.

Erfolgt die Übertragung über ein öffentliches Netz, so ist eine geeignete Sicherung – beispielsweise mit HTTPS, das auch von `wget` unterstützt wird – einzusetzen, um gegen Manipulationen durch Dritte geschützt zu sein. Die Abfrage wird per `cron` und `wget` in geeigneten Intervallen, etwa alle 5 min, ausgeführt und das Ergebnis an einen Hardware-Kanal des Anwendungsrouters übermittelt. Hierbei sind lediglich die `cron`-Tabelle und die entsprechenden `wget`-Parameter geeignet anzupassen, wie bereits in Kapitel 4.6.1.2 und 4.6.1.3 beschrieben.

Über die Schnittstelle kann im Rahmen der Arbeit keine Aussage getroffen werden. Im Idealfall liefert sie die Information des Strompreises so kompakt wie möglich aus, z.B. in einer HTTP-Antwort als reinen Text. Dies ist keine Notwendigkeit; der Hardware-Kanal kann mit Hilfe des Protokollinterpreters auf eine Vielzahl unterschiedlicher Eingangsnachrichten parametrisiert werden.

Der Hardware-Kanal wiederum aktualisiert den Wert eines virtuellen Eingangskanals und löst die Abarbeitung einer Reihe von Regeln aus, die der Steuerung einzelner Geräte dienen.

4.6.3.2 Benachrichtigung durch Energielieferanten

Bei diesem Konzept sind `cron` und `wget` nicht mehr notwendig. Die Benachrichtigung des Energielieferanten, die über eine TCP-Verbindung eintrifft, wird direkt an den Anwendungsrouter zugestellt.

Zunächst ist daher eine ständige, sichere Verbindung zwischen Kunde und Energieversorger einzurichten, die neben Verschlüsselung auch Vertraulichkeit bietet. Ein virtuelles privates Netz, beispielsweise mit OpenVPN [110] umgesetzt, erscheint für die Anwendung geeignet:

- OpenVPN stellt die VPN-Verbindung nach Unterbrechung der Internetverbindung sobald wie möglich selbständig wieder her.
- Es kann für das Tunneln der OSI-Schichten 2 oder 3 verwendet werden und belegt damit nur ein Minimum an Ressourcen.
- Es unterstützt den Transport seiner eigenen Daten über UDP.
- IT-Sicherheit wird entweder zertifikatsbasiert oder per Preshared-Key gewährleistet. Damit sind die Kriterien Authentizität und Verschlüsselung erfüllt.

Auf dem Internet-Gateway wird die Zone, die das lokale Interface beinhaltet, isoliert. Aus ihr heraus sind keine Verbindungen in das lokale Heimnetz oder die Internetverbindung möglich. Umgekehrt können aber auch Geräte des Heimnetzes keine Daten über das VPN versenden.

Da der Anwendungsrouter lokal auf dem Internet-Gateway Zugriff auf die OpenVPN-Schnittstelle besitzt, reicht es, einen auf einem vereinbarten Port hörenden Socket auf der entsprechenden VPN-Schnittstelle zu erstellen. Die Nachrichten des Energielieferanten werden direkt an den Anwendungsrouter übermittelt. Ein Schutz gegen Angriffe aus dem Netz des Energielieferanten heraus ist so allerdings nicht geboten.

Das verwendete Nachrichtenformat wird im Hardware-Kanal passend für den Strompreis und den Fall des Lastabwurfs parametrisiert. Mit Hilfe von Bedingungen werden die entsprechenden Nachrichten an die Smart Devices abgesetzt. Bei statischen Befehlen, die keine Werte-Ersetzung erfordern, kann auf virtuelle Eingangskanäle verzichtet werden; die passenden Nachrichten werden als statischer Text im Hardware-Kanal hinterlegt.

5 Ergebnisse und Evaluierung

Die in den Kapiteln 3 und 4 konzipierten und implementierten Komponenten werden abschließend – wo passend – gegen bereits existierende Lösungen allgemein wie auch gegen die in der Einleitung aufgestellten Kriterien evaluiert. Das erstellte Funkprotokoll für die RFM70-Module wird dazu getrennt von den beiden übrigen Teilen, dem Anwendungsprotokoll und der konstruierten Hardware, behandelt.

5.1 Anwendungsprotokoll

Zunächst wird das entworfene Anwendungsprotokoll SHDP bewertet. In diesem Unterkapitel interessiert ein Vergleich mit den Anwendungsprotokollschichten der untersuchten, hauptsächlich Byte-basierten Steuerungslösungen aus Kapitel 2.1 und den im OSI-Modell höher angesiedelten reinen Anwendungsprotokollen aus Kapitel 2.2.

Zur Bewertung werden die Kriterien *Funktionen*, *Nachrichtengröße*, *Erweiterbarkeit* und *Skalierbarkeit* der einzelnen Lösungen herangezogen und jeweils gegenübergestellt.

5.1.1 Funktionen der Anwendungsebene

Da nicht immer alle OSI-Schichten in allen Produkten und Protokollen implementiert sind, werden nur die Funktionen auf *Anwendungsprotokollschicht* verglichen. Eckdaten des Nachrichtentransports der Produkte spielen in diesem Unterpunkt keine Rolle.

Tabelle 5.1 bietet eine Übersicht über die in dieser Arbeit behandelten Lösungen. Sie ist in drei größere Kategorien unterteilt. Auf der linken Seite finden sich die fünf derzeitig verfügbaren integrierten Steuerungsprodukte, im Mittelteil sind die reinen Anwendungsprotokolle gelistet und in der letzten Spalte ist das entworfene Anwendungsprotokoll dargestellt.

Während die reinen XML-Anwendungsprotokolle Funktionen zur semantischen Beschreibung von Geräten und Diensten für menschliche Benutzer bieten, ist das bei den Anwendungsschichten der untersuchten Heimsteuerungsprodukten nicht der Fall. Die notwendigen Metainformationen und Bedeutungen der einzelnen Nachrichtenteile sind nicht im Protokoll, sondern implizit über die Protokollspezifikation festgelegt und werden innerhalb

	Z-Wave	ZigBee	EnOcean	HomeMatic	KNX	HTTP	XMLRPC	SOAP, DPWS	CoAP / CoRE	SHDP
	Gerätespezifische Funktionen									
Entdecken verbundener Geräte	✗	✗	✗	✗	✓	✗	✗	✓	✓	✓
Metainformationen	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
	Dienstespezifische Funktionen									
Entdecken von Diensten	✗	✓	✗	✗	✗	✗	✓	✓	✓	✓
Metainformationen	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
Unicast-Konsum[1]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Multicast-Konsum[2]	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓
Broadcast-Konsum[3]	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
Relative Werte numerischer Dienste	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	Geräte-initiierte Ereignisse									
Ereignisnachrichten	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
Direkte Verknüpfung zwischen Geräten	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
	Sonstiges									
Verarbeitung als Nachrichtenstrom	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓

Tabelle 5.1: Gegenüberstellung der Funktionen der jeweiligen Anwendungsprotokollteile und direkt der Anwendungsprotokolle. Sofern eine Eigenschaft nur durch „externe“ Hilfsmittel (Datei oder ähnlich) oder Interaktion (Knopfdruck) und nicht automatisch ermöglicht wird, so wird sie als auf Anwendungsprotokollebene „nicht verfügbar“ gewertet.

der Implementierungen der jeweiligen Geräte berücksichtigt.

Bei Insellösungen, in denen ein Standardisierungsgremium die von allen Herstellern zu akzeptierenden Funktionen, Formate und Codes vorgibt, ist dieses Vorgehen anwendbar. Sind aber viele gleichberechtigte Hersteller auf eine Lösung angewiesen und an ihr beteiligt, so schlägt der Ansatz im Allgemeinen aus im Kapitel 2 angegebenen Gründen fehl.

Aus der Tabelle werden folgende Ergebnisse abgelesen:

- Bei den gängigen Heimsteuerungsprodukten werden verbundene Geräte in der Regel nicht automatisch erkannt. Eine Aufzählung („enumeration“, „discovery“) findet

1 *Unicast-Konsum* bezeichnet den Aufruf eines einzigen Dienstes mit einem einzigen Wert in einer einzigen Nachricht.

2 *Multicast-Konsum* bezeichnet den Aufruf von zwei oder mehreren Diensten mit ihren –durchaus verschieden belegten– Werten in einer einzigen Nachricht.

3 *Broadcast-Konsum* bezeichnet das Schreiben eines bestimmten Argumentes an *alle* auf einem Gerät verfügbaren Dienste in einer einzigen Nachricht.

manuell statt, indem entweder am Konsumenten die Adresse des Geräte händisch eingetragen wird oder das Gateway und das neue Gerät über einen Anlernprozeß statisch miteinander verbunden werden. Semantische Beschreibungen von Geräten über Metainformationen für menschliche und maschinelle Clients sind bei XML-basierten Protokollen aufzufinden.

- Auffinden und Beschreiben angebotener Dienste erfolgt bei Heimsteuerungsprodukten manuell über vorab getroffene Vereinbarungen über Zuordnungstabellen, die sowohl dem Gerät wie auch dem Konsumenten bekannt sind. Erst mit größerer Abstraktion auf Anwendungsprotokollebene wird ausreichend Information für eine automatische Erkennung übertragen.
- Große Unterschiede bestehen beim Konsum, d.h. dem Aufruf und Ausführen, von Diensten. Die Mehrheit der Lösungen unterstützt keinen Multi- oder Broadcast-Konsum. Damit muß eine Funktion, die die Interaktion mehrerer Geräte erfordert, in mehreren Unicast-Nachrichten abgearbeitet werden. Bei geringen Datenraten der verknüpften Nachrichtentransportsysteme ergibt sich daraus ein Nachteil.
- Als einziges Protokoll unterstützt SHDP das Feature der relativen Werte für numerische Datenpunkte. Es besitzt damit ein Alleinstellungsmerkmal.
- Ereignisnachrichten und deren Verarbeitung auf Geräten und Clients werden von den meisten der untersuchten und allen wichtigen Protokollen unterstützt.
- Zwar liegen über die verwendeten Produkte keine konkreten Informationen über die Codierung auf Anwendungsschicht vor, es kann aber wegen begrenzter Ressourcen von einfachen sequentiellen Codierungen ausgegangen werden. Damit wird auch eine Verarbeitung als Nachrichtenstrom bei allen Produkten außer SOAP und DPWS unterstützt. Sie können mit zeichenorientierten Zustandsautomaten bearbeitet werden.

Es konnte somit gezeigt werden, daß das entworfene Protokoll auf Anwendungsebene alle wichtigen Kriterien und aufgestellten Anforderungen erfüllt und auch ein Alleinstellungsmerkmal gegenüber dem Stand der Technik besitzt.

5.1.2 Nachrichtengrößen

Da der Vorteil, viele Funktionen in einem Protokoll zu vereinen, mit Abstraktion und redundanter Codierung von Information durch textuelle Repräsentation erkauft wird, ist das Protokoll hinsichtlich der Nachrichtengröße zu untersuchen und mit den übrigen Lösungen zu vergleichen.

Zu diesem Zweck werden fünf Anwendungsfälle betrachtet. Tabelle 5.2 gibt, sofern anwendbar, typische Nachrichtengrößen der entsprechenden Protokolle an.

Auch wenn eine Vergleichbarkeit der Anwendungsprotokollschichten (von Produkten und reinen Anwendungsprotokollen) aufgrund unterschiedlicher Konzepte von Geräte- und Dienstedefinition nur schwer möglich ist und nur qualitativ durchgeführt werden konnte,

	Z-Wave	ZigBee	EnOcean	HomeMatic	KNX	HTTP	XMLRPC	SOAP, DPWS	CoAP / CoRE	SHDP
	Funktionen auf einem Gerät									
Discovery	✗	✗	✗	✗	✗	✗	✗	10 000 B	300 B	4500 B
Diensteliste	✗	*k.I.*	✗	✗	✗	3000 B	5000 B	5000 B	300 B	1500 B
	Konsum von Diensten									
Unicast	10 B	15 B	15 B	15 B	20 B	100 B	500 B	1000 B	50 B	100 B
Multicast	✗	✗	✗	✗	✗	✗	1000 B	2000 B	✗	150 B
Broadcast	✗	15 B	✗	✗	20 B	✗	✗	✗	✗	90 B

Tabelle 5.2: Gegenüberstellung typischer Nachrichtengrößen der jeweiligen Anwendungsprotokollteile. Es wird jeweils die Anfrage und eine eventuelle Bestätigung addiert angegeben. Alle Angaben stellen nur qualitative Größen dar und hängen von konkreten Dienstedefinitionen und übertragenen Werten ab.

so ist aus den typischen Nachrichtengrößen dennoch ein Trend erkennbar:

- Die mit Abstand geringsten Nachrichtengrößen bieten erwartungsgemäß die Protokolle mit der geringsten Abstraktion, in diesem Falle die bisherigen Heimsteuerungslösungen.
- Auf der anderen Seite versenden innerhalb der in der Tabelle getroffenen Klassifikation die XML-basierten Anwendungsschichtprotokolle, die für eine Heimsteuerung geeignet wären, die größten Nachrichten; hauptsächlich ist das der beinhalteten Schemainformation (Datentypen, Hierarchie, usw.) in jeder übertragenen Nachricht geschuldet.
- Das neu konzipierte CoAP in Kombination mit CoRE bildet hiervon eine Ausnahme und nimmt für vergleichbare Nachrichten im Gegensatz zu den übrigen Protokollen um Längen weniger Speicherplatz in Anspruch bei annähernd gleicher Funktion, bleibt selbst aber auch deutlich über den bisherigen Heimsteuerungsprodukten.
- Das in dieser Arbeit konzipierte Anwendungsprotokoll ist im Mittelfeld einzuordnen; auch wenn die (selten ausgeführte) Discovery am meisten Übertragungsvolumen beansprucht, kann bei Abfrage von nur wenigen Diensten kein Vorteil bezüglich der Nachrichtengröße gegenüber bisherigen Produkten und CoAP/CoRE gewonnen werden. Das Protokoll benötigt für typische Aufgaben mehr Übertragungsvolumen als konkurrierende Lösungen.

Damit muß der Schluß gezogen werden, daß rein von der Nachrichtengröße her ein Nachteil gegenüber dem bisherigen Stand der Technik existiert. Das Protokoll muß daher in anderen Bereichen gegenüber vorhandenen Produkten Vorteile bieten.

5.1.3 Erweiterbarkeit der Gerätebeschreibungen und Dienstedefinitionen

Während Byte-orientierte Protokolle aufgrund ihrer impliziten Bit-Interpretation von Nachrichten generell nur schwer bis gar nicht erweiterbar sind, stellt dies für abstraktere, XML-basierte Protokolle kein Hindernis dar. Die Geräte- und Dienstebeschreibungen korrelieren hierbei mit den aktuell auf dem Gerät vorhandenen Eigenschaften und Funktionen und können unabhängig von anderen Teilnehmern angepaßt werden.

Im Detail unterscheidet sich SHDP von den anderen Protokollen wie folgt:

- Zunächst einmal bietet XMLRPC keine Gerätebeschreibungen. Die Methodenbeschreibungen beschränken sich auf eine einzige Zeichenkette pro Methode. Dem gegenüber können im Anwendungsprotokoll beliebige Metainformationen für ein Gerät sowie fix definierte Attribute für Dienste angegeben werden. Eine semantische Beschreibung der Dienste erfolgt über ihren Namen.
- SOAP bietet für Geräte und Dienste freie und vom Entwickler festzulegende Attribute und Beschreibungen. Sie werden in einer externen Schemabeschreibung vorgehalten und können gleichzeitig in mehreren Sprachen vorhanden sein. Knotennamen sind im Schema konstant und müssen keine sprechenden Namen tragen.
- Eng daran orientiert sich DPWS, das im Gegensatz zu SOAP allerdings bestimmte Attribute für Gerätetypen und konkrete Modelle definiert und so zumindest eine Gerätebeschreibung semantisch formalisiert.
- Einen anderen Ansatz verfolgt CoRE. Hierbei liegt der Fokus weniger auf einer Gerätebeschreibung, sondern auf einer kompakten und standardisierten Darstellung von Diensten des Gerätes durch statisch definierte Attribute mit WADL, die sowohl für Maschinen wie auch für Menschen lesbar sind.

Hinsichtlich der Flexibilität und Erweiterbarkeit der Beschreibungen von Diensten und ihrer Attribute können die Protokolle SOAP, DPWS, CoRE und SHDP als gleichwertig eingestuft werden. Sie alle bieten die Möglichkeit, eine unbegrenzte Zahl von Diensten mit festgelegten Attributen zu beschreiben. Am ehesten ist SHDP hier mit CoRE vergleichbar, bietet die Funktionalität aber über die Standards XML und HTTP auf Kosten einer höheren Nachrichtengröße.

Bei Gerätebeschreibungen ist SHDP mit DPWS, das einige Attribute im Schema festlegt, vergleichbar. Es bietet allerdings dank unbegrenzt möglicher `meta`-Elemente umfangreichere und flexiblere Möglichkeiten der Gerätebeschreibung.

5.1.4 Skalierbarkeit

Hier ergeben sich zwei verschiedene Klassen. Die maximal mögliche Anzahl an Geräten hängt bei den untersuchten Heimsteuerungsprodukten in der Regel von Nachrichtentransport ab und skaliert im begrenzten Rahmen der verfügbaren Adressfelder. Dienste sind pro Geräteklasse, -profil oder -typ statisch definiert und skalieren somit nicht.

Auf der anderen Seite bieten die untersuchten Anwendungsprotokolle keine Limitierung hinsichtlich der Anzahl der unterstützten Geräte oder Dienstedefinitionen. Entwickler und Hersteller können die Anwendungsschnittstelle frei nach Anforderung konzipieren.

Auch SHDP skaliert als Anwendungsprotokoll hinsichtlich Geräteanzahl und angebotener Dienste pro Gerät unbegrenzt. Limitierungen ergeben sich nur durch die auf den Geräten vorhandenen Ressourcen sowie dem verwendeten Nachrichtentransportsystem.

5.2 RFM70-Funkprotokoll

Bei der verwendeten Antenne (PCB-Antenne) handelt es sich keinesfalls um einen Rundstrahler. Schon eine Verdrehung um einen Winkel von 5° bis 10° konnte die Verlustrate von Paketen bei ansonsten gleich gebliebenen Parametern drastisch erhöhen.

Nach einer Abschätzung zur Verwendbarkeit des Protokolls mit anderen Funkmodulen werden vier Meßreihen beschrieben, die die Antwortzeiten der Smart Devices von und zu unterschiedlichen Clients angeben, und die teilweise überraschenden Ergebnisse interpretiert. In weiteren Meßreihen wird die Antwortzeit auf Anwendungsebene bestimmt, in der eine vollständige XML-Antwort über HTTP angefordert werden konnte.

Abschließend werden die beiden Betriebsmodi des Protokolls gegeneinander ausgewertet und hinsichtlich ihrer Effizienz beurteilt.

5.2.1 Allgemeine Verwendbarkeit

Ziel war, ein möglichst flexibel einsetzbares Protokoll zu konzipieren, das vom verwendeten Kommunikationsmodul keine besonderen Eigenschaften (Belegt-Erkennung, Bestätigung, usw.) erfordert. Damit ist eine Vielzahl von Übertragungsmodulen verwendbar.

Aus Sicht des Protokolls, das beliebige Nutzdaten und nicht nur Ethernet-Frames übertragen kann, ist die einzige Variable die Länge der Teildaten („Chunks“), in welche die Nutzdaten aufgeteilt werden. Sie kann im Quelltext an die Paketgröße des jeweiligen Funkmoduls angepaßt werden.

Unabhängig vom Protokoll muß auch der Treiber für das jeweilige Kommunikationsmodul angepaßt werden. Die Verbindung erfolgt asynchron über Callback-Funktionen und – sofern von der Hardware unterstützt – Interrupts. Da das Protokoll in *C* geschrieben ist, kann es prinzipiell für eine Reihe eingebetteter Hardware portiert werden. Somit wurde das Ziel einer allgemeinen Verwendbarkeit erreicht.

5.2.2 Performanz der Funkverbindung auf OSI-Vermittlungsschicht

Mit der vorhandenen Implementierung, bestehend aus RFM70-Funkmodul und den vorgestellten Smart Devices, wurden Messungen zur Latenzzeit und Datenrate für einen Sender, den USB-Dongle, und einen Empfänger, repräsentiert durch ein Ventilator-Gerät und einen LED-Controller, durchgeführt.

Da das Funkmodul im 2,4 GHz-Band operiert, ist eine Interferenz mit WiFi-Signalen denkbar. Um dieses Risiko für die Messungen zu minimieren, wurden zum einen alle lokalen Access-Points deaktiviert und zum anderen die Messungen auf vier anderen Kanälen des RFM70-Funkmoduls durchgeführt. Beide Maßnahmen hatten keinen statistisch signifikanten Einfluß auf die Reaktionszeit und die Paketverlustrate des RFM70-Moduls.

Stattdessen scheint aufgrund des Antennendesigns eine Ausrichtung der Module zueinander viel größeren Einfluß auf die Bitfehler und damit die Paketverlustrate zu besitzen. Die folgenden Messungen wurden daher bewußt in zwei verschiedenen Lagen von Sender und Empfänger zueinander durchgeführt, wobei durch absichtliches Fehlpositionieren zum einen eine möglichst ungünstige Lage festgelegt wurde und zum anderen mit einer durch Probieren ermittelten bewußt günstig gewählten Ausrichtung die Fehlerrate im Rahmen der Messungen minimiert werden konnte („Best-Case“).

Um den Einfluß durch die zusätzlichen Tasks der Firmware bei Verwendung eines LCD (Display und Tasten) unabhängig betrachten zu können, wurde neben dem Ventilator-Gerät auch ein LED-Controller untersucht.

Zur Ermittelung der Reaktionszeit der Geräte an sich und zur Abschätzung der Latenzzeit der Funkverbindung wird das in den meisten Betriebssystemen vorhandene Werkzeug `ping` verwendet, das direkt auf dem Internetrouter ausgeführt wird und hierbei 1200 ICMP `echo-request`-Pakete im Abstand von einer Sekunde an das jeweilige Gerät schickt. Es kommen dabei unterschiedliche Längen von Ping-Paketen zum Einsatz, womit sich die Wahrscheinlichkeit eines Gesamtverlustes bei Teilverlust eines RFM-Paketes erhöht, da mehrere Teilpakete für ein Frame benötigt werden.

Die grundsätzliche Länge $l_{Frame_{min}}$ eines ICMP-Paketes in der Übertragung mit RFM70 beträgt $l_{Frame_{min}} = l_{Header_{MAC}} + l_{Header_{IP}} + l_{Header_{ICMP}} = 14 + 20 + 8 = 42$ B[1]. Die kleinstmögliche Größe an Nutzdaten, bei der das `ping`-Kommando unter OpenWRT Zeitwerte ausgibt, beträgt 4 B, daher wird dieser Wert als Mindestgröße der Meßreihe festgelegt. Um die Paketverlustrate der Funkmodule bei unterschiedlichen Bedingungen abzuschätzen, werden auch größere Nutzdaten übertragen:

- Mit 4 B ICMP-Nutzdaten ergibt sich eine Framegröße von 46 B, 2 RFM70-Pakete.
- Mit 34 B ICMP-Nutzdaten ergibt sich eine Framegröße von 76 B, 3 RFM70-Pakete.
- Mit 94 B ICMP-Nutzdaten ergibt sich eine Framegröße von 136 B, 5 RFM70-Pakete.

1 Das Padding auf die minimale Länge des Ethernet-Frames von 64 B bezieht sich nur auf die Ausgabe des Frames auf ein entsprechendes Medium. Bei Verwendung der USB-Netzwerkkarte und im Stack des Betriebssystems werden die Padding-Bytes nicht hinzugefügt.

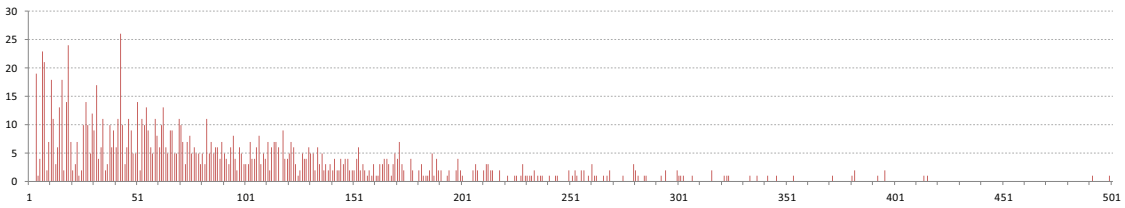
Die gemessenen Antwortzeiten werden gespeichert und deren Ergebnisse in den folgenden Tabellen und Abbildungen dargestellt. In allen Diagrammen zeigt die Y-Achse die Häufigkeiten und erhält keine Einheit, der Maximalwert wird angepaßt dargestellt. Auf der X-Achse ist die jeweilige Zeit in Millisekunden bis 501 ms angetragen. Diese Skalierung wird der besseren Vergleichbarkeit halber für alle Diagramme beibehalten. Verlorene Pakete sind nur in den jeweiligen Tabellen aufgeführt, nicht aber in den Diagrammen eingezeichnet.

5.2.2.1 Ergebnisse bei ungünstiger Ausrichtung, am Internet-Gateway gemessen

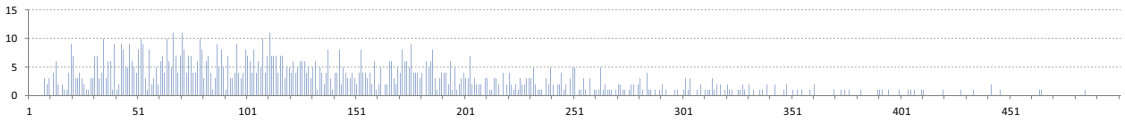
In Abbildung 5.1 sind die Ping-Häufigkeiten des Ventilators angetragen.

Erkennbar ist eine nach Framegröße deutlich schlechtere Aufteilung der Antwortzeiten des Gerätes. Zwar werden die RFM-Teilpakete per Interrupt vom Funkmodul nahezu verzögerungsfrei der MCU angezeigt, allerdings erfordert das Auslesen und Komponieren der Antwort eine Bedienung des entsprechenden Tasks in der Hauptschleife.

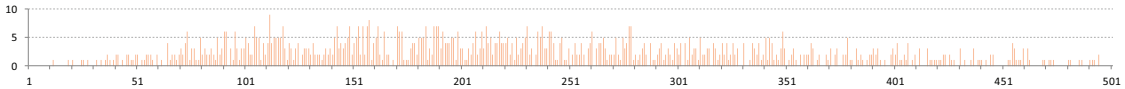
Zunächst scheint es so, daß mit steigender Anzahl an Teilpaketen so auch die Wahrscheinlichkeit, daß der entsprechende Task im richtigen Augenblick zeitnah ausgeführt wird, sinkt und letztlich die Antwortzeit ansteigt. Es handelt sich um eine bewußte Entwicklerentscheidung: das Benutzerinterface am Gerät erhält schwerpunktartig die CPU-Ressourcen zugeteilt, da Benutzereingaben höher priorisiert werden. Wäre eine geringere Netzwerklatenz erforderlich, so kann die Verarbeitung auch innerhalb der ISR des Funkmoduls durchgeführt werden, was aber andere Tasks beeinträchtigte.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.

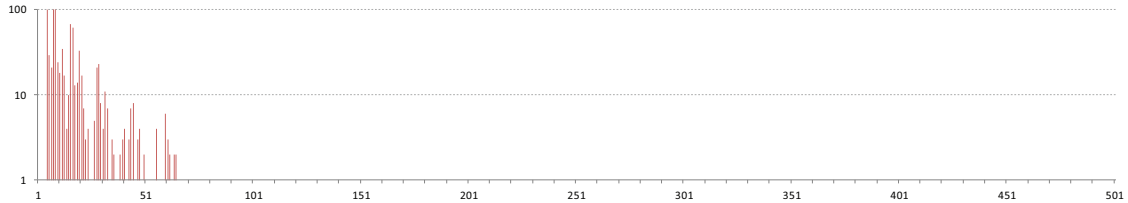


(c) 136 B Framegröße, 5 RFM-Pakete.

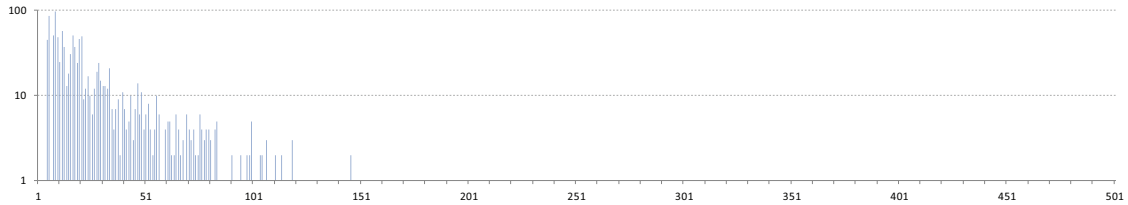
Abbildung 5.1: Antragen der Häufigkeiten der am Internet-Gateway gemessenen Antwortzeiten des *Ventilators* bei ungünstiger Ausrichtung und unterschiedlichen Paketgrößen. Antwortzeiten über 500 ms werden nicht dargestellt.

	RFM70		ICMP-Pakete			Ping-Zeiten		
	Größe	Pakete	gesendet	verloren	Verlust	*min*()	*max*()	*avg*()
Ventilator	46 B	2	1200	18	1,5 %	3,7 ms	604,6 ms	90,9 ms
Ventilator	76 B	3	1200	40	3,33 %	7,9 ms	592,2 ms	135,4 ms
Ventilator	136 B	5	1200	39	3,25 %	12,4 ms	834,0 ms	233,7 ms
LED-Gerät	46 B	2	1200	4	0,33 %	3,5 ms	138,3 ms	12,9 ms
LED-Gerät	76 B	3	1200	3	0,25 %	5,1 ms	237,1 ms	27,9 ms
LED-Gerät	136 B	5	1200	1	0,08 %	8,4 ms	353,0 ms	35,3 ms

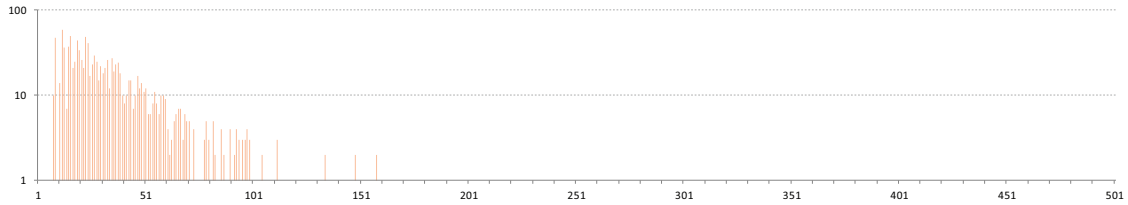
Tabelle 5.3: Die Tabelle zeigt die ermittelten Antwortzeiten der beiden Smart Devices bei zum Sender ungünstiger Ausrichtung, gemessen am Router. Es wurden je 1200 ICMP-Pakete mit unterschiedlicher Größe im Abstand von 1 s versendet. Beide Geräte hatten einen Abstand von ~1 m zum Sender bei direkter Sichtverbindung.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.



(c) 136 B Framegröße, 5 RFM-Pakete.

Abbildung 5.2: Antragen der Häufigkeiten der am Internet-Gateway gemessenen Antwortzeiten des *LED-Controllers* bei schlechter Positionierung und unterschiedlichen Paketgrößen. Wegen deutlich höherer Werte wurde für die Y-Achse eine logarithmische Skalierung zur Basis 10 gewählt.

Abbildung 5.2 zeigt die Daten der gleichen Meßreihe für den LED-Controller. Da der LED-Controller wie in Kapitel 4.3 beschrieben kein Benutzerinterface besitzt und seine Peripherie unabhängig von der CPU arbeitet, ist nur ein Netzwerktask in der Hauptschleife definiert.

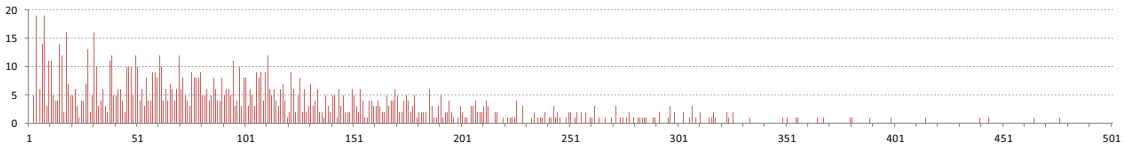
Auch hier scheint es, als spiegelten die abgebildeten Antwortzeiten diesen Sachverhalt teilweise wider, denn nahezu unabhängig von der Framegröße liegt eine Antwort in viel kürzerer mittlerer Zeit vor als beim Ventilatorgerät.

5.2.2.2 Ergebnisse bei ungünstiger Ausrichtung, am Windows-PC gemessen

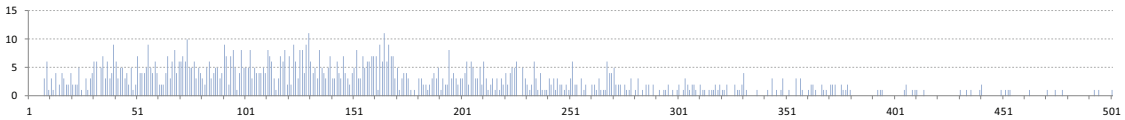
Die gleiche Serie wurde von einem über Ethernet an den Internet-Gateway angeschlossenen Windows-Host aus durchgeführt, um den Einfluß des Betriebssystems und der Tasks auf dem Internet-Gateway zu bewerten.

Da das in Windows integrierte `ping`-Werkzeug keine Genauigkeit im Mikrosekundenbereich bietet, wurde das Werkzeug `hrping` [111] verwendet. Die über Ethernet eingeführte Latenz wird vernachlässigt, da ein Ping des Internetrouters vom Windows-Host aus mit konstanten Werten um ~0,5 ms stets um eine Größenordnung geringer ausfiel als die Reaktionszeit des Smart Devices.

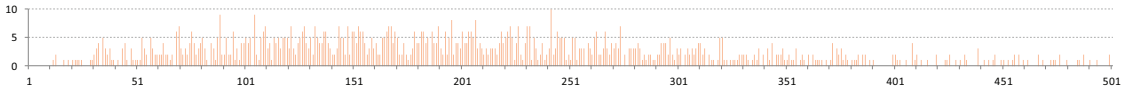
Abbildung 5.3 zeigt die Meßreihe mit dem Ventilator, Abbildung 5.4 stellt die gleichen Daten mit dem LED-Controller dar. Wie auch aus Tabelle 5.4 zu entnehmen ist, sind die Unterschiede zwischen den Meßreihen am Windows-PC und am Internet-Gateway direkt statistisch nicht signifikant. Der Einfluß von Betriebssystem auf PC und Internet-Gateway sowie die laufenden Tasks auf den Geräten kann daher vernachlässigt werden.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.

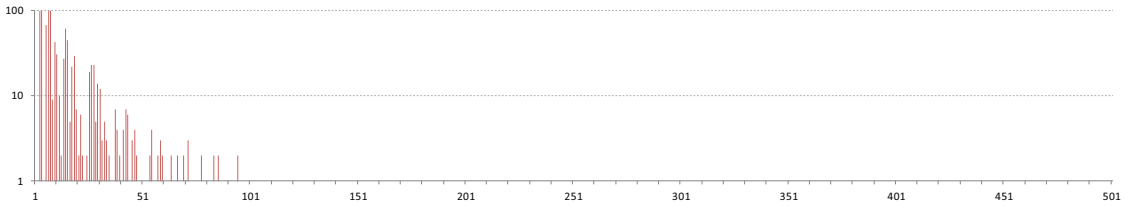


(c) 136 B Framegröße, 5 RFM-Pakete.

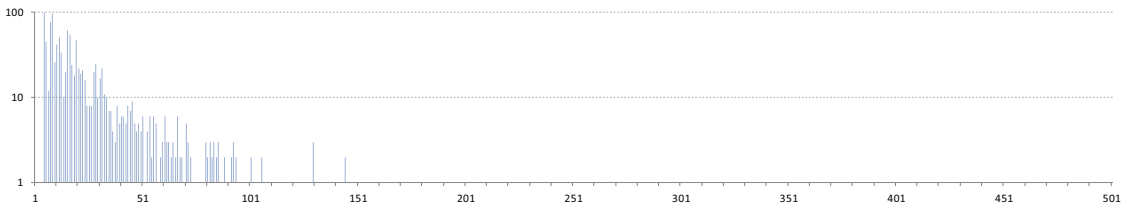
Abbildung 5.3: Antragen der Häufigkeiten der am Windows-PC gemessenen Antwortzeiten des *Ventilators* bei ungünstiger Ausrichtung und unterschiedlichen Paketgrößen. Antwortzeiten über 500 ms werden nicht dargestellt.

	RFM70		ICMP-Pakete			Ping-Zeiten		
	Größe	Pakete	gesendet	verloren	Verlust	*min*()	*max*()	*avg*()
Ventilator	46 B	2	1200	7	0,58 %	3,1 ms	529,3 ms	104,7 ms
Ventilator	76 B	3	1200	14	1,17 %	7,9 ms	745,9 ms	155,3 ms
Ventilator	136 B	5	1200	8	0,67 %	12,4 ms	741,0 ms	209,7 ms
LED-Gerät	46 B	2	1200	5	0,42 %	3,2 ms	115,2 ms	13,6 ms
LED-Gerät	76 B	3	1200	3	0,25 %	4,9 ms	235,1 ms	24,1 ms
LED-Gerät	136 B	5	1200	1	0,08 %	8,2 ms	285,6 ms	35,9 ms

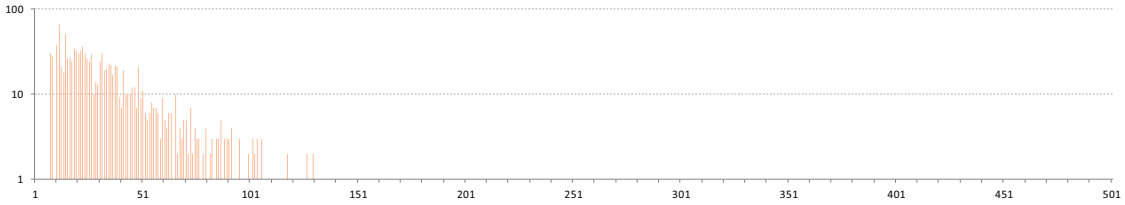
Tabelle 5.4: Die Tabelle zeigt die ermittelten Antwortzeiten der beiden Smart Devices bei zum Sender ungünstiger Ausrichtung, gemessen am Windows-PC mit `hrping`. Es wurden dieselben Parameter wie in der vorigen Meßreihe verwendet. Die Geräte befanden sich in exakt gleicher Position. Der deutliche Unterschied zwischen den Meßreihen des Ventilators ist daher auf äußere Einflüsse zurückzuführen.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.

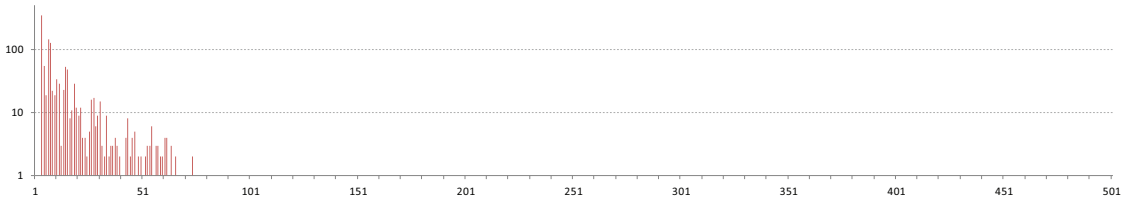


(c) 136 B Framegröße, 5 RFM-Pakete.

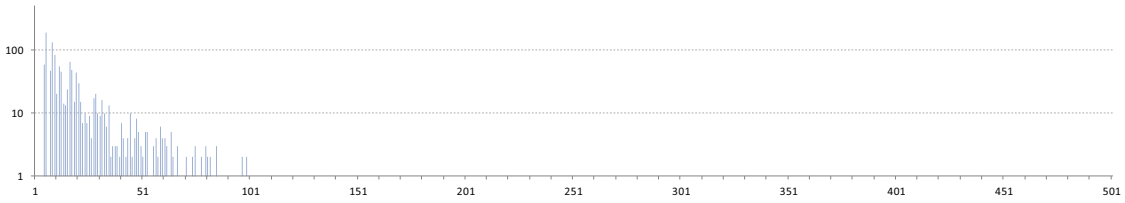
Abbildung 5.4: Antragen der Häufigkeiten der am Windows-PC gemessenen Antwortzeiten des *LED-Controllers* bei ungünstiger Ausrichtung und unterschiedlichen Paketgrößen. Wegen deutlich höherer Werte wurde für die Y-Achse eine logarithmische Skalierung zur Basis 10 gewählt.

5.2.2.3 Ergebnisse bei günstiger Ausrichtung, am Internet-Gateway gemessen

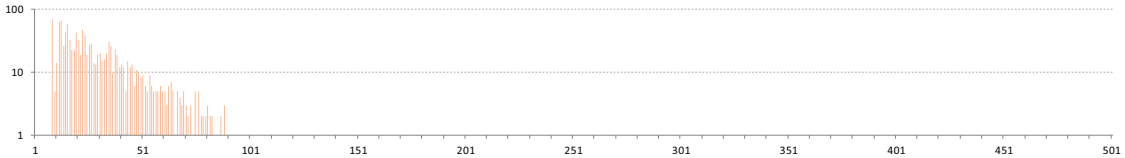
Zum Vergleich wurden beide Geräte in eine günstige Lage zum Sender hin ausgerichtet.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.



(c) 136 B Framegröße, 5 RFM-Pakete.

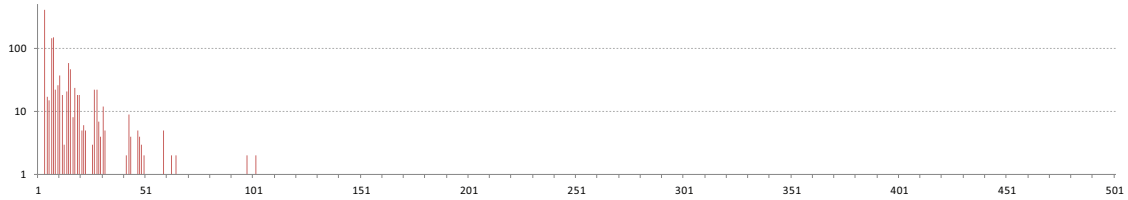
Abbildung 5.5: Antragen der Häufigkeiten der am Internet-Gateway gemessenen Antwortzeiten des *Ventilators* bei günstiger Ausrichtung und unterschiedlichen Paketgrößen.

In Abbildung 5.5 sind die sich nun ergebenden Ping-Häufigkeiten des Ventilators angetragen; Abbildung 5.6 zeigt die selben Daten für den LED-Controller. Die Statistiken der gemessenen Pakete sind in Tabelle 5.5 aufgeführt.

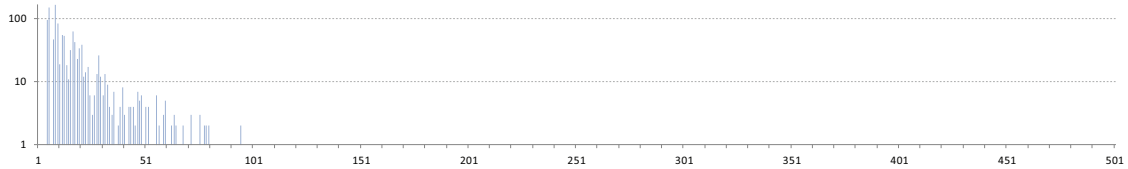
Der LED-Controller konnte sich sowohl in der Verteilung wie auch in der durchschnitt-

	RFM70		ICMP-Pakete			Ping-Zeiten		
	Größe	Pakete	gesendet	verloren	Verlust	*min*()	*max*()	*avg*()
Ventilator	46 B	2	1200	2	0,17 %	3,6 ms	218,7 ms	14,3 ms
Ventilator	76 B	3	1200	1	0,08 %	5,2 ms	196,9 ms	21,3 ms
Ventilator	136 B	5	1200	7	0,58 %	8,6 ms	453,1 ms	31,7 ms
LED-Gerät	46 B	2	1200	1	0,08 %	3,5 ms	166,1 ms	12,9 ms
LED-Gerät	76 B	3	1200	3	0,25 %	5,1 ms	474,3 ms	18,7 ms
LED-Gerät	136 B	5	1200	16	1,33 %	8,4 ms	637,7 ms	34,3 ms

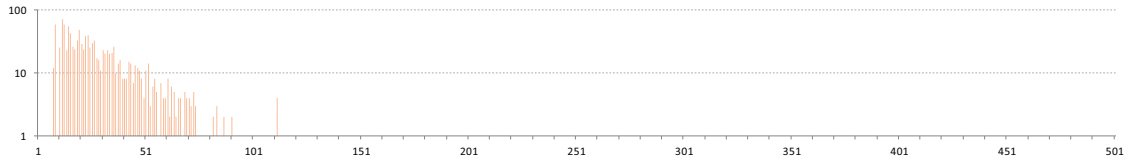
Tabelle 5.5: Die Tabelle zeigt die ermittelten Antwortzeiten der beiden Smart Devices bei zum Sender günstiger Ausrichtung, gemessen am Internet-Gateway.



(a) 46 B Framegröße, 2 RFM-Pakete.



(b) 76 B Framegröße, 3 RFM-Pakete.



(c) 136 B Framegröße, 5 RFM-Pakete.

Abbildung 5.6: Antragen der Häufigkeiten der am Internet-Gateway gemessenen Antwortzeiten des *LED-Controllers* bei günstiger Positionierung und unterschiedlichen Paketgrößen.

lichen Antwortzeit noch einmal geringfügig gegenüber seiner ungünstigen Ausrichtung verbessern.

Deutlich erkennbar dagegen ist die nun um eine Größenordnung geringere durchschnittliche Antwortzeit des Ventilators. Im direkten Vergleich zum LED-Controller sinkt der Abstand auf nur wenige Millisekunden, im Gegensatz zu den vorher ermittelten Werten. Dies ergibt sich aus einer weitaus geringeren Verlustrate der zugrundeliegenden RFM70-Pakete.

Die Aussage aus Kapitel 5.2.2.1, daß die hohe Latenz den Tasks der Benutzerschnittstelle geschuldet ist, muß daher noch einmal präzisiert werden.

Richtig ist, daß jedes einzelne RFM-Teilpaket auf einen Interrupt hin vom RFM-Modul ausgelesen wird. Eine Verzögerung ergibt sich, wenn einer der in Kapitel 4.4 beschriebenen Timer überläuft und den Verlust eines RFM-Paketes anzeigt, da der Task, der diese Software-Timer bedient, auch innerhalb der Hauptschleife ausgeführt wird. Bei einer hohen Verlustrate an RFM70-Paketen ist dessen Ausführung innerhalb der Hauptschleife daher öfter erforderlich. Eine logische Konsequenz wäre, daß die vorhandene Implementierung bestehend aus Firmware und Hardware mit einer bestimmten Taktfrequenz diese Ausführung nicht schnell genug abarbeiten kann und deswegen RFM-Pakete endgültig durch Verschulden des Smart Devices verlorengingen.

Ein Profiling der betroffenen Routinen der Firmware ergab allerdings eine andere Ursache: offenbar läßt sich der Ventilator aufgrund elektrischer Eigenschaften der Platine und des

Aufbaus deutlich ungünstiger zum Sender ausrichten, was sich in einer deutlich erhöhten Paketverlustrate auf dem Funkkanal niederschlägt. Der Einfluß der größeren Anzahl an Tasks, einschließlich Tasten und LCD-Routinen, innerhalb der Hauptschleife kann dagegen vernachlässigt werden.

Diese Erkenntnis wird auch von Ergebnissen des späteren Kapitels 5.2.3 gestützt.

5.2.2.4 Ergebnisse bei günstiger Ausrichtung, am Windows-PC gemessen

Da sich zu den Abbildungen 5.5 und 5.6 vergleichbare Graphen ergeben, ist nur die Statistik der Antwortzeiten in Tabelle 5.6 gegeben. Die Meßreihe wurde wiederum mit `hrping` vom Windows-PC aus durchgeführt und ihre Ergebnisse statistisch ausgewertet.

	RFM70		**ICMP-Pakete**			**Ping-Zeiten**		
	Größe	Pakete	gesendet	verloren	Verlust	*min*()	*max*()	*avg*()
Ventilator	46 B	2	1200	2	0,17 %	3,3 ms	204,1 ms	13,4 ms
Ventilator	76 B	3	1200	1	0,08 %	5,0 ms	213,4 ms	19,9 ms
Ventilator	136 B	5	1200	5	0,42 %	8,1 ms	332,8 ms	33,6 ms
LED-Gerät	46 B	2	1200	3	0,25 %	3,1 ms	122,9 ms	12,7 ms
LED-Gerät	76 B	3	1200	1	0,08 %	5,0 ms	348,2 ms	18,7 ms
LED-Gerät	136 B	5	1200	15	1,25 %	7,9 ms	375,4 ms	29,9 ms

Tabelle 5.6: Die Tabelle zeigt die ermittelten Antwortzeiten der beiden Smart Devices bei zum Sender günstiger Ausrichtung, gemessen am Windows-PC mit `hrping`. Es wurden je 1200 ICMP-Pakete mit unterschiedlicher Größe im Abstand von 1 s versendet. Beiden Geräte hatten einen Abstand von ~1 m zum Sender bei direkter Sichtverbindung.

Es ergibt sich kein signifikanter Unterschied zu den Ergebnissen der Messungen am Internet-Gateway.

5.2.2.5 Schlußfolgerungen und Einschätzung

Mit den vorliegenden Meßreihen können die folgenden Schlüsse gezogen werden:

- *Internet-Gateway.* Der Internetrouter besitzt keinen signifikanten Einfluß auf die Reaktionszeiten der Smart Devices.
- *Ausrichtung der Geräte.* Die Lage der Geräte und insbesondere die Ausrichtungen der Antennen zueinander hat definitiv großen Einfluß auf die Verlustwahrscheinlichkeit der RFM-Pakete und damit die Kapazität des Funkkanals. Im Datenblatt ist keine Abstrahlcharakteristik der PCB-Antenne angegeben, daher kann eine optimale Position nur durch Probieren ermittelt werden.
- *Antennendesign.* Die aufgedruckte PCB-Antenne der RFM70-Module besitzt eine ausgesprochene Richtcharakteristik und ist für Anwendungen, die eine Neuausrichtung der Antenne nicht ermöglichen, nicht geeignet.

- *Reaktionszeiten der Geräte.* Bei optimaler Lage verhalten sich die vorliegenden Geräte mit Reaktionszeiten von wenigen Millisekunden vergleichbar performant wie gängige WiFi-Teilnehmer betreffend ihrer Antwortzeiten.

Bemerkenswert ist insbesondere das Ergebnis der geringen Verlustrate von *Ethernetframes* auf der proprietären Funkstrecke insgesamt. Sie liegt durchweg in Bereichen um unter 1 %. Das Ziel, ein verlustfreies Medium auf schlechter Funkadaption zu emulieren, wird daher als erreicht angesehen. In Kapitel 5.2.4 wird diese Aussage auch theoretisch qualifiziert.

Die Funkstrecke könnte signifikant verbessert werden, wenn direkt auf dem Modul Maßnahmen wie Fehlerkorrektur (Forward Error Correction (FEC)), eine passendere Modulation und Demodulation mit Matched-Filter und höherer Signal-to-Noise-Ratio (SNR) insgesamt, sowie eine geeignete Antenne mit Rundstrahlcharakteristik verwendet werden.

Würden gleichzeitig Absenderadressen eingeführt, könnte die Behandlung von Neuübertragungen zumindest bei Unicast-Kommunikation vollständig vom Funkmodul übernommen werden. Ob eine Implementierung eines zuverlässigen, NACK-basierten Broadcast-Algorithmus sinnvoll ist, kann an dieser Stelle nicht beantwortet werden, da andere integrierte Funktechniken bereits Mechanismen zum Verteilen von Nutzlast per kooperativem Multicast implementieren.

5.2.3 Performanz der Funkverbindung auf OSI-Anwendungsschicht

Da eine direkte Messung der Latenzzeit nur wenig Aussagekraft für die Performanz und insgesamte Reaktionszeit auf Anwendungsebene besitzt, werden für zwei Smart Devices auch die jeweiligen Zeiten gemessen, die Unicast-Anfragen nach bestimmten Diensten zur vollständigen Übertragung der XML-Antwort benötigen:

- Discovery-Ausgabe: `GET /`

	URL		HTTP-Downloadzeiten		
	Pfad	Größe	*min*()	*max*()	*avg*()
Ventilator	`/`	1358 B	77,9 ms	5251,0 ms	226,1 ms
Ventilator	`/ventilator_4`	360 B	41,6 ms	1749,7 ms	121,2 ms
Ventilator	`/ventilator_4/Ventilator`	165 B	30,7 ms	272,8 ms	98,0 ms
Ventilator	`/ventilator_4/Ventilator/Drehzahl`	113 B	29,6 ms	1746,7 ms	97,1 ms
LED-Gerät	`/`	4542 B	379,6 ms	3097,8 ms	631,7 ms
LED-Gerät	`/led_controller_1`	1348 B	109,7 ms	1992,8 ms	246,4 ms
LED-Gerät	`/led_controller_1/led1`	343 B	42,6 ms	2302,6 ms	142,7 ms
LED-Gerät	`/led_controller_1/led1/red`	118 B	34,7 ms	1950,8 ms	108,1 ms

Tabelle 5.7: HTTP-Antwortzeiten am Router beider Devices bei kontinuierlicher Abfrage mittels `httping`. Es wurden 1200 `GET`-Anfragen an den jeweiligen URL-Pfad gerichtet und die vollständige Download-Zeit gemessen.

- Dienste und Datenpunkte: `GET /ventilator_4, /led_controller_1`
- Ein Dienst: `GET /ventilator_4/Ventilator, /led_controller_1/led1`
- Ein Datenpunkt: `GET /ventilator_4/Ventilator/Drehzahl, /led_controller_1/led1/red`

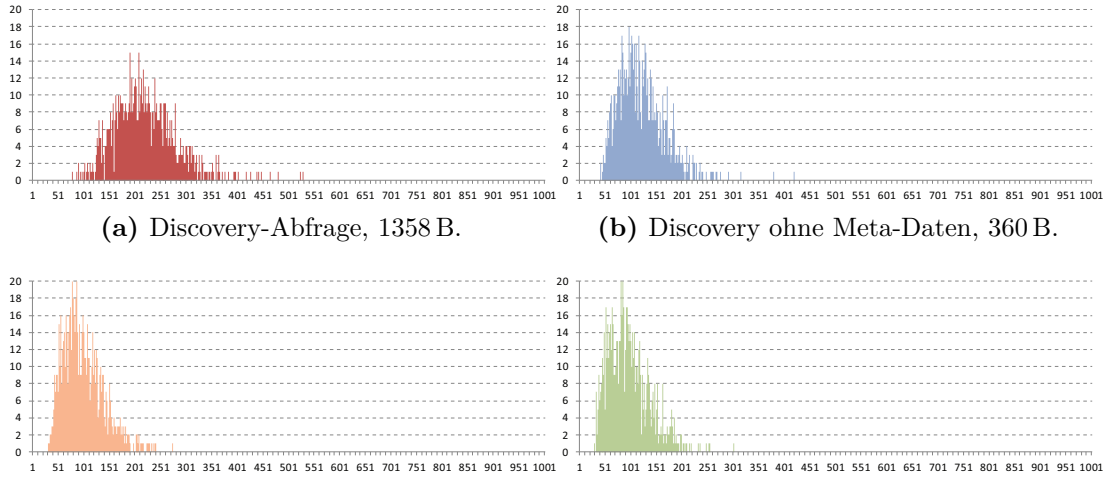
Repräsentativ werden für die Messung wieder die beiden vorgenannten Geräte verwendet, eines mit LCD (Ventilator) und eines ohne (LED-Controller). Die Geräte befinden sich für die Meßreihen in optimaler Ausrichtung zum USB-Dongle.

Zur Ermittlung der entsprechenden Zeiten wird auf dem Internetrouter das Programm `httping` [112] installiert. Es akzeptiert verschiedene Kommandozeilenparameter, mit denen jeweils die URLs der obigen Liste als Ziel festgelegt und die Anzahl der Anfragen auf 1200 pro URL begrenzt wird.

Das Werkzeug gibt eine mit dem regulären `ping` vergleichbare Ausgabe auf der Kommandozeile aus. Für die Verbindung mit dem Host werden allerdings keine ICMP-Pakete verwendet, sondern TCP-Verbindungen. Ein Verlust einer gesamten Anfrage ist daher selbst bei Verlust von zugrundeliegenden Ethernet-Frames durch das Funkmedium sehr unwahrscheinlich und kam in der gesamten Meßreihe nicht vor.

Die Ergebnisse sind in den Abbildungen 5.7 und 5.8 sowie Tabelle 5.7 dargestellt.

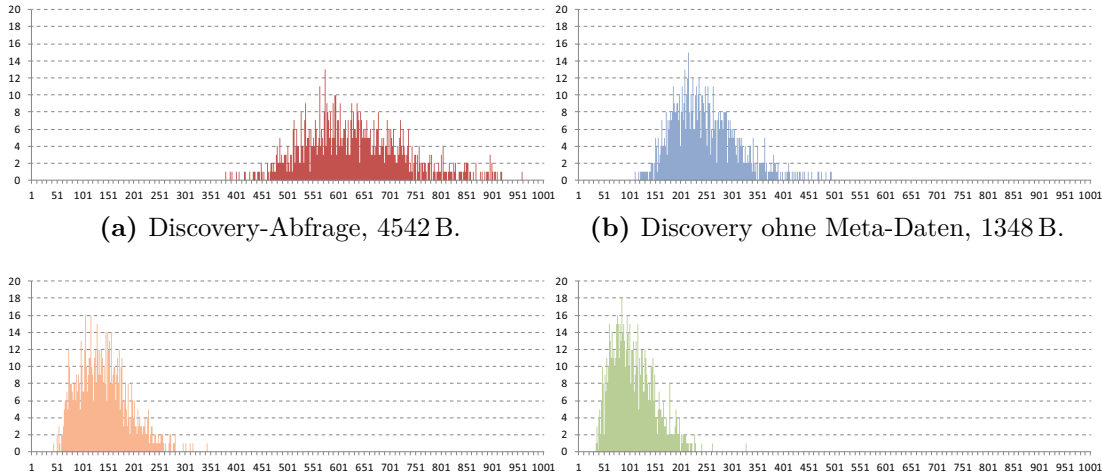
Zunächst ist erwartungsgemäß eine deutliche Abnahme der durchschnittlichen Download-Zeit von XML-Beschreibungen in Abhängigkeit der Größe der angeforderten Information erkennbar. Die Discovery-Anfragen benötigen im Mittel die meiste Zeit, müssen aber auch am wenigsten oft durchgeführt werden.



(a) Discovery-Abfrage, 1358 B. **(b)** Discovery ohne Meta-Daten, 360 B.

(c) Abfrage eines Services, 165 B. **(d)** Abfrage eines Datenpunktes, 113 B.

Abbildung 5.7: Abbildungen der Häufigkeiten der HTTP-Antwortzeiten des *Ventilators* bei Aufruf unterschiedlicher Dienste. Es wurde jeweils die Zeit gemessen, die zum vollständige Download der XML-Beschreibung über eine TCP-Verbindung im Unicast-Modus erforderlich war.



(a) Discovery-Abfrage, 4542 B.

(b) Discovery ohne Meta-Daten, 1348 B.

(c) Abfrage eines Services, 343 B.

(d) Abfrage eines Datenpunktes, 118 B.

Abbildung 5.8: Abbildungen der Häufigkeiten der HTTP-Antwortzeiten des *LED-Controllers* bei Aufruf unterschiedlicher Dienste. Es wurde jeweils die Zeit gemessen, die zum vollständige Download der XML-Beschreibung über eine TCP-Verbindung im Unicast-Modus erforderlich war.

Der wahrscheinlichere Fall des Abrufens einer einzelnen Information, sprich „Datenpunkt“, benötigt auf beiden Geräten nur um die 100 ms in der vorliegenden Implementierung. Die Abfragerate kann damit theoretisch bis zu 10 Hz für einen singulären Datenpunkt betragen, wobei das Medium in der Zeit natürlich belegt und für andere Anfragen blockiert ist.

Auch verhalten sich beide Geräte auf Anwendungsebene nahezu identisch bei gleichem Übertragungsvolumen, was sich aus dem Vergleich der Abbildungen 5.7a und 5.8b ergibt.

Überraschend sind allerdings zwei Ergebnisse:

1. Zum einen antwortet der LED-Controller trotz leicht geringerer Ping-Zeiten um Größenordnungen *langsamer* auf Discovery-Anfragen als der Ventilator. Sehr deutlich ist dies im direkten Vergleich der Abbildungen 5.7a und 5.8a zu erkennen. Der Sachverhalt ist allein der größeren Nutzdatenlänge geschuldet; er verhält sich annähernd linear (in diesem Falle ungefähr um den Faktor 3).

2. Eine untere Grenze der durchschnittlichen Antwortzeit scheint sich bei ungefähr 100 ms für kleine Nutzdatenlängen bis 150 B einzupendeln; sie ist von der Anzahl der übertragenen RFM-Pakete offenbar nicht abhängig (siehe Tabelle 5.7, untere Zeilen des Ventilators). Offenbar ist hiermit die minimale Zeit des TCP-Handshakes bei vorliegendem Nachrichtentransport erreicht.

Abschließend ist anzumerken, daß die Geräte, sofern eine Sollgröße bzw. ein Datenpunkt geändert wird, im Allgemeinen schneller reagieren als die XML-Antwort beim Anfragenden eingetroffen ist: ein Befehl wird ausgeführt, bevor das Zusammenstellen und Versenden

der XML-Antwort begonnen hat. In der Regel ist mit „bloßem Auge“ keine Verzögerung zwischen Absenden des Befehls und Reaktion des Geräts feststellbar, auf eine Messung der Reaktionszeit am Gerät wurde verzichtet.

5.2.4 Wahrscheinlichkeit einer fehlgeschlagenen Übertragung

In diesem Unterkapitel wird untersucht, wie groß die Wahrscheinlichkeit für eine fehlgeschlagene Übertragung eines gesamten Ethernet-Frames ist. Die Anzahl der Wiederholungen eines Teilpakets wird erst im nächsten Kapitel untersucht.

5.2.4.1 Unicast-Modus

Da im Unicast-Modus nur ein einziger Empfänger beteiligt und ein statisches Prozedere aus Sendung und ACK eingerichtet ist, läßt sich ein einfaches mathematisches Modell für die Übertragung erstellen. Die Wahrscheinlichkeit, daß ein Ethernet-Frame final nach einer bestimmten Zahl an Übertragungen von RFM-Teilpaketen verloren ist, ist nur von der Fehlerwahrscheinlichkeit des einzelnen RFM-Pakets sowie der Framegröße insgesamt abhängig. Im Folgenden werden die theoretischen Beschreibungen entwickelt.

Die Anzahl n_{RFM} der RFM-Pakete, die zur Übertragung einer beliebigen Nutzlast der Größe $size_{Payload}$ in Byte erforderlich ist, wird über (5.1) ermittelt.

$$n_{RFM}(size_{Payload}) = \begin{cases} 1, & \text{für } size_{Payload} < 30 \\ \left\lceil \frac{size_{Payload}-29}{30} \right\rceil + 1, & \text{für } size_{Payload} \geq 30 \end{cases} \tag{5.1}$$

Da die Mindestgröße[1] $size_{ETH}$ eines Ethernet-Frames der vorangegangenen Meßreihen mit IPv4 und UDP 42 Byte – ohne Nutzdaten – beträgt, kann (5.1) zur Darstellung in (5.2) vereinfacht werden. Bei Verwendung von TCP belegt ein Frame mindestens 54 Byte, bei ICMP-Ping mit 4 Byte Daten liegt die Framegröße bei 46 Byte.

$$n_{RFM}(size_{ETH}) = \left\lceil \frac{size_{ETH} - 29}{30} \right\rceil + 1, \qquad \text{für } 46 \leq size_{ETH} \leq 1514 \tag{5.2}$$

Bei gegebener Fehlerwahrscheinlichkeit $P_{errChn} \in [0, 1]$ eines einzelnen RFM-Paketes in einer Übertragung kann die Wahrscheinlichkeit für ein insgesamt fehlgeschlagenes RFM-Paket $P_{errSP} \in [0, 1]$ nach (5.3) berechnet werden. Zu beachten ist, daß wegen des

1 Anmerkung: Auch bei Betrachtung von lediglich Ethernet- und IPv4-Header ($20 + 14 = 34$) ist die Grenze von 29 Byte schon überschritten und mindestens zwei RFM-Pakete sind erforderlich. Die Größe von 46 B bei ICMP wurde der besseren Vergleichbarkeit mit der vorigen Meßreihe halber gewählt.

obligatorischen ACK-Pakets die Übertragung zweifach erfolgreich ausgeführt werden muß, ansonsten gilt die Teilübertragung als fehlgeschlagen und wird wiederholt. Alle Übertragungen werden als statistisch unabhängig, sowie P_{errChn} als konstant angenommen.

$$\begin{aligned} P_{errSP} &= 1 - (1 - P_{errChn})^2 = 1 - (1 - 2\,P_{errChn} + {P_{errChn}}^2) \\ &= 2\,P_{errChn} - {P_{errChn}}^2 \end{aligned} \tag{5.3}$$

Die Anzahl der maximal ausgeführten Versuche einer Übertragung heißt $n_{maxTries}$. Damit ist die Wahrscheinlichkeit P_{errRFM} eines insgesamt verlorenen RFM-Pakets nach $n_{maxTries}$ versuchten Übertragungen nach (5.4) definiert. Die einzelnen Übertragungen werden als statistisch unabhängig voneinander betrachtet.

$$P_{errRFM} = {P_{errSP}}^{n_{maxTries}} = \left(2\,P_{errChn} - {P_{errChn}}^2\right)^{n_{maxTries}} \tag{5.4}$$

Um ein gesamtes Ethernet-Frame zu verlieren, reicht es aus, daß von allen RFM-Teilpaketen mindestens eines nicht korrekt empfangen wird. Mit anderen Worten besteht bei jedem Teilpaket die Chance, das Frame zu verlieren. Die Gesamtfehlerwahrscheinlichkeit P_{errETH} wird über die Wahrscheinlichkeit des Gegenereignisses nach (5.5) bestimmt.

$$\begin{aligned} P_{errETH}(size_{ETH}) &= 1 - \left((1 - P_{errRFM})^{n_{RFM}(size_{ETH})}\right) \\ &= 1 - \left(\left(1 - (2\,P_{errChn} - {P_{errChn}}^2)^{n_{maxTries}}\right)^{n_{RFM}(size_{ETH})}\right) \\ &= 1 - \left(\left(1 - (2\,P_{errChn} - {P_{errChn}}^2)^{n_{maxTries}}\right)^{\left(\left\lceil \frac{size_{ETH}-29}{30} \right\rceil + 1\right)}\right) \end{aligned} \tag{5.5}$$

Auf eine Herleitung über die Binomialverteilung, die durch Aufsummieren der Wahrscheinlichkeiten der günstigen Ereignisse zum gleichen Ergebnis käme, wird an dieser Stelle verzichtet.

Mit den Grenzen der Größe des Ethernet-Frames erhält man:

$$P_{errETH}(46) = 1 - \left(1 - (2\,P_{errChn} - {P_{errChn}}^2)^{n_{maxTries}}\right)^2 \tag{5.6a}$$

$$P_{errETH}(1514) = 1 - \left(1 - (2\,P_{errChn} - {P_{errChn}}^2)^{n_{maxTries}}\right)^{51} \tag{5.6b}$$

Auf die Fehlerwahrscheinlichkeit des Kanals hat man im Allgemeinen keinen Einfluß.

Deswegen ist von Interesse, wie zum einen die Verlustwahrscheinlichkeit des gesamten Frames von der maximalen Anzahl der Wiederholungen abhängt. Zum anderen ist zu prüfen, ob die eingangs willkürlich festgelegte Anzahl $n_{maxTries} = 25$ hinreichend gut gewählt ist:

1. Abbildungen 5.9a und 5.9b zeigen entsprechende Kurvenscharen, bei denen die Anzahl $n_{maxTries}$ vier unterschiedliche Werte annimmt. Wie insbesondere aus der zweiten Abbildung erkennbar ist, können mit der vorhandenen Implementierung auch vollständig besetzte Ethernet-Frames bei Kanalfehlerwahrscheinlichkeiten von 50 % noch nahezu fehlerfrei übertragen werden. Die Anforderung, ein zuverlässiges Medium zu emulieren, wird als daher erfüllt betrachtet.

2. Die im vorigen Kapitel gemessenen Verlustraten der Ethernet-Frames liegen zwischen 0,08 % und 3,33 % bei schlechter Positionierung der Geräte, vergleiche Tabelle 5.3. Bei bis zu 25 Wiederholungen läßt sich damit auf eine maximale Verlustrate von etwa 60 % der RFM-Pakete schließen, vergleiche auch Abbildung 5.9a. Die Anforderung, eine möglichste schlechte Implementierung eines Funkmediums zu verwenden, wird damit ebenfalls als erfüllt angesehen.

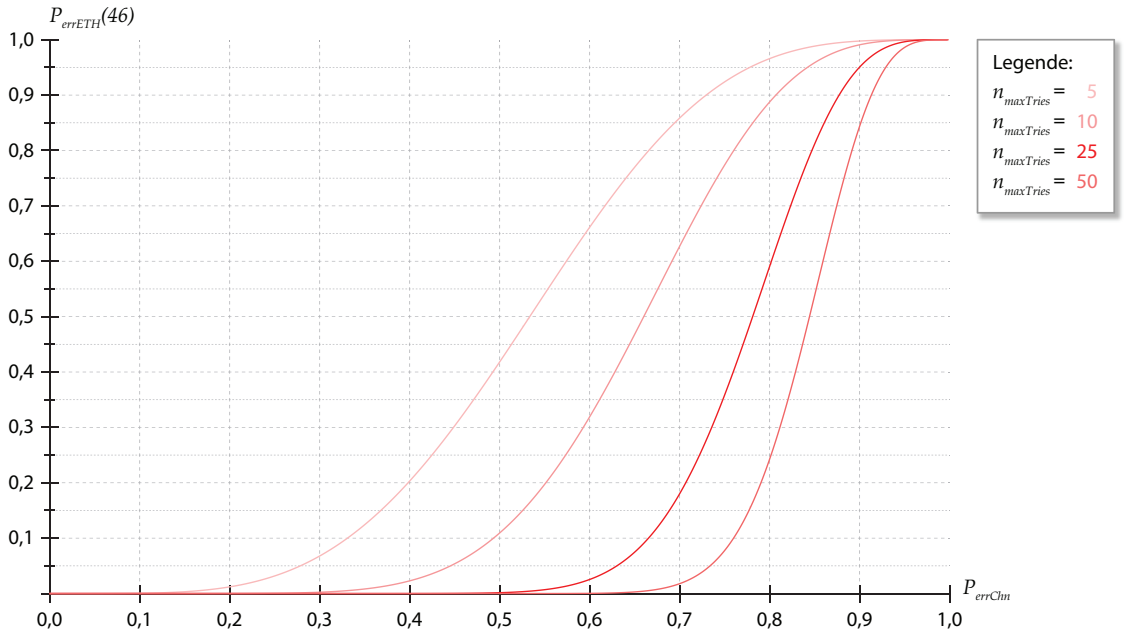
5.2.4.2 Broadcast-Modus

In diesem Übertragungsmodus kommt ein NACK-basierter Mechanismus zum Tragen, bei welchem fehlerhafte RFM-Teilpakete frühestens mit dem nächsten Teilpaket anhand der Sequenznummer von Sender erkannt werden. In dem Fall fordert ein Empfänger das RFM-Teilpaket erneut über ein NACK-RFM-Paket an, wodurch erschwerend in der Modellierung auch der Verlust dieses NACKs ab dem zweiten RFM-Teilpaket berücksichtigt werden muß.

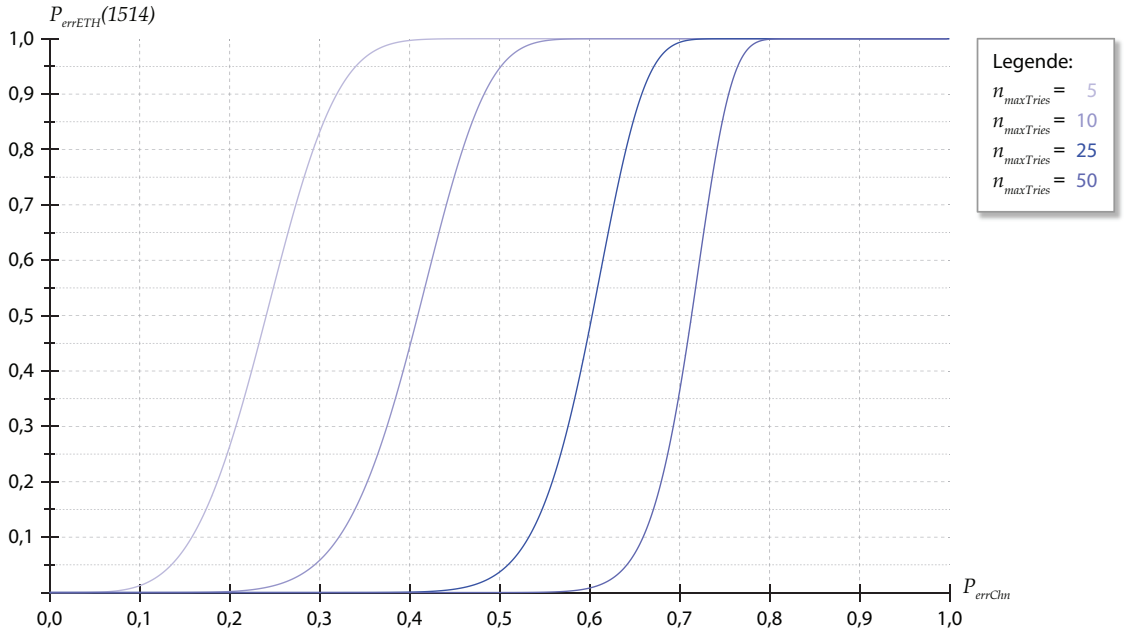
Die Überlegungen des vorigen Unterkapitels 5.2.4.1 werden auf den Broadcast-Modus erweitert, um die Verlustwahrscheinlichkeit in Abhängigkeit der Framegröße $size_{ETH}$ (und damit der Anzahl der RFM-Teilpakete) und der Anzahl k der verbundenen Geräte angeben zu können. Wie im vorigen Unterkapitel wird auch in diesem Modus von einer konstanten Kanalfehlerwahrscheinlichkeit P_{errChn} ausgegangen.

Die Übertragungen bzw. die Verlustwahrscheinlichkeit eines RFM-Teilpakets hingegen wird hier nicht mehr als unabhängig betrachtet, da bei mehreren Geräten mit jedem verlorenen RFM-Teilpaket eine gleiche Chance auf ein NACK-Paket besteht. Davon reicht ein einziges aus, um dem gesamten Teilpaket oder ggf. einigen vorigen Teilpaketen eine erneute Chance auf eine günstige Übertragung zu ermöglichen.

Im Gegensatz zum Unicast-Modus wird pro Teilpaket-"Schritt" allerdings nur ein einziges NACK vom Empfänger entweder für das aktuelle Teilpaket bei Verlust und nachfolgendem Timeout, oder eben bei korrektem Empfang eines unerwartet späteren Teilpakets gesendet. Geht das NACK aller Geräte in der letzten Teilübertragung verloren, so ist das Ethernet-Frame für diese Geräte verloren. Sind dagegen noch weitere Teilübertragungen geplant, so kann der Empfänger nach dem nächsten korrekt empfangenen Teilpaket erneut ein



(a) Framegröße bei 46 B; Plot der Funktion (5.6a).



(b) Framegröße bei 1514 B; Plot der Funktion (5.6b).

Abbildung 5.9: Verlustwahrscheinlichkeiten verschieden großer Ethernet-Frames bei unterschiedlicher maximaler Wiederholung der Übertragung.

NACK absetzen.

Es wird zunächst die Frage nach der Verlustwahrscheinlichkeit des Ethernet-Frames P_{errRBP} an einem einzigen Gerät, $k = 1$, untersucht. Für den einfachsten Fall, $n_{RFM} = 2$, läßt sich noch ein übersichtliches Zustandsübergangsdiagramm nach Abbildung 5.10 konstruieren.

Mit Hilfe des Zustandsgraphen wird eine geschlossene Darstellung für P_{errRBP} angegeben, indem die möglichen Pfade in den einzigen günstigen Zustand `RX[1]` sequentiell aufaddiert und Schleifen als Summen für die mögliche Anzahl an Iterationen geschrieben werden.

Das Resultat ist in (5.7) mit $P_E \equiv P_{errChn}$ und $P_{\neg E} \equiv 1 - P_{errChn}$ dargestellt. Jede Schleife wird maximal $n_{maxTries} - 1 = 24$ mal durchlaufen.

$$
\begin{aligned}
P_{errRBP} = 1 - \Bigg[\quad & {P_{\neg E}}^2 \quad + \quad {P_{\neg E}}^3 \cdot P_E \cdot \sum_{i=0}^{24} \left(P_{\neg E} \cdot P_E\right)^i \quad + \\
& P_E \cdot {P_{\neg E}}^4 \cdot \sum_{j=0}^{24} \left({P_{\neg E}}^2 \cdot P_E\right)^j \quad + \\
& {P_E}^2 \cdot {P_{\neg E}}^5 \cdot \sum_{j=0}^{24} \left({P_{\neg E}}^2 \cdot P_E\right)^j \cdot \sum_{i=0}^{24} \left(P_{\neg E} \cdot P_E\right)^i \quad \Bigg]
\end{aligned} \tag{5.7}
$$

Wie der Baum aus Abbildung 5.10 und (5.7) bereits andeuten, kann der Algorithmus insbesondere bei steigernder Anzahl von RFM-Paketen aufgrund der voneinander abhängigen Übertragungen mathematisch nicht mehr einfach geschlossen ausgedrückt werden.

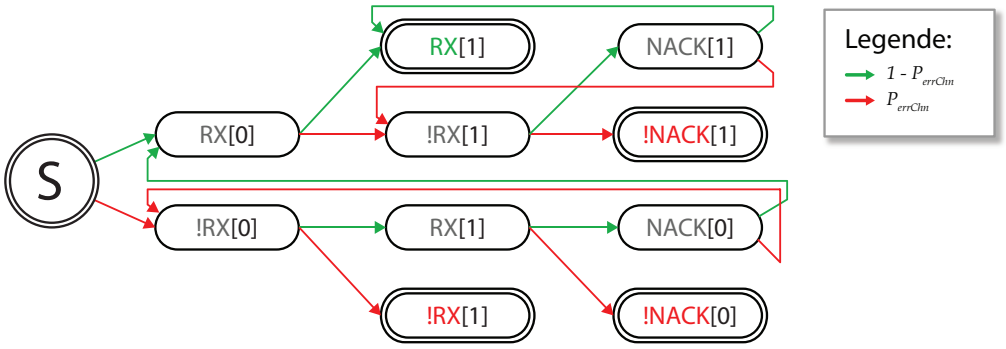


Abbildung 5.10: Zustandsübergangsdiagramm mit eingezeichneten Übergangswahrscheinlichkeiten des einfachsten Falls des implementierten Broadcast-Protokolls: Ein Gerät empfängt ein Ethernet-Frame, das aus zwei RFM-Teilpaketen besteht. Die maximale Anzahl der Wiederholungen (Schleifen) $n_{maxTries}$ eines Teilpakets ist auf 25 begrenzt, allerdings im Diagramm nicht dargestellt. Die Notation `RX` bedeutet ein erfolgreich empfangenes Teilpaket des Indexes in eckigen Klammern aus Sicht des Empfängers, `!RX` bezeichnet einen Verlust. Gleiches gilt für die NACK-Pakete, allerdings aus Sicht des Senders.

Daher wurde stattdessen eine Simulation programmiert, mit Hilfe derer über die Fehlerwahrscheinlichkeit P_{errChn} von 0 bis 1 in Schritten von 0,1 % iteriert und die sich ergebende Verlustwahrscheinlichkeit P_{errETH} aus insgesamt $n_{Versuche} = 100\,000$ simulierten Übertragungen ermittelt wird.

In Abbildung 5.11 sind zunächst die sich ergebenden Graphen für genau einen einzigen Empfänger mit den beiden Paketgrößen von 46 B (2 Teilpakete) und 1514 B (51 Teilpakete) dargestellt. Gestrichelt eingezeichnet ist zum Vergleich der Verlauf des Unicast-Modus, der jedes einzelne Teilpaket stets bis zu 25 mal zu jedem einzelnen Gerät überträgt.

Erwartungsgemäß zeichnet sich ein deutlich schlechteres Verhalten des Broadcast-Modus ab, da hier ein Doppelfehler bei RFM-Teilpaket und Neuübertragungsanforderung im Gegensatz zum Unicast-Schema wegen Erreichen eines Timeouts auf Sender- oder Empfängerseite sofort zu einem verlorenen Ethernet-Frame führt und eben kein statisches Kommunikationsschema implementiert ist.

Die Unicast-Übertragung bietet damit im Falle eines einzigen Gerätes eine wesentlich geringere Fehlerwahrscheinlichkeit für ein einzelnes Ethernet-Frame, und dies nahezu unabhängig von seiner Größe.

Allerdings ist bereits ein Trend erkennbar: Mit steigender Anzahl an RFM-Teilpaketen verringert sich der Unterschied, da im Broadcast-Modus eine größere Chance auf erfolgreiches Übertragen von Teilen des Frames besteht. In diesem Fall beginnt auch das Broadcast-Protokoll dank des indizierten NACK-Paketes die Übertragung an der richtigen Stelle von Neuem.

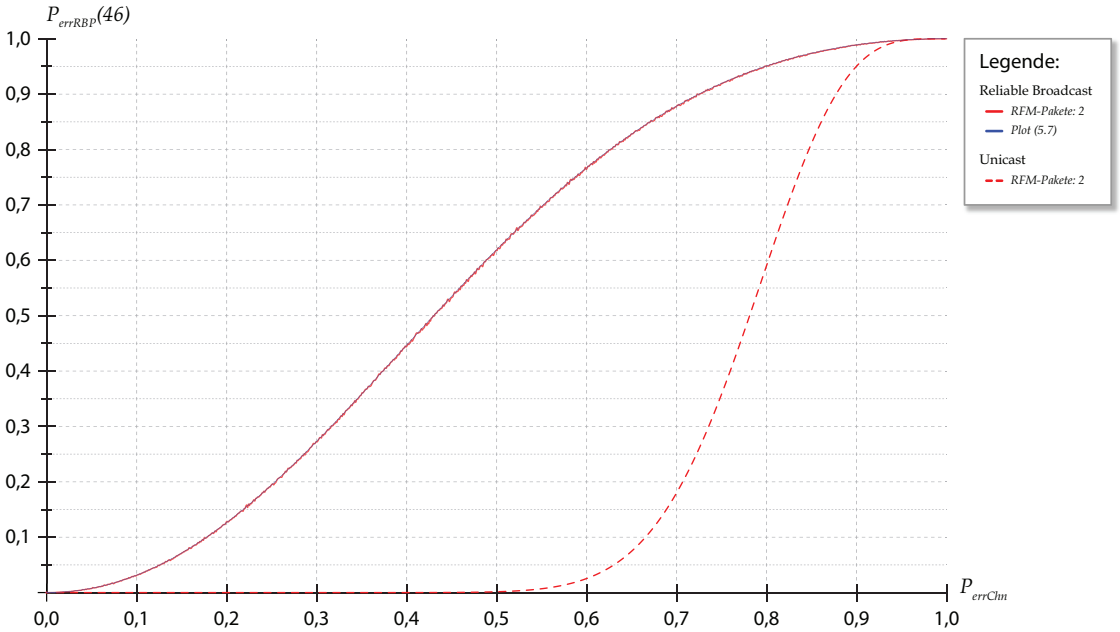
5.2.4.3 Unicast- und Broadcast-Modus im Vergleich bei mehreren Empfängern

In einer weiteren Untersuchung wird nun geprüft, wie sich beide Varianten bei mehreren Empfängern hinsichtlich der Fehlerwahrscheinlichkeit für ein Ethernet-Frame bei einer gegebenen Kanalfehlerwahrscheinlichkeit für RFM-Teilpakete verhalten.

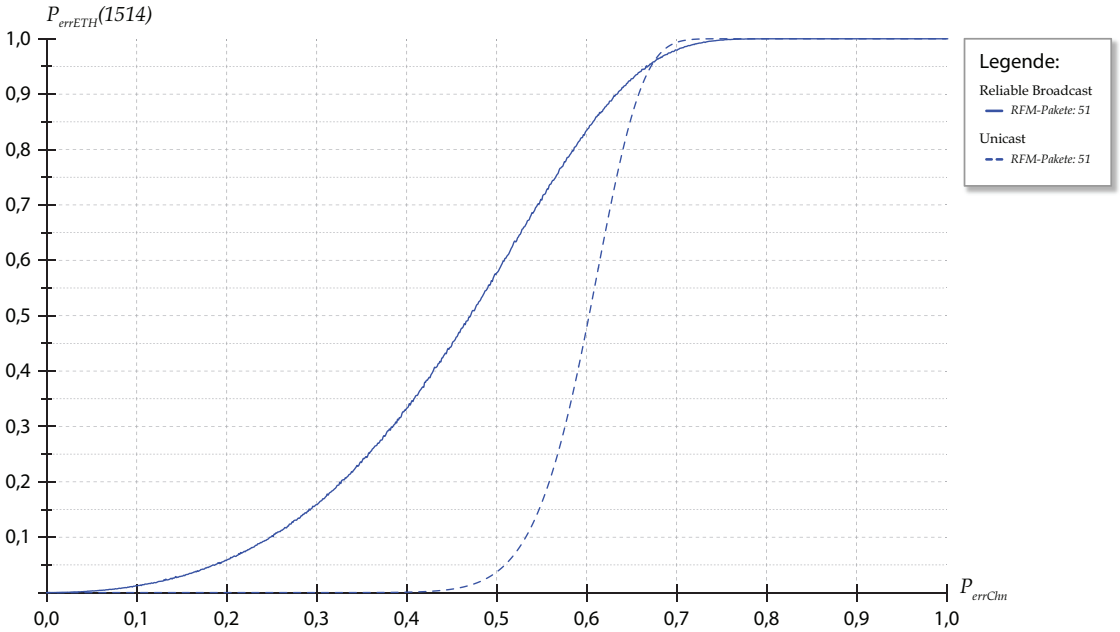
Aufgrund der angenommenen, statistisch unabhängigen Übertragung wird für den Unicast-Modus die Verlustwahrscheinlichkeit nach (5.5) verwendet; es spielt keine Rolle, ob dasselbe Paket konsekutiv an denselben oder mehrere unterschiedliche Empfänger gesendet wird.

Im Broadcast-Modus dagegen liegen abhängige Übertragungen vor, da jedes verlorene Teilpaket an jedem Empfänger potentiell ein korrekt empfangenes NACK auslöst, welches eine erneute Übertragung zur Folge hat. Insgesamt ist nur ein einziges erfolgreich empfangenes NACK pro verlorenem Teilpaket notwendig. Mit steigender Anzahl an Geräten steigt daher letztlich auch die Wahrscheinlichkeit für ein korrekt übertragenes NACK.

Der Broadcast-Modus wird deswegen durch Simulation des Kanalfehlers unter Verwendung des Zufallszahlengenerators des Rechners und Zwischenspeicherung aller Empfängerzustände behandelt. Die Anzahl k der Empfänger wird in 8 Stufen aus der Menge $\{1, 2, 5, 10, 25, 50, 75, 100\}$ variiert.



(a) Framegröße bei 46 B; Plot der Simulation und (5.5).



(b) Framegröße bei 1514 B; Plot der Simulation und (5.5).

Abbildung 5.11: Verlustwahrscheinlichkeiten verschieden großer Ethernet-Frames bei Reliable-Broadcast-Übertragung zu genau einem einzigen Empfänger. Zum Vergleich sind gestrichelt die Graphen der jeweiligen Unicast-Übertragung eingezeichnet.

Abbildung 5.12 stellt die Graphen der Ergebnisse dar, wobei die gestrichelten Linien wie im vorigen Bild 5.11 den Unicast-Modus repräsentieren. Auch hier wird die Untersuchung für die Frame-Größen von 46 B (2 Teilpakete) und 1514 B (51 Teilpakete) durchgeführt.

Aus den Graphen lassen sich mehrere Ergebnisse bezüglich der *Verlustwahrscheinlichkeit* der Ethernet-Frames ableiten:

- Bei kleiner Framegröße oder allgemein wenigen RFM-Teilpaketen erscheint das konsekutive Übertragen von Frames an jedes Gerät mittels *Unicast*-Modus für wenige Geräte ($k < 25$) als die bessere Lösung. Grund für das schlechtere Abschneiden des Broadcast-Modus ist die geringere Wahrscheinlichkeit für versendete NACKs, da die Gesamtzahl der Teilpakete gering ist.

- Der Unterschied zwischen Unicast- und Broadcast-Übertragung verringert sich mit steigender Anzahl an RFM-Teilpaketen pro Ethernet-Frame, da hierbei die Chance für ein NACK auch für vorige fehlgeschlagene Teilübertragungen steigt und daher eine Fehlerkorrektur eher möglich wird.

- Im Broadcast-Modus profitiert das gesamte System immer weniger von steigender Anzahl der Geräte; im rechten Teil von Abbildung 5.12b ist der Unterschied zwischen 75 und 100 Geräten deutlich geringer als etwa zwischen 1 und 25.

- Bei geringer Fehlerwahrscheinlichkeit des Kanals $P_{errChn} < 0{,}05$ verhalten sich beide Varianten vergleichbar unabhängig von der Ethernet-Framegröße.

5.2.5 Skalierverhalten bei unterschiedlicher Anzahl der Empfänger

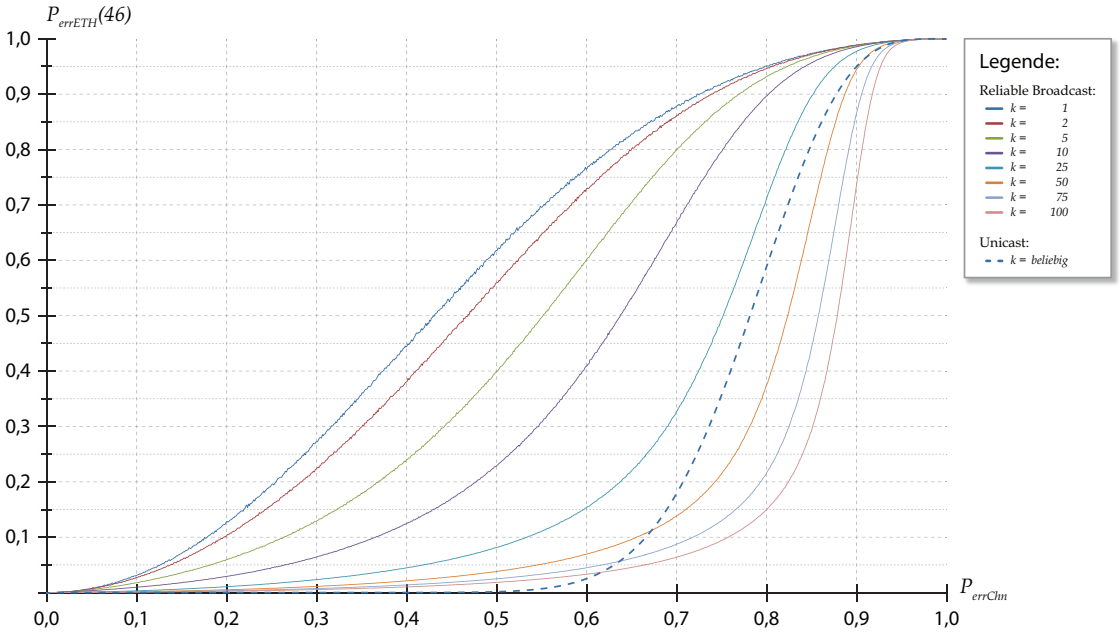
Von weiterem Interesse ist die Skalierbarkeit des Protokolls in beiden Übertragungsmodi wiederum bei unterschiedlicher Größe der Nutzlast sowie der bereits angegebenen Anzahl der beteiligten Empfänger.

Es wird dazu ein direkter Vergleich der erneut vom Sender zu übertragenden Teilpakete unter den folgenden Annahmen vorgenommen:

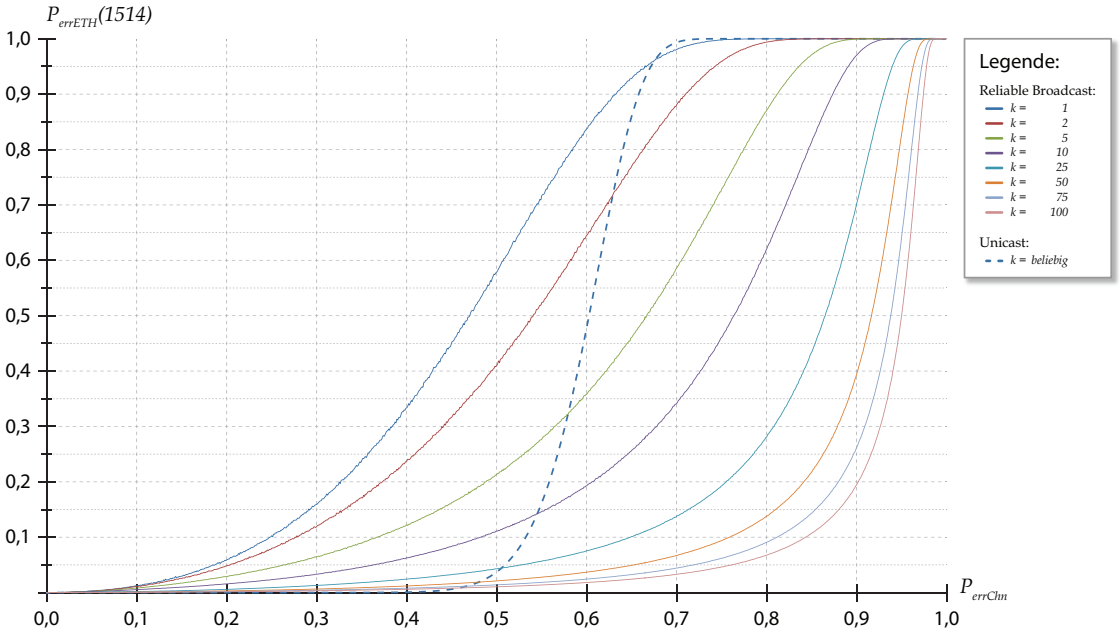
- Die Anzahl der ACKs im Unicast- und die der NACKs im Broadcast-Modus bleibt wegen mangelnder Vergleichbarkeit unberücksichtigt.

- Der Sender kennt im Unicast-Modus die Zieladressen der jeweiligen Empfänger; dies ist so nicht implementiert.

- Kollisionen von NACKs kommen im Modell nicht vor; implementiert ist in der Firmware eine geringfügig zufällige Varianz der Sendezeitpunkte auf jedem Gerät, um Kollisionen möglichst zu vermeiden.



(a) Framegröße bei 46 B; Plot der Simulation und (5.6a) mit $n_{maxTries} = 25$.



(b) Framegröße bei 1514 B; Plot der Simulation und (5.6b) mit $n_{maxTries} = 25$.

Abbildung 5.12: Verlustwahrscheinlichkeiten verschieden großer Ethernet-Frames bei Reliable-Broadcast-Übertragung zu mehreren Empfängern. Zum Vergleich sind gestrichelt die Graphen der jeweiligen Unicast-Übertragung eingezeichnet.

5.2.5.1 Vergleich der mittleren Anzahl der gesendeten RFM-Teilpakete pro Ethernet-Frame

Der erste Schritt zur Bestimmung der Effizienz besteht im Vergleich der mittleren Anzahl der zu sendenden RFM-Teilpakete pro Ethernet-Frame in beiden Übertragungsmodi für die bereits definierte Menge der Empfängerzahlen.

In Abhängigkeit der Fehlerrate wird für den Broadcast-Modus zunächst die Gesamtanzahl der notwendigen Pakete bei 100 000 Versuchen für die beiden Frame-Größen 46 B und 1514 B durch Simulation ermittelt. Die Anzahl der tatsächlich gesendeten Pakete wird dann wieder durch die Anzahl der Versuche dividiert, um die mittlere Zahl der der gesendeten Teilpakete $\overline{N_{RBP}}$ zu erhalten.

Für den Unicast-Modus wird die mittlere erwartete Anzahl der Pakete $\overline{N_{UNI}}$ bei unabhängigen Einzelübertragungen nach (5.8) bestimmt.

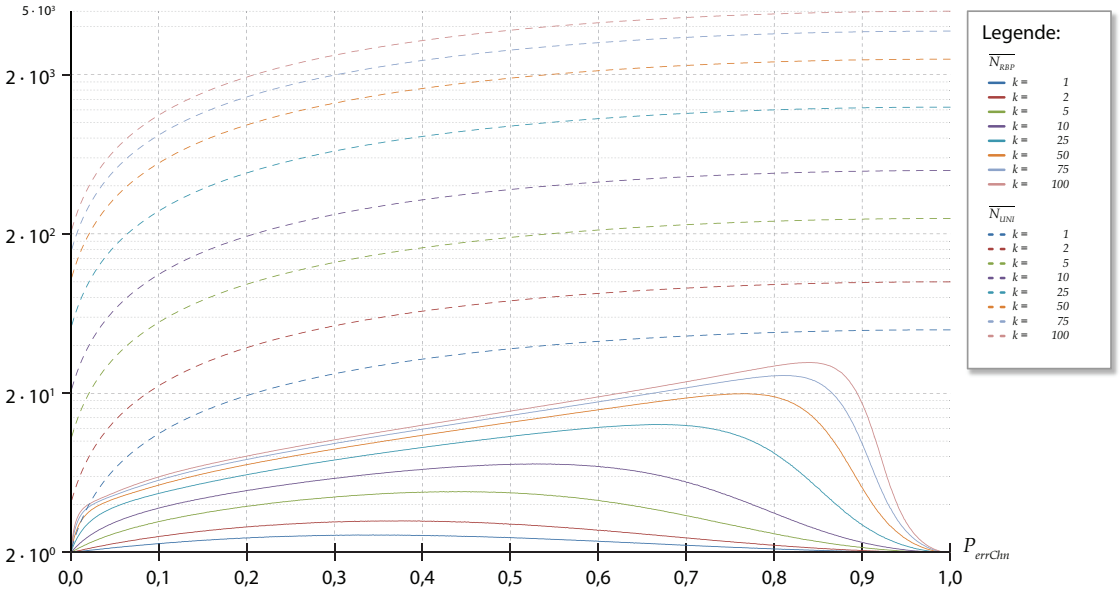
$$\begin{aligned}\overline{N_{UNI}}(P_{errChn}, n_{RFM}, k) &= (1 + 24 \cdot P_{SP}) \cdot n_{RFM} \cdot k \\ &= \left(1 + 24 \cdot \left(2\,P_{errChn} - {P_{errChn}}^2\right)\right) \cdot n_{RFM} \cdot k \end{aligned} \tag{5.8}$$

Abbildung 5.13 stellt die beiden Varianten gegenüber. Die Graphen ermöglichen bereits einige Beobachtungen:

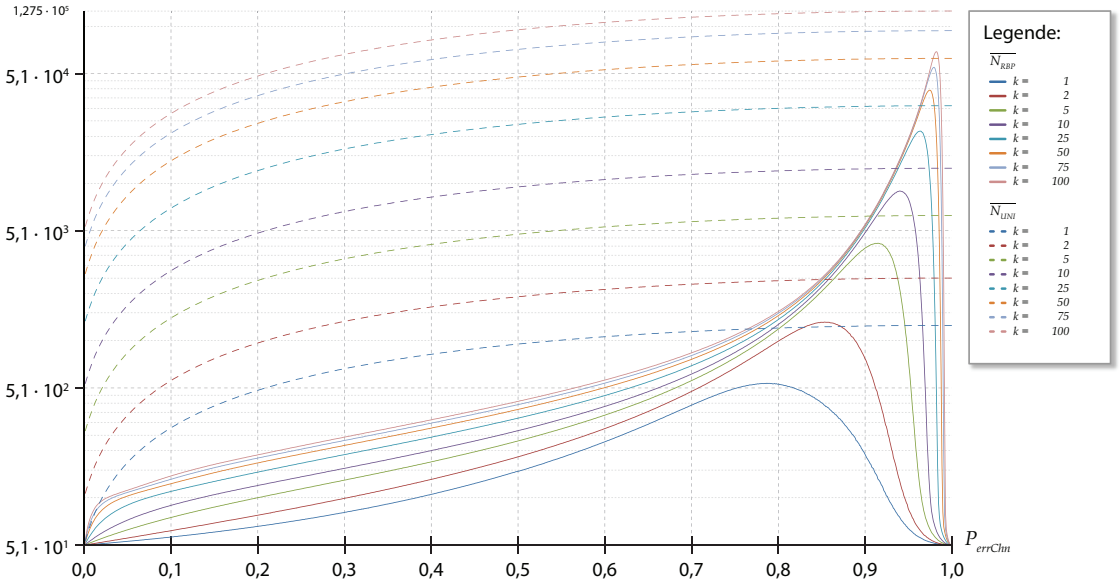
- Die maximale mittlere Anzahl der Pakete fällt bei größten Fehlerraten im Broadcast-Modus wieder auf den minimalen Wert ab; ohne NACKs werden keine Pakete wiederholt. Das Unicast-Protokoll dagegen erreicht das Maximum der erwarteten mittleren Anzahl an Teilpaketen erst mit maximaler Fehlerrate.
- Über die mittlere Zeitdauer der Kanalbelegung läßt sich unter der Annahme, daß die NACKs von allen Empfängern quasi-zeitgleich gesendet werden, die Aussage treffen, daß diese beim Broadcast-Modus um Größenordnungen geringer ist.
- Das Broadcast-Protokoll benötigt in der implementierten Parametrisierung unabhängig von der Fehlerrate des Kanals, der Anzahl der Geräte und der Anzahl der RFM-Teilpakete stets eine weitaus geringere mittlere Anzahl an versendeten RFM-Paketen.

Zunächst erscheint so das Broadcast-Protokoll effizienter als der Unicast-Modus. Allerdings ist in dieser Überlegung die Abhängigkeit von der Fehlerwahrscheinlichkeit des gesamten Ethernet-Frames nicht berücksichtigt, dies erfolgt im nächsten Unterkapitel.

Unberücksichtigt bleibt ebenfalls, daß im Unicast-Modus die Datenrate bei niedrigen Fehlerraten höher ausfällt, da hier ein ACK des Senders unverzüglich die Übertragung des nächsten Teilpakets auslöst. Im Broadcast-Modus muß vom Sender stets auf NACKs gewartet werden, da über die Menge und Zustände der Empfänger nichts bekannt ist.



(a) Plot der Simulation und (5.8) mit $n_{Versuche} = 100\,000$ und $n_{RFM} = 2$.



(b) Plot der Simulation und (5.8) mit $n_{Versuche} = 100\,000$ und $n_{RFM} = 51$.

Abbildung 5.13: Vergleich der mittleren Anzahl der gesendeten Pakete im Unicast- und Broadcast-Modus pro Ethernet-Frame bei unterschiedlichen Frame-Größen. Zur übersichtlicheren Darstellung ist in beiden Teilbildern die logarithmische Skalierung verwendet.

5.2.5.2 Finden des Optimums durch Angleichung der Fehlerwahrscheinlichkeit

Um nun das Optimum aus beiden Protokollen bei definierten bzw. bekannten Parametern n_{RFM} bzw. $size_{ETH}$ und k für eine festgelegte, einzuhaltende Fehlerwahrscheinlichkeit P_{errETH} zu bestimmen, wird der Verlauf der Fehlerwahrscheinlichkeiten eines vollständigen Ethernet-Frames aus Abbildung 5.12 in beiden Protokollvarianten an den Verlauf der mittleren erwarteten Anzahl gesendeter RFM-Teilpakete in Abbildung 5.13 angeglichen und die sich ergebende mittlere Anzahl an notwendigen RFM-Teilpaketen angetragen.

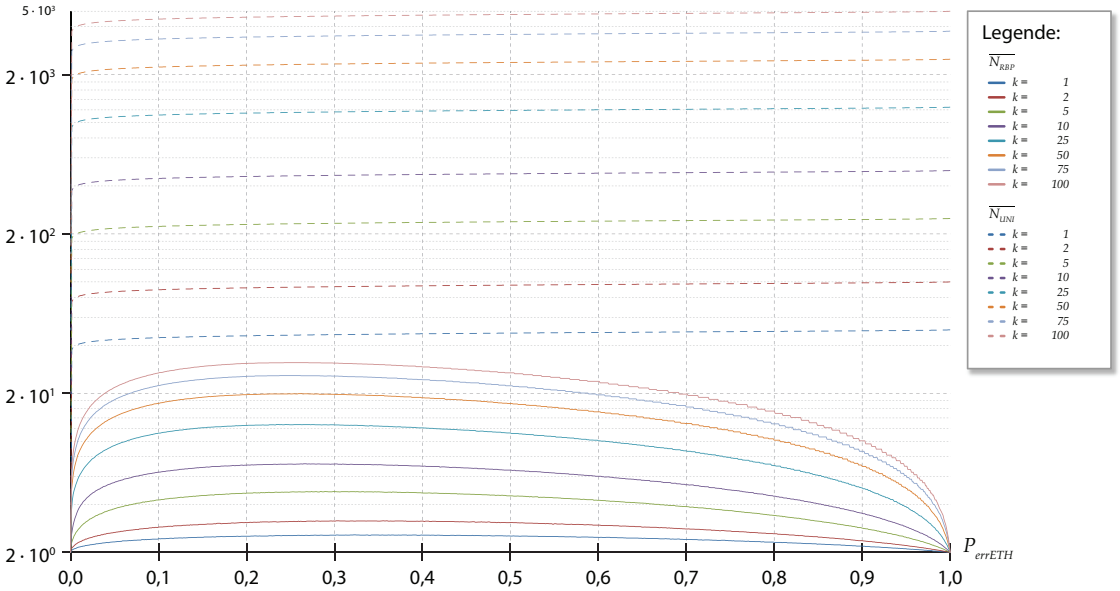
Aufgrund der Simulation des Broadcast-Protokolls wurde diese Aufgabe numerisch gelöst; dadurch ergibt sich eine praktisch nicht vorhandene Auflösung im Bereich extrem kleiner Fehlerraten P_{errETH} in der Nähe des Koordinatenursprungs.

Das Ergebnis ist in Abbildung 5.14 dargestellt. Da sich die Graphen für jeweils eine bestimmte Anzahl k Empfänger für keine Fehlerwahrscheinlichkeit überschneiden, können abschließend für die Evaluierung folgende Schlüsse gezogen werden:

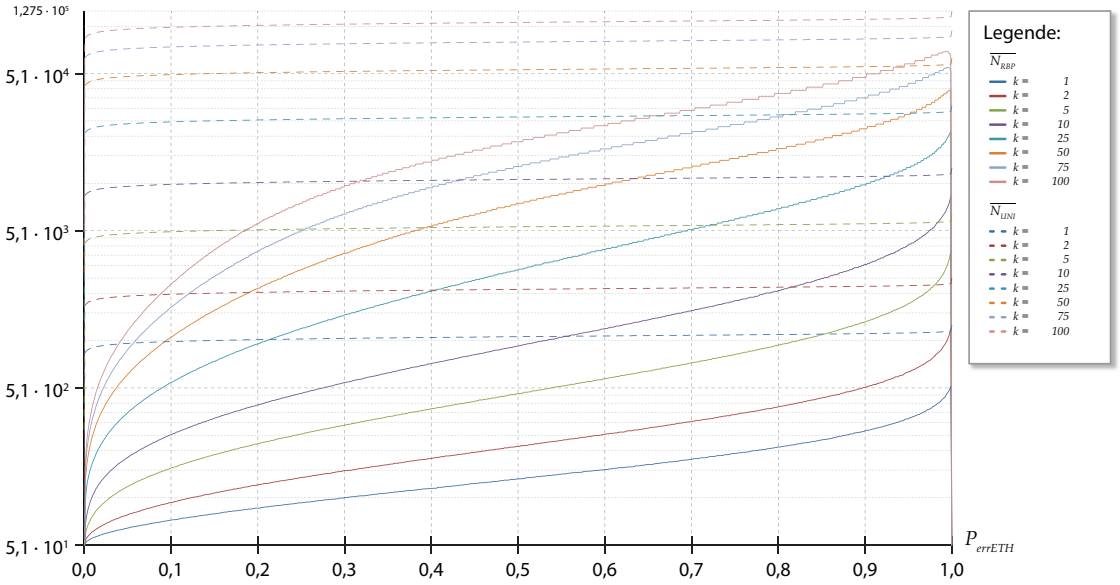
- Das Broadcast-Protokoll erreicht die maximalen mittleren Versandzahlen erst relativ spät bei großer erlaubter Fehlerrate, während der Unicast-Modus insbesondere bei mehreren Empfängern frühzeitig zu einem hohen Versandaufkommen führt.
- Für den Nachrichtentransport bedeutet die Verwendung eines reinen Broadcast-Modus einen möglichen Verzicht auf Adressierbarkeit auf der Ebene der Funkmodule; die Komplexität des Gesamtsystems kann reduziert werden.

Das Broadcast-Protokoll ist damit in der aktuellen Implementierung für alle Variablen und den größten Teil der vorgegebenen Fehlerwahrscheinlichkeiten aufgrund einer geringeren erwarteten RFM-Paketzahl mehrfach effizienter für den Absender eines Ethernet-Frames.

Anzumerken bleibt, daß das Broadcast-Protokoll für einen einzigen Hop zwischen Sender und Empfänger(n) untersucht wurde. Ein interessanter Ausblick besteht in der Untersuchung des Protokolls bei vorhandenen Relay-Stationen und mehreren unabhängigen Kanälen zwischen allen Teilnehmern.



(a) Mittlere Anzahl der gesendeten Teilpakete bei $size_{ETH} = 46$.



(b) Mittlere Anzahl der gesendeten Teilpakete bei $size_{ETH} = 1514$.

Abbildung 5.14: Vergleich der mittleren Anzahl der gesendeten RFM-Pakete im Unicast- und Broadcast-Modus pro Ethernet-Frame bei unterschiedlichen vorgegebenen Ethernet-Fehlerraten. Es wird erneut die logarithmische Skalierung verwendet.

5.3 Hardwareplattformen

Zuletzt werden die erstellten Hardwareplattformen evaluiert. Aufgrund der unterschiedlichsten Anforderungen an potentielle Smart Devices kann dies allerdings nicht allumfassend oder tiefgreifend geschehen. Der Schaltungsentwurf wird weiterhin aus Gründen der besseren Integrierbarkeit und geringeren Time-to-Market in der Verantwortung der jeweiligen Hersteller verbleiben.

5.3.1 Erweiterbarkeit

Dennoch konnte mit den flexibel finalisierbaren Platinen, die bereits eine Benutzerschnittstelle beinhalten und über das erstellte Toolkit angesprochen werden können, ein möglicher Kompromiß aus geringerer Entwicklungszeit und Freiheit bei den Schaltungsparametern und ihrer Ausstattung aufgezeigt werden.

Die Steuerungen sind wie in Kapitel 4 bereits beschrieben, flexibel mit kleinen Bauteilen auf die Anforderungen des Geräts anpaßbar. Sofern größere Elektronik benötigt wird, kann diese durch Zusatzplatinen über Kabel angebunden werden; in diesem Fall werden nur die Steuersignale auf der Benutzerschnittstelle erzeugt.

Weiterhin ist auch die in *C* geschriebene Software nur als Anregung zu verstehen. Sofern genug Speicherplatz zur Verfügung steht, kann die Steuerung und weitere Merkmale auf Protokollebene oder im Schnittstellendesign ergänzt werden.

Alles in allem dient das Toolkit, bestehend aus Hard- und Software, der Demonstration einer flexiblen Architektur, die zumindest konzeptionell auch im Smart Home Einzug halten könnte, um durch offenes Design die Nutzer zu neuen Ideen anzuregen. Inwieweit ein Hersteller die Vorschläge allerdings umsetzt, kann an dieser Stelle nicht beantwortet werden.

5.3.2 Produktive Verwendbarkeit

Es konnte weiterhin gezeigt werden, daß das erstellte Toolkit auch mit ressourcenarmen MCUs verwendet werden kann, die keine Benutzeroberfläche bieten und dennoch starke Merkmale wie Ethernet in Software verarbeiten können.

Ob die Plattform robust genug für einen produktiven Einsatz unter allen erdenklichen Bedingungen ist, kann an dieser Stelle nicht beantwortet werden. Sie sollte eher als Demonstrationsprojekt verstanden werden, mit dem ein paar Geräte für den Hausgebrauch erstellt wurden. Insbesondere die Funkmodule weisen große Schwächen auf und erscheinen eher nicht für produktiven Einsatz geeignet.

6 Schlußbemerkung

In der vorliegenden Arbeit wurde eine Reihe von konzeptionellen und technischen Aspekten zur Vernetzung und Steuerung von Geräten im Smart Home diskutiert. Die folgenden Ausführungen fassen Inhalt und Ergebnis jedes Kapitels zusammen und geben abschließend einen Überblick über ungelöste Punkte und gestatten einen Ausblick zukünftige Arbeiten.

6.1 Zusammenfassung

Kapitel 1 — Im ersten Kapitel wurde die Arbeit mit der im Zuge des *Smart Grid* erforderlich werdenden Möglichkeit zur Steuerung von elektrischen Verbrauchern innerhalb von Privathaushalten, den *Smart Homes*, motiviert. Dazu wurde zunächst auf künftige Energieerzeugungsarten für elektrischen Strom eingegangen, die wegen regenerativer Quellen teilweise durch ein unstetes Erzeugungsprofil gekennzeichnet sind. Weiterhin wurden technische Bestandteile von Geräten innerhalb eines Smart Homes beschrieben, die im Rahmen einer vernetzten Steuerung von Interesse sind. Wie erläutert, können sie in Gruppierungen ähnlicher Funktionen, den *Domänen*, zusammengefaßt betrachtet werden. In einem eigenen Unterkapitel konnten ferner drei *Interessengruppen*, Verbraucher, Gerätehersteller und Energieversorger identifiziert sowie deren unterschiedliche Sichtweisen auf das Smart Home gezeigt werden. Das Kapitel wurde mit Motivation, Zielen und Aufbau der Arbeit abgeschlossen.

Kapitel 2 — Im zweiten Kapitel wurde der Begriff „Heimsteuerung“ von der „Gebäudesteuerung“ abgegrenzt und allgemein ein Überblick über die vorhandenen Lösungen, Produkte und Protokolle gegeben. Nach einer Diskussion von generellen Eingenschaften und Parametern von Heimsteuerungslösungen, insbesondere deren Eigenschaften wie Bussysteme, Netzwerktopologien und verfügbare Geräteklassen, wurde eine Abschätzung über ihre Verwendbarkeit zur Steuerung von generischen elektrischen Verbrauchern entwickelt.

Im nächsten Schritt wurde eine vergleichbare Methodik auf reine Anwendungsprotokolle angewendet und geprüft, ob diese – passend parametrisiert – für den Einsatz im Smart Home sinnvoll verwendet werden können. Es konnte gezeigt werden, daß ein generischer Ansatz mit keinem Protokoll vollständig umsetzbar ist und eine Notwendigkeit für eine eigene Entwicklung besteht.

Kapitel 3 — Im dritten Kapitel wurde nach einer Festlegung konkreter Ziele für ein Anwendungsprotokoll im Smart Home schrittweise das *Smart Home Device Control*

Protocol (SHDP) konzipiert, das der Abfrage und Steuerung von vernetzten Geräten dient. Neben dem Erkennen von im Heimnetz vorhandenen Geräten und ihrer Dienste bietet es auch eine auf wesentliche Eigenschaften reduzierte, kompakte Beschreibung dieser Dienste mit Hilfe von XML und ermöglicht die Nutzung Letzterer durch Steuereinheiten.

Der Begriff *Application Level Manycast* wurde für die Funktion des Protokolls eingeführt, mit nur wenigen Nachrichten eine flexible Kombination aus Diensten und Geräten anzusprechen. Da nur die Anwendungsprotokollebene betrachtet wurde, ist es von eingesetzten Netzwerktechniken unabhängig.

Abschließend wurde eine Integration von externen Parteien, Energieversorgungsunternehmen, in ein Smart Home Netzwerk diskutiert, welche mit Hilfe des konzipierten Protokolls für Geräte umgesetzt werden kann, ohne daß dafür eine global eineindeutige Spezifikation notwendig ist. Vielmehr bieten Flexibilität und Abstraktionsmöglichkeiten des Protokolls eine geeignete Schnittstelle, Steueranweisungen eines EVU passend an betroffene Smart Devices umzusetzen.

Kapitel 4 — Im vierten Kapitel wurde eine umfassende, beispielhafte Implementierung von Komponenten innerhalb eines Smart Home beschrieben. Es wurden einige Modelle von Smart Devices entworfen und konstruiert, die eine Reihe von Funktionen zur Steuerung über das beschriebene Anwendungsprotokoll anbieten. Neben einer kabelgebundenen Nachrichtentransportlösung wurde zur Demonstration eines Bad-Case-Szenarios ein eigenes, für proprietäre und qualitativ geringwertige Funkmodule entworfenes Transportprotokoll implementiert, das ein zuverlässiges Broadcast-Medium simuliert. Im Heimnetz gängige Protokolle wie Ethernet und das Internet Protokoll (IP) konnten so in Software auf ressourcenbeschränkten Geräten ohne zusätzliche Peripherie eingesetzt werden.

Auf der Softwareseite wurden zwei Modelle von Internet-Routern mit einem eigens entworfenen Programmmodul zur Terminierung von IP-basierten Nachrichten und zur umfangreichen Transformation von Inhalten auf Anwendungsebene ausgestattet. Durch eine flexible Regelinterpretation erlaubt die Software eine Parametrisierung von Funktionalitäten unterschiedlicher Hersteller im Smart Home. Sie wurde speziell auf den beschränkten Speicherplatz der Plattformen der Internetrouter hin optimiert und konnte erfolgreich auf beiden Routern stabil eingesetzt werden.

Mit geringerem Detailgrad wurden auch einige Varianten der Verbindung zwischen einem Smart Home und dem jeweiligen Energieversorgungsunternehmen (EVU) diskutiert, die – anders als medial kolportiert – keine Smart Meter erfordern, um eine durch den Benutzer freiwillig gestattete Gerätesteuerung über die bestehende Internetverbindung für zwei beispielhafte Anwendungsfälle zu ermöglichen.

Kapitel 5 — Im fünften Kapitel wurden die theoretischen und praktischen Arbeiten evaluiert, indem ihre Ergebnisse gegen die bereits existierenden Lösungen des zweiten Kapitels und die Anforderungen des dritten Kapitels verglichen wurden. Ferner wurde das implementierte Protokoll des Nachrichtentransports durch empirische und theoretische Arbeit auf seine Performanz unter realistischen Bedingungen hin untersucht.

Insgesamt konnte für den Bereich der Geräteinteraktion im Smart Home, der Gerätebeschreibung, der Steuerung und der Verknüpfung von Diensten, signifikanter Fortschritt erzielt werden. Dank einer Konzentration auf ein offenes Konzept und etablierte Protokolle lassen sich auf dieser Arbeit aufsetzende Projekte auch auf anderen technischen Plattformen einfach realisieren.

Die in der Einleitung aufgestellten Anforderungen der drei beschriebenen Interessensgruppen konnten entweder direkt erfüllt oder durch das vorgestellte Konzept zumindest unterstützt werden. Existierende Lösungen wurden ferner in ihrer Funktionalität stellenweise übertroffen.

6.2 Ausblick

Mit vorliegender Hard- und Software wurde eine beispielhafte Umsetzung der Steuerung eines Smart Homes mit einigen Demonstratoren für eine bestimmte Menge an Gerätetypen aufgezeigt. Auf einer tiefen Hierarchiestufe wurden Basisfunktionen beliebiger elektronischer Geräte über passende Vernetzung innerhalb des Smart Home anderen Verbrauchern zugänglich gemacht und durch eigens erstellte, ebenfalls demonstrativen Charakter besitzende Software gesteuert.

Im größeren Bild allerdings kann diese Demonstration nur als grundsätzliche Arbeit angesehen werden, als die wichtige Grundlage, die weiteren Fortschritt erst ermöglicht. Ein paar darauf aufbauende und sich ergebende Fragestellungen werden zum Abschluß für den Ausblick auf zukünftige Arbeiten genannt und in Relation zu Erreichtem gesetzt.

Weitere Geräteklassen — Zunächst wurde nur die bestimmte Gerätegruppe der weißen Ware experimentell und demonstrativ betrachtet. Im zukünftigen Smart Home werden allerdings eine Reihe weiterer Geräteklassen, etwa mit medizinischer Funktion, zur Erhöhung der Sicherheit, zur Unterhaltung und Kommunikation oder auch völlig anderer Bereiche wie eMobility, Einzug halten. Von großem Interesse sind daher zu untersuchende Fragen nach der Kompatibilität der erarbeiteten Lösungen und Konzepte mit diesen Gerätetypen und Anwendungen.

Reicht für eine zukünftige Anwendung im Smart Home die beschriebene Basissteuerung aus oder muß das Protokoll um neue Elemente, z.B. Videostreams, erweitert werden? Wie werden solche Erweiterungen spezifiziert und ihre Funktion einschließlich der vorhandenen Geräte im Netzwerk bekanntgegeben? Können existierende Standards, etwa die der Digital Living Network Alliance (DLNA), hierfür verwendet und ganz oder teilweise integriert werden oder muß vollkommen Neues geschaffen werden?

Ohne den experimentellen Beleg zu liefern, hat die vorliegende Arbeit bereits Grundlagen für eine Integration mit anderen Techniken und Protokollen geschaffen: auf entsprechend ausgerüsteten Smart Devices wird SHDP einfach parallel zu den übrigen Diensten, z.B. Videostreams, betrieben und bietet über seine `meta`-Elemente Verweise auf diese übrigen Dienste des Gerätes, wie etwa die URL zu einem Videostream des TV-Gerätes oder einen Link auf die für den menschlichen Benutzer optimierte Webseite des internen Webservers.

Weiterhin können Dienste, wie etwa das Ändern der Lautstärke des Gerätes oder ein Kanalwechsel, simultan über SHDP und andere Standards (z.B. der DLNA) angeboten werden.

Dieses Vorgehen ist nicht auf Home-Entertainment beschränkt, sondern läßt sich auch auf andere Bereiche (eMobility, Lademanagement, Medizinprodukte, etc.) anwenden. Ob allerdings alle erdenklichen – auch zukünftigen – Anwendungsfälle erschöpfend abgedeckt sind, ließe sich erst weiterer Arbeit im Dialog mit Beteiligten (Nutzer und Hersteller) untersuchen.

Benutzerschnittstellen und Steuerung — Bewußt vernachlässigt wurde eine für Endbenutzer ansprechende, und möglicherweise sich selbst konfigurierende Steueroberfläche. Die sich ergebenden Fragen betreffen auch das Smart Home als Gesamtkonzept.

In welchem Umfang und welcher Detailtiefe müssen welche Funktionen eines Smart Devices für den Endanwender zugänglich gemacht werden? Wie werden sie visualisiert? Kann dieses Konzept auch für logisch komplexere Verknüpfungen zwischen Smart Devices funktionieren? Welche Anwendungen sind überhaupt denkbar und wer kann sie erstellen? Werden Hersteller sämtliche Vorgaben treffen oder können ambitionierte Benutzer selbst Konfigurationen erstellen? Welche Geräte eignen sich überhaupt zur Steuerung?

Auf die gestellten Fragen können noch keine Antworten gegeben werden, weil sie großteils Anwendungen im Smart Home betreffen. Da bislang keine allumfassende Möglichkeit der Steuerung von Geräten gegeben ist, sind auch Nutzerwillen und -bedarf dafür einfach zu wenig erforscht. Die vorliegende Arbeit hat somit erst einmal die Grundlage geschaffen, um den Beteiligten Gelegenheit zur Auseinandersetzung und insbesondere zum Experimentieren zu geben.

Integration in den Smart Grid-Komplex — Nur am Rande betrachtet wurde die Integration des Smart Home in ein (noch zu konzipierendes) dynamisches Stromverteilungsnetz. Darin kann elektrische Energie potentiell zu jeder Zeit an allen Stellen entnommen und prinzipiell auch eingespeist werden, etwa durch kleine dezentrale Energiewandler (Windkraft, Photovoltaik, Verbrennungskraftanlagen, usw.); wie das Smart Home am besten in ein solches Netz integriert werden sollte, wirft Fragen auf.

Reichen vorhandene Kommunikationsnetze (Telefonnetz, Mobilfunk und Internet) für eine flächendeckende und zeitlich unkritische Anbindung aller Teilnehmer aus? Kann das Smart Grid auch mit offenen Protokollen und dezentralen Verantwortlichkeiten realisiert werden? Was geschieht, wenn sich Knoten fehlverhalten oder ausfallen? Kann ein dezentrales Management funktionieren oder ist man wieder auf große zentrale Steuerrechner angewiesen („Google“-Ansatz)?

Die genannten Fragen beziehen sich auf Sachverhalte außerhalb des Smart Home. Im Wesentlichen sind Entwurf, Aufbau und Betrieb des Smart Grid Aufgabe der Energieversorgungsunternehmen bzw. Netzbetreiber, die für die gesamte Energieversorgung einstehen. Ob daher das angeregte Kommunikationskonzept zwischen Smart Home und Smart Grid allen Anforderungen gerecht wird und alle Anwendungsfälle desselben abdeckt oder von Endanwendern akzeptiert wird, bleibt Gegenstand zukünftiger Untersuchungen,

bei denen ein intensiver Dialog mit Netzbetreibern zu suchen ist.

Betrachtung der IT-Sicherheit — Auch im Rahmen der Sicherung des Smart Homes konnten bei weitem nicht alle Fragen abschließend geklärt werden. Da ein Smart Home nur sinnvoll existieren kann, wenn eine Kommunikation mit der Außenwelt, dem Smart Grid, stattfindet und dadurch zusätzliche Information des „Schwarms“ zur Steuerung der einzelnen Zelle verfügbar ist, öffnet sich unweigerlich die Tür in die digitale persönliche Zone des Endverbrauchers, und zwar in beide Richtungen.

Wie kann die Privatsphäre und der Datenschutz hierbei gewährleistet werden? Wie können Unternehmen am Mißbrauch der entstandenen personenbezogenen Verbrauchsdaten und daraus abgeleiteter impliziter Information gehindert werden? Welche existierenden Lösungen der IT-Sicherheit zur Verschlüsselung und Anonymisierung sind bereits vorhanden? Lassen sie sich von einem Endanwender ohne technisches Know-How bedienen?

Hierfür hat die vorliegende Arbeit auf sich ergebende Schwierigkeiten in unterschiedlichen Bereichen der IT-Sicherheit hingewiesen und Anregungen zur Minimierung der Risiken gegeben. Es sei angemerkt, daß für das Ziel, ein Anwendungsschichtprotokoll zu entwickeln, die IT-Sicherheit nur eine untergeordnete Rolle spielt und diese insbesondere von den Herstellern der jeweiligen Smart Devices umzusetzen ist. Dennoch handelt es sich hierbei um einen sehr wichtigen Teilaspekt des Smart Homes, der kooperativ von Betreibern des Smart Grid und Herstellern von Produkten hierfür zu bearbeiten ist.

Alles in allem hat die Arbeit am Smart Home gerade erst begonnen.

Literaturverzeichnis

[1] Strese, Hartmut ; Seidel, Uwe ; Knape, Thorsten ; Alfons, Botthof: *Smart Home in Deutschland*. Institut für Innovation und Technik (iit), 2010. – ISBN 978–3897501652

[2] Reichwald, Ralf ; Piller, Frank: *Interaktive Wertschöpfung. Open Innovation, Individualisierung und neue Formen der Arbeitsteilung*. First. Gabler, 2009. – ISBN 3834901067

[3] Martini, Florian: *Siemens Global Website, Pictures of the Future*. http://www.siemens.com/innovation/de/publikationen/zeitschriften_pictures_of_the_future/pof_herbst_2005/intelligente_vernetzung/t_com_haus.htm. Version: September 2005. – [Online; Abruf am 17. Juli 2013]

[4] Bundesministerium für Wirtschaft und Technologie: *E-Energy – Smart Energy made in Germany*. http://www.e-energy.de/. – [Online; Abruf am 28. September 2013]

[5] Bundesministerium der Justiz: *Gesetz über die Elektrizitäts- und Gasversorgung (EnWG), §21*. http://www.gesetze-im-internet.de/enwg_2005/__21.html. Version: Januar 2005. – [Online; Abruf am 17. Juli 2013]

[6] Hope RF: *RFM70 Datenblatt*. http://www.hoperf.com/upload/rf/RFM70.pdf. – [Online; Abruf am 22. Juli 2013]

[7] Z-Wave Alliance: *Z-Wave Territories and Products*. http://products.z-wavealliance.org/. – [Online; Abruf am 22. Juli 2013]

[8] Z-Wave Alliance: *Our Companies*. http://www.z-wavealliance.org/member-companies. Version: Juni 2013. – [Online; Abruf am 22. Juli 2013]

[9] International Telecommunication Union: *G.9959: Short range narrow-band digital radiocommunication transceivers*. http://www.itu.int/rec/T-REC-G.9959-201202-I/en. Version: Februar 2012. – [Online; Abruf am 22. Juli 2013]

[10] International Telecommunication Union (ITU): *G.9959: Short range narrow-band digital radiocommunication transceivers - PHY and MAC layer specifications*. Recommendation G.9959. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.9959-201202-I!!PDF-E&type=items. Version: Februar 2012 (Recommendation). – [Online; Abruf am 22. Juli 2013]

[11] ZigBee Alliance: *Understanding ZigBee*. http://zigbee.org/About/UnderstandingZigBee.aspx. – [Online; Abruf am 1. August 2013]

[12] IEEE: *Institute of Electrical and Electronics Engineers 802.15™: WIRELESS PERSONAL AREA NETWORKS (PANs)*. http://standards.ieee.org/about/get/802/802.15.html. – [Online; Abruf am 1. August 2013]

[13] ZigBee Alliance: *ZigBee Standards Overview*. http://zigbee.org/Standards/Overview.aspx. – [Online; Abruf am 1. August 2013]

[14] Dusan Stevanovic: *ZigBee / IEEE 802.15.4 Standard*. http://www.cse.yorku.ca/~dusan/Zigbee-Standard-Talk.pdf. Version: Juni 2007. – [Online; Abruf am 1. August 2013]

[15] ZigBee Alliance: *ZigBee Certification*. http://zigbee.org/Certification.aspx. – [Online; Abruf am 1. August 2013]

[16] DIGI Webshop: *XBee®868LP for Europe*. http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-868lp#models. – [Online; Abruf am 1. August 2013]

[17] EnOcean Alliance: *EnOcean Radio Protocol Specification 1.0*. http://www.enocean.com/fileadmin/redaktion/pdf/tec_docs/EnOceanRadioProtocol.pdf. – [Online; Abruf am 2. August 2013]

[18] EnOcean Alliance: *Liste der verfügbaren Produkte*. http://www.enocean-alliance.org/de/produkte. – [Online; Abruf am 2. August 2013]

[19] International Organisation for Standardisation (ISO): *Wireless Short-Packet (WSP) protocol optimized for energy harvesting - Architecture and lower layer protocols*. http://hes-standards.org/doc/SC25_WG1_N1493.pdf. Version: 06 2011. – [Online; Abruf am 2. August 2013]

[20] EnOcean GmbH: *EnOcean Equipment Profiles - Version 2.5*. http://www.enocean-alliance.org/eep/. Version: 03 2013. – [Online; Abruf am 2. August 2013]

[21] EnOcean GmbH: *Security of EnOcean Radio Networks - Version 1.3*. http://www.enocean.com/en/security-specification/. Version: 10 2012. – [Online; Abruf am 2. August 2013]

[22] eQ-3 AG: *HomeMatic – wir machen Home Control einfach*. http://www.homematic.com/. – [Online; Abruf am 5. August 2013]

[23] König, Rudolf: *FHEM Webseite*. http://fhem.de/fhem.html#Description. – [Online; Abruf am 31. Juli 2013]

[24] König, Rudolf: *FHEM Command Reference (Section CUL_HM)*. http://fhem.de/commandref.html#CUL_HM. – [Online; Abruf am 31. Juli 2013]

[25] sh-Elektronik GmbH: *CC1101-USB-Lite 868MHz (CUL)*. http://shop.busware.de/product_info.php/cPath/1/products_id/29. – [Online; Abruf am 31. Juli 2013]

[26] KNX Deutschland: *Grundlagenwissen zum KNX Standard*. http://www.knx.de/download/index.php?navid=66. – [Online; Abruf am 2. August 2013]

[27] KNX Association: *KNX Powerline PL 110*. http://www.knx.org/fileadmin/template/documents/downloads_support_menu/KNX_tutor_seminar_page/basic_documentation/Powerline_E0307d.pdf. – [Online; Abruf am 3. August 2013]

[28] WEINZIERL ENGINEERING GmbH: *Stack Implementierung für KNX-RF*. http://www.weinzierl.de/download/references/KnxRf_Paper_D.pdf. Version: Juli 2005. – [Online; Abruf am 8. August 2013]

[29] L., Markus: *KNX Paketaufbau*. http://knx-user-forum.de/lexikon/paketaufbau-60/knx-eib-1.html. Version: 12 2010. – [Online; Abruf am 8. August 2013]

[30] KNX Association: *System Specifications — Interworking — KNX Datapoint Types*. ftp://85.214.247.170/Download/Datapoint.pdf. Version: 2010. – [Online; Abruf am 8. August 2013]

[31] Miele & Cie. KG Deutschland: *Hausgerätevernetzung – Was ist Miele@home?* http://www.miele.de/de/haushalt/produkte/44668.htm. – [Online; Abruf am 26. Juli 2013]

[32] Innova24 Preisvergleich: *Miele XGW 2000 Miele@home Gateway*. http://www.innova24.biz/item/haushaltsgeraete/zubehoer-einbaugeraete/miele-xgw-2000-mielehome-gateway-77529.htm. – [Online; Abruf am 24. Juli 2013]

[33] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. http://www.rfc.net/rfc2616.html. Version: 1999. – [Online; Abruf am 5. August 2013]

[34] Goland, Y. ; Whitehead, E. ; Faizi, A. ; Carter, S. ; Jensen, D.: *RFC 2518: HTTP Extension for Distributed Authoring – WebDAV*. http://www.ietf.org/rfc/rfc2518.txt. Version: 1999. – [Online; Abruf am 23. September 2013]

[35] Söllner, Christoph ; Baumgarten, Uwe: Bridging the Last Mile on Application Level in a Smart Home. In: *The 4th International Conference on Smart Communications in Network Technologies (SaCoNeT)*. Paris, Frankreich : Paris ESLSCA Business School, Juni 2013

[36] Winer, Dave: *XML-RPC Specification*. http://xmlrpc.scripting.com/spec. Version: Juni 2003. – [Online; Abruf am 23. Juli 2013]

[37] Laurent, Simon S. ; Dumbill, Edd ; Johnston, Joe: *Programming Web Services with XML-RPC*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2001. – ISBN 0596001193

[38] Harold, Elliotte: *Chapter 2. XML Protocols: XML-RPC and SOAP*. http://cafeconleche.org/books/xmljava/chapters/ch02s05.html. Version: August 2002. – [Online; Abruf am 25. Juli 2013]

[39] Harold, Elliote R. ; Means, W. S.: *XML in a nutshell*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2002. – ISBN 0–596–00292–0

[40] Snell, James ; Tidwell, Doug ; Kulchenko, Pavel: *Programming Web services with SOAP*. Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2002. – ISBN 0–596–00095–2

[41] World Wide Web Consortium: *Namespaces in XML 1.0 (Third Edition)*. http://www.w3.org/TR/xml-names/. Version: Dezember 2009. – [Online; Abruf am 22. Juli 2013]

[42] Organization for the Advancement of Structured Information Standards (OASIS): *Devices Profile for Web Services Version 1.1*. http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html. Version: Juli 2009. – [Online; Abruf am 22. Juli 2013]

[43] Zeeb, Elmar: *Web Services for Devices*. http://www.ws4d.org/. Version: 2013. – [Online; Abruf am 23. September 2013]

[44] Shelby, Zach ; Hartke, K. ; Bormann, Carsten: *Constrained Application Protocol (CoAP)*. https://datatracker.ietf.org/doc/draft-ietf-core-coap/. Version: 2013 (draft). – [Online; Abruf am 26. September 2013]

[45] IETF: *Constrained RESTful Environments (core)*. https://datatracker.ietf.org/wg/core/. Version: 2013. – [Online; Abruf am 26. September 2013]

[46] Shelby, Zachary: *Constrained RESTful Environments (CoRE) Link Format*. RFC 6690 (Draft Standard). http://datatracker.ietf.org/doc/rfc6690/. Version: Februar 2012 (Request for Comments). – [Online; Abruf am 20. Oktober 2013]

[47] World Wide Web Consortium: *Web Application Description Language*. http://www.w3.org/Submission/wadl/. Version: August 2009. – [Online; Abruf am 22. Juli 2013]

[48] Contiki Community: *Contiki – The Open Source OS for the Internet of Things*. http://www.contiki-os.org/. – [Online; Abruf am 28. August 2013]

[49] Söllner, Christoph ; Baumgarten, Uwe: Event-Driven Communication on Application Level in a Smart Home. In: *The Second International Conference on Smart Systems, Devices and Technologies (SMART)*. Rom, Italien : IARIA Journals, Juni 2013

[50] Martin, Robert C.: *Clean Code: A Handbook of Agile Software Craftsmanship*. 1. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2008. – ISBN 0132350882, 9780132350884

[51] Simpson, John E.: *O'REILLY — The Naming of Parts*. http://www.xml.com/pub/a/2001/07/25/namingparts.html. Version: 7 2005. – [Online; Abruf am 8. August 2013]

[52] O'Sullivan, Nathan: *UDP Broadcast Packet Relay*. https://www.joachim-breitner.de/udp-broadcast-relay/. Version: September 2003. – [Online; Abruf am 14. August 2013]

[53] Stadtwerke Bochum: *Smart Meter*. http://www.stadtwerke-bochum.de/privatkunden/produkte/strom/smart_meter.html. – [Online; Abruf am 20. August 2013]

[54] Future Technology Devices International Ltd.: *FT232R USB UART IC*. http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf. Version: 2010. – [Online; Abruf am 21. August 2013]

[55] Microchip Technology Inc.: *ENC28J60 Data Sheet*. http://ww1.microchip.com/downloads/en/devicedoc/39662a.pdf. Version: 2004. – [Online; Abruf am 21. August 2013]

[56] TP-Link: *Ultimate Wireless N Gigabit Router – TL-WR1043ND*. http://www.tp-link.com/Resources/document/TL-WR1043ND_V1_Datasheet.zip. – [Online; Abruf am 21. August 2013]

[57] OpenWrt Wiki: *OpenWRT Table of Hardware – TP-Link TL-WR1043ND*. http://wiki.openwrt.org/toh/tp-link/tl-wr1043nd. Version: August 2013. – [Online; Abruf am 21. August 2013]

[58] TP-Link: *N900 Wireless Dual Band Gigabit Router – TL-WDR4900*. http://www.tp-link.com.de/Resources/document/TL-WDR4900_V1.0_Datasheet.zip. – [Online; Abruf am 21. August 2013]

[59] OpenWrt Wiki: *OpenWRT Table of Hardware – TP-Link TL-WDR4900*. http://wiki.openwrt.org/toh/tp-link/tl-wdr4900. Version: August 2013. – [Online; Abruf am 21. August 2013]

[60] Dillmann, Rüdiger ; Gockel, Tilo ; Schröder, Joachim: *Embedded Linux*. Springer-Verlag Berlin Heidelberg, 2009. – ISBN 9783540786207

[61] OpenWrt Developer Team: *OpenWRT official website*. https://openwrt.org/. – [Online; Abruf am 21. August 2013]

[62] OpenWrt Developer Team: *Index of /backfire/10.03.1/ar71xx/*. http://downloads.openwrt.org/backfire/10.03.1/ar71xx/openwrt-ar71xx-tl-wr1043nd-v1-squashfs-factory.bin. – [Online; Abruf am 21. August 2013]

[63] OpenWrt Developer Team: *Index of /snapshots/trunk/mpc85xx/*. http://downloads.openwrt.org/snapshots/trunk/mpc85xx/openwrt-mpc85xx-generic-tl-wdr4900-v1-squashfs-factory.bin. – [Online; Abruf am 21. August 2013]

[64] OpenWrt Developer Team: *OPKG Package Manager*. http://wiki.openwrt.org/doc/techref/opkg. – [Online; Abruf am 21. August 2013]

[65] Cadsoft GmbH: *EAGLE Download Page*. http://www.cadsoft.de/download-eagle/. – [Online; Abruf am 24. August 2013]

[66] Beta Leyout GmbH – PCB-Pool: *PCB-Pool Webseite*. http://www.pcb-pool.com/ppde/index.html. – [Online; Abruf am 24. August 2013]

[67] Pfeifer, Thomas: *Platinen ätzen mit der Direkt-Toner-Methode*. http://thomaspfeifer.net/platinen_aetzen.htm. – [Online; Abruf am 24. August 2013]

[68] Schmitz, Jörg: *Platinen selber ätzen (Belichten und Entwickeln)*. http://www.analog-synth.de/selberaetzen/belichten.htm. – [Online; Abruf am 24. August 2013]

[69] Margolis, Michael: *Arduino Cookbook - Recipes to Begin, Expand, and Enhance Your Projects.* O'Reilly, 2011. – ISBN 978–0–596–80247–9

[70] Atmel Corporation: *8-bit Atmel XMEGA A Microcontroller – MEGA A MANUAL – 8077I*. http://www.atmel.com/Images/doc8077.pdf. Version: November 2012. – [Online; Abruf am 24. August 2013]

[71] Atmel Corporation: *8/16-bit Atmel XMEGA Microcontroller – ATxmega[128/64/32/16]A4U – 8387D*. http://www.atmel.com/images/atmel-8387-8-and16-bit-avr-microcontroller-xmega-a4u_datasheet.pdf. Version: Februar 2013. – [Online; Abruf am 24. August 2013]

[72] mikrocontroller.net: *AVR In System Programmer*. http://www.mikrocontroller.net/articles/AVR_In_System_Programmer. – [Online; Abruf am 24. August 2013]

[73] Hope RF: *RFM70 V1.0 (Low Power High Performance 2.4 GHz GFSK Transceiver Module)*. http://www.hoperf.com/upload/rf/RFM70.pdf. – [Online; Abruf am 24. August 2013]

[74] Hope RF: *RFM70 2.4G RF transceiver*. http://www.hoperf.com/rf_fsk/24g/rfm70.htm. – [Online; Abruf am 24. August 2013]

[75] Hope RF: *RFM73 Replace RFM70 Precautions*. http://embeddedwirelesssolutions.com/resources/Precautions_on_RFM73_Replacing_RFM70.pdf. Version: December 2012. – [Online; Abruf am 30. August 2013]

[76] International Rectifier: *IRF7341 HEXFET Power MOSFET*. http://www.irf.com/product-info/datasheets/data/irf7341.pdf. Version: April 2005. – [Online; Abruf am 26. August 2013]

[77] Sensirion AG: *Datasheet SHT1x (SHT10, SHT11, SHT15) – Humidity and Temperature Sensor IC* . http://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity/Sensirion_Humidity_SHT1x_Datasheet_V5.pdf. – [Online; Abruf am 26. August 2013]

[78] Hope RF: *HP03 Series of calibrated sensor module (v1.3)*. http://www.hoperf.com/upload/sensor/HP03S.pdf. Version: Februar 2010. – [Online; Abruf am 26. August 2013]

[79] AVAGO TECHNOLOGIES: *APDS-9300*. http://www.avagotech.com/docs/AV02-1077EN. – [Online; Abruf am 26. August 2013]

[80] International Rectifier: *IRLML2502 HEXFET Power MOSFET*. http://www.irf.com/product-info/datasheets/data/irlml2502.pdf. Version: April 2003. – [Online; Abruf am 26. August 2013]

[81] Electronic Assembly: *EA DOGM LCDs – Produktwebseite*. http://www.lcd-module.de/produkte/dog.html. Version: August 2013. – [Online; Abruf am 26. August 2013]

[82] Electronic Assembly: *DOGM GRAFIK SERIE – 132x32 Pixel*. http://www.lcd-module.de/deu/pdf/grafik/dogm132-5.pdf. Version: Januar 2009. – [Online; Abruf am 26. August 2013]

[83] maxim integrated: *DS18B20 – Programmable Resolution 1-Wire Digital Thermometer*. http://pdfserv.maximintegrated.com/en/ds/DS18B20.pdf. Version: April 2008. – [Online; Abruf am 26. August 2013]

[84] Springbok Digitronics: *Understanding 1-Wire Series*. http://www.1wire.org/Files/Articles/1-Wire-Design%20Guide%20v1.0.pdf. Version: August 2004. – [Online; Abruf am 26. August 2013]

[85] Otto, Fabian: *Konzeption und Implementierung eines Funkbusses zur IP-basierten Gebäudeautomation*, Technische Universität München, Masterarbeit, Mai 2013. – betreut von Christoph Söllner

[86] Söllner, Christoph ; Baumgarten, Uwe: Implementation of a Reliable Broadcast Protocol to relay Ethernet Frames in a Smart Home Scenario Over Low-End Proprietary Radio Hardware. In: *Eighth IEEE International Workshop on Practical Issues in Building Sensor Network Applications*. Sydney, Australien : n/a, Oktober 2013. – akzeptiert, nicht veröffentlicht

[87] Tanenbaum, Andrew: *Computer Networks*. 4th. Prentice Hall Professional Technical Reference, 2002. – ISBN 0130661023

[88] Droms, R.: *Dynamic Host Configuration Protocol*. RFC 2131 (Draft Standard). http://www.ietf.org/rfc/rfc2131.txt. Version: März 1997 (Request for Comments). – [Online; Abruf am 28. August 2013]

[89] Camera, Dean: *LUFA (2013)*. http://www.fourwalledcubicle.com/LUFA.php. Version: März 2013. – [Online; Abruf am 28. August 2013]

[90] Montenegro, G. ; Kushalnagar, N. ; Hui, J. ; Culler, D.: *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944 (Proposed Standard). http://www.ietf.org/rfc/rfc4944.txt. Version: September 2007 (Request for Comments). – [Online; Abruf am 28. August 2013]

[91] Hui, J. ; Thubert, P.: *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282 (Proposed Standard). `http://www.ietf.org/rfc/rfc6282.txt`. Version: September 2011 (Request for Comments). – [Online; Abruf am 28. August 2013]

[92] Ouyang, Beini ; Hong, Xiaoyan ; Yi, Y.: A comparison of reliable multicast protocols for mobile ad hoc networks. In: *SoutheastCon, 2005. Proceedings. IEEE*, 2005, S. 339–344

[93] Demirbas, M. ; Hussain, M.: A MAC Layer Protocol for Priority-based Reliable Multicast in Wireless Ad Hoc Networks. In: *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, 2006, S. 1–10

[94] Saha, S. ; Hussain, S.R. ; Ashikur Rahman, A.K.M.: RBP: Reliable Broadcasting Protocol in Large Scale Mobile Ad Hoc Networks. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010. – ISSN 1550–445X, S. 526–532

[95] Eckert, Claudia: *IT-Sicherheit: Konzepte - Verfahren - Protokolle*. Oldenbourg Wissenschaftsverlag, 2013. – ISBN 9783486721386

[96] Atmel Corporation: *AVR1317: Using the XMEGA built-in DES accelerator*. `http://www.atmel.com/Images/doc8105.pdf`. Version: April 2008. – [Online; Abruf am 31. August 2013]

[97] Atmel Corporation: *AVR1318: Using the XMEGA built-in AES accelerator*. `http://www.atmel.com/Images/doc8106.pdf`. Version: April 2008. – [Online; Abruf am 31. August 2013]

[98] Reusing, Tobias: *Embedded AVR Webservices*, Technische Universität München, Interdisziplinäres Projekt, August 2012. – betreut von Christoph Söllner

[99] Messdaghi, Aram: *Demonstration of Web Service Orchestration within a Sensor Node Network using HTTP Based Communication over Different Physical Layers*, Technische Universität München, Masterarbeit, November 2012. – betreut von Christoph Söllner

[100] TinyOS Website: *TinyOS Website*. `http://www.tinyos.net/`. – [Online; Abruf am 28. August 2013]

[101] Baumgarten, Uwe ; Siegert, Hans-Jürgen: *Betriebssysteme - eine Einführung (6. Auflage)*. Oldenbourg, 2006. – ISBN 9783486582116

[102] Harbison, Samuel P. ; Steele, Guy L.: *C: A Reference Manual (5^{th} Edition)*. Prentice-Hall, 2002. – ISBN 013089592X

[103] Reusing, Tobias: *Design and Implementation of a flexible and configurable software based upon OpenWRT to transform and route IP messages*, Technische Universität München, Masterarbeit, September 2013. – betreut von Christoph Söllner

[104] IEEE and The Open Group: *The Open Group Base Specifications Issue 7 – crontab*. http://pubs.opengroup.org/onlinepubs/9699919799/utilities/crontab.html. Version: 2013. – [Online; Abruf am 11. September 2013]

[105] die.net: *nc(1) – Linux man page*. http://linux.die.net/man/1/nc. Version: August 2006. – [Online; Abruf am 11. September 2013]

[106] Gordon Lyon: *Nmap Network Scanning – Chapter 18. Nping Reference Guide*. http://nmap.org/book/nping-man-briefoptions.html. – [Online; Abruf am 11. September 2013]

[107] Tristan Lelong: *libroxml version=2.2.3*. http://www.libroxml.net/. Version: 2013. – [Online; Abruf am 12. September 2013]

[108] Groff, James ; Weinberg, Paul: *SQL The Complete Reference, 3rd Edition*. 3. New York, NY, USA : McGraw-Hill, Inc., 2010. – ISBN 0071592555, 9780071592550

[109] Kreibich, Jay A.: *Using SQLite*. O'Reilly Media, 2010. – ISBN 9781449399641

[110] OpenVPN Technologies, Inc.: *OpenVPN corporate website*. http://openvpn.net/. Version: 2013. – [Online; Abruf am 12. September 2013]

[111] cFos Software GmbH: *hrPING v5.04*. http://www.cfos.de/de/ping/ping.htm. – [Online; Abruf am 19. September 2013]

[112] Folkert Vanheusden: *httping(1) – Linux man page – v1.4.1*. http://www.cfos.de/de/ping/ping.htm. – [Online; Abruf am 22. September 2013]

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Verwendete Abkürzungen

3DES	Triple-DES
6LoWPAN	IPv6-over-LowPower-Wireless-Networks
ACK	Acknowledge
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
ATM	Asychronous Transfer Mode
BidCoS	Bidirectional Communication System
CA	Collision Avoidance
CDC	Communication-Device-Class
CMAC	Cipher-based Message Authentication Code
CoAP	Constrained Application Protocol
CoRE	Constrained ReSTful Environments
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense Multiple Access
DECT	Digital Enhanced Cordless Telecommunications
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DLNA	Digital Living Network Alliance
DNS	Domain Name System
DOM	Document Object Model
DoS	Denial-of-Service
DPWS	Devices Profile for Web Services
DSL	Digital Subscriber Line
DTD	Document Type Definition
EEPROM	Electrically Erasable and Programmable Read-Only Memory
EIA	Electronic Industries Alliance
EIB	European Installation Bus
EMV	elektromagnetische Verträglichkeit
EVU	Energieversorgungsunternehmen
FEC	Forward Error Correction
FHEM	Freundliche Hausautomatisierung und Energie-Messung

FSK	Frequency Shift Keying
FTP	File Transfer Protocol
GPIO	General Purpose Input/Output
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol, gesichert
I^2C	Inter-Integrated-Circuit
IC	Integrated Circuit
ICMP	Internet Control Message Protocol
ID	Identifikation
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protokoll
IPv4	Internet Protocol, Version 4
IPv6	Internet Protocol, Version 6
IRQ	Interrupt-Request
ISO	International Organisation for Standardisation
ISP	In-System Programmierung
ISR	Interrupt Service Routine
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
KNX	Konnex Protokoll-Suite
LAN	Local Area Network
LCD	Liquid Crystal Display
LDAP	Leightweight Directory Access Protocol
LED	Light Emitting Diode
MAC	Media Access Control
MCU	Micro Controller Unit
MIMO	Multiple Input Multiple Output
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MSB	Most Significant Bit
MTU	Maximum Transmission Units
NACK	Non-Acknowledge
NAT	Network Address Translation
NIC	Network Interface Card
NRZ	Non-Return to Zero
NTP	Network Time Protocol

OHCI	Open Host Controller Interface
OOK	On-Off Keying
OSI	Open Systems Interconnection
PAT	Port Address Translation
PDU	Protocol Data Unit
PCB	Printed Circuit Board
PHY	Physical Layer
PKI	Public Key Infrastructure
PoE	Power-over-Ethernet
PPPoE	Point-to-Point-Protocol-over-Ethernet
PSK	Phase Shift Keying
PWM	Pulsweitenmodulation
RAM	Random-Access Memory
ReST	Representational State Transfer
RFC	Request for Comment
RNDIS	Remote Network Driver Interface Specification
ROM	Read-Only Memory
RPC	Remote Procedure Call
RSS	Rich Site Summary
RTC	Realtime Clock
SAX	Simple API for XML
SRAM	Static Random Access Memory
SSID	Service Set Identifier
SGML	Standard Generalized Markup Language
SHDP	Smart Home Device Control Protocol
SMT	Surface Mounted Technology
SNR	Signal-to-Noise-Ratio
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SQL	Structured Query Language
TCP	Transmission Control Protocol
TTL	Transistor-to-Transistor Logic
TWI	Two-Wire Interface
USART	Universal Serial Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UHCI	Universal Host Controller Interface
URI	Unified Resource Identifier
URL	Unified Resource Locator
USB	Universal Serial Bus

VPN	Virtual Private Network
WADL	Web Application Description Language
WAN	Wide Area Network
WiFi	Wireless Local Area Network
WLAN	Wireless Local Area Network
WWW	World Wide Web
WS4D	Webservices for Devices
WSDL	Webservice Description Language
XML	Extensible Markup Language
XMLRPC	Extensible Markup Language Remote Procedure Call
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation