MNM-Team (Hrsg.)

Seminarband
„Eingebettete Kommunikationssysteme“ SS 2017

14. – 15. Juli 2017
München, Deutschland

Gesellschaft für Informatik e.V. (GI)

Vorwort

Eingebettete Systeme (engl. Embedded Systems) werden ein immer zentralerer Bestandteil des alltäglichen Lebens. Seien es Kühlschränke und Heizungen in Smart Homes oder Ampeln, Züge und Autos in Smart Cities, diese "Dinge" sind ein fester Bestandteil unseres Alltags geworden. Die globale Vernetzung dieser "intelligenten Dinge" wird gerne als das Internet der Dinge (engl. Internet of Things, IoT) dargestellt.

Die zumeist eingeschränkten Ressourcen solcher eingebetteten Systeme (z.B. Rechenleistung, Speicher, Energie) machen die Verwendung einschlägiger Kommunikationsprotokolle schwierig oder gar unmöglich, was Forscher und Ingenieure vor neue Herausforderungen stellt. Eine der größten Herausforderung neben der eigentlichen Vernetzung ist die Sicherheit, denn diese "Dinge" beeinflussen nicht nur "virtuelle Ressourcen", wie Bankkonten oder Webseiten in deren (kritischen) Umgang wir mittlerweile geübt sind, sondern können unsere „physische“ Umgebung oder gar uns selbst beeinflussen.

Dieser Seminarband gibt einen Überblick über aktuelle Systeme, Programmiersprachen, Protokolle und IT-Sicherheit und setzt sich kritisch damit auseinander.

München, Juli 2017

Tagungsleitung

Leitung des Programmkomitees:	Prof. Dr. Dieter Kranzlmüller, LMU München
	Prof. Dr. H.-G. Hegering (em.)

Programmkomitee

Tobias Adolph	Leibniz-Rechenzentrum
Marcel Breuer	Leibniz-Rechenzentrum
Nils gentschen Felde	LMU München
Tobias Fuchs	LMU München
Karl Fürlinger	LMU München
Tobias Guggemos	LMU München
Sharique Javaid	Leibniz-Rechenzentrum
Bastian Kemmler	Leibniz-Rechenzentrum
Roger Kowalewski	LMU München
Matthias Maiterth	LMU München / Intel
Stefan Metzger	Leibniz-Rechenzentrum
Thomas Odaker	Leibniz-Rechenzentrum
Jan Schmidt	LMU München
David Schmitz	Leibniz-Rechenzentrum
Cuong Ngoc Tran	LMU München
Markus Wiedemann	Leibniz-Rechenzentrum
Jule Ziegler	Leibniz-Rechenzentrum

Organisationsteam

Nils gentschen Felde	LMU München
Tobias Guggemos	LMU München

Inhaltsverzeichnis

Ad-Hoc Netze

Sicherheit

Technischer Datenschutz

Neue Trends in eingebetteten Kommunikationssystemen

Autorenverzeichnis

Begriffe und Hardware

Übersicht & Vergleich verschiedener Plattformen und Architekturen

Robert Pospisil[1]

Abstract: Die Bedeutung eingebetteter Systeme sowie deren Verwendung steigen nach wie vor an. Die vorliegende Seminararbeit liefert einen Überblick über die in eingebetteten Systemen verwendete Hardware und kann als Einstieg in das Thema genutzt werden. Dies beinhaltet deren grundlegenden Aufbau, spezielle Anforderungen an diese sowie Unterschiede zwischen verschiedenen Systemen. Des Weiteren ist eine kurze Vorstellung einiger Produkte verschiedener Hersteller enthalten. Abschließend wird auf die Vergleichbarkeit von unterschiedlichen Systemen eingegangen.

Keywords: Eingebettete Systeme; Mikrocontroller; Überblick; ARM Cortex; Atmel AVR; Infineon TriCore; Intel

1 Einleitung

Mark Weiser spricht in seinem Aufsatz „The Computer for the 21st Century" im Jahr 1991 von dem Begriff „Ubiquitous computing", allgegenwärtige Computer [We91]. In diesem beschreibt er die Entwicklung von Computern zu immer kleiner werdenden Systemen, die sich in die Welt der Menschen integrieren und untereinander automatisch kommunizieren. Schlussendlich verschwinden Sie aus der Wahrnehmung der Menschen. Eine ähnliche Entwicklung ließ sich schon bei vielen Technologien beobachten; ein Beispiel wäre die Stromversorgung, die aus dem alltäglichen Blickfeld verschwunden ist. Auch wenn seine Vision noch nicht vollständig eingetroffen ist, so sind kleine, oft unsichtbare Computersysteme aus dem Alltag der meisten Menschen nicht mehr wegzudenken.
Ein großer Teil dieser verborgenen Computersysteme besteht aus sogenannten eingebetteten Systemen. Betrachtet man den Markt für Prozessoren, so lag der Anteil für eingebettete Systeme bereits im Jahr 2002 bei 98% und lediglich 2% entfielen auf reguläre PCs, vgl. [Tu02]. In der heutigen Zeit werden eingebettete Systeme in immer mehr Produkten integriert. Auch in Zukunft wird der Bedarf nach immer intelligenteren Gegenständen nicht abnehmen. Seien es autonom agierende Fahrzeuge oder auch Smarte Kleidung, sie alle sind genau so wie viele andere Produkte der heutigen Zeit in ihrer „smarten" Form ohne eingebettete Systeme undenkbar. Bereits jetzt handelt es sich bei dem Begriff „eingebettetes System" um ein sehr umfassendes Gebiet, für welches in dieser Arbeit ein Überblick für

[1] Ludwig-Maximilians-Universität, Kommunikationssysteme und Systemprogrammierung, Oettingenstr. 67, 80538 München, Deutschland pospisil@cip.ifi.lmu.de

den Einstieg in das Thema gegeben wird.
Zunächst bedarf es einer genaueren Abgrenzung. Marwedel definiert ein eingebettetes System beispielsweise folgendermaßen: „Eingebettete Systeme sind informationsverarbeitende Systeme, die in ein größeres Produkt integriert sind und die normalerweise nicht direkt vom Benutzer wahrgenommen werden.“ [Ma08, 1]
Auf den nachfolgenden Seiten werden zunächst einige Grundlagen zu eingebetteter Hardware gegeben. Dies beinhaltet aus welchen Komponenten eingebettete Systeme bestehen, welche gesonderten Anforderungen an die Hardware dieser gestellt wird und wie sich diese unterscheiden lassen. Anschließend werden einige Architekturen kurz vorgestellt und abschließend auf die Vergleichbarkeit verschiedener Systeme eingegangen.

2 Grundlagen

Zunächst sollte geklärt werden aus welchen Komponenten sich ein eingebettetes System zusammensetzt. Hierfür wird im speziellen auf den Aufbau eines Mikrocontrollers eingegangen. Anschließend werden einige wichtige Anforderungen an eingebettete Systeme aufgelistet sowie weitere Unterschiede aufgezeigt.

2.1 Grundlegender Aufbau eines Mikrocontrollers

Ein eingebettetes System besteht in der Regel aus einem Mikrocontroller, im Falle eines sehr kleinen Systems auch nur aus einem Mikroprozessor.

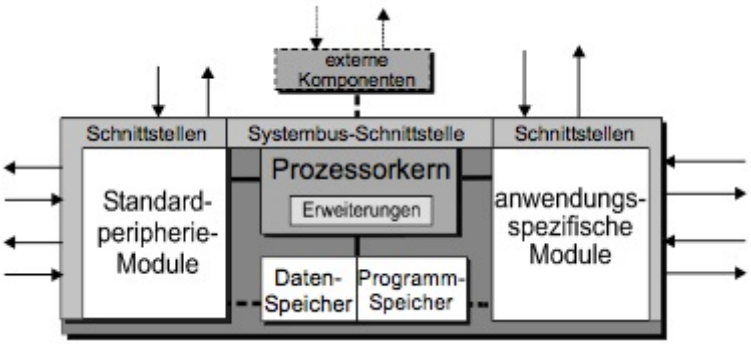


Abb. 1: Grober Aufbau eines Mikrocontrollers [Bä10, 17] Abb. 2.9

Der grundlegende Aufbau eines Mikrocontrollers lässt sich wie in Abbildung 1 darstellen. Bei einem Mikrocontroller handelt es sich um einen Mikrorechner auf einem einzelnen Chip. Dieser setzt sich unter anderem aus einem oder mehreren Mikroprozessoren sowie Arbeitsspeicher (Programm und Datenspeicher) und Ein/Ausgabeschnittstellen (im weiteren

auch Module genannt) zusammen. Unter einem Mikroprozessor versteht man die Zentraleinheit eines Digitalrechners. Er setzt sich aus einem Prozessorkern sowie Schnittstellen zur Außenwelt zusammen. Ein Prozessorkern stellt sich schlussendlich aus einem Steuerwerk (zeitlicher Ablauf) und einem Rechenwerk (arithmetische/logische Operationen) zusammen, vgl. [Bä10, 11] und [BU10, 1ff].

Zusammenfassend besteht ein Mikrocontroller aus:

- dem Prozessorkern(en) und eventuellen Erweiterungen,
- Programm- und Datenspeichern,
- Standardperipherie-Modulen,
- anwendungsspezifischen Modulen

Oft wird für den Prozessorkern ein weit verbreiteter Mikroprozessor genutzt. Standardperipherie-Module beziehen sich auf Komponenten, die häufig in Mikrocontrollern zu finden sind. Dazu zählen unter anderem Timer Module, Interrupt-Controller und digitale sowie analoge Ein-/Ausgabe Ports. Anwendungsspezifische Module finden sich im Gegensatz dazu nur bei Mikrocontrollern für bestimmte Bereiche wie im Automobilsegment. Häufig handelt es sich dabei um anwendungsspezifische Schnittstellen wie den CAN-Bus, vgl. [Bä10, 17].

2.2 Anforderungen an eingebettete Hardware

In diesem Abschnitt werden einige der wichtigeren Anforderungen für eingebettete Systeme kurz vorgestellt. Brinkschulte und Ungerer beschreiben in 'Mikrocontroller und Mikroprozessoren' verschiedene Anforderungen an eingebettete Systeme [BU10, 8ff]. Hierzu zählen:

- Anforderungen an die Schnittstellen
- Anforderungen der Umgebung
- Anforderungen an die Zuverlässigkeit
- Zeitanforderungen
- Sonstige Anforderungen

Eingebettete Systeme benötigen oft einen erweiterten Umfang und eine größere Vielfalt an Ein-/Ausgabeschnittstellen im Vergleich zu herkömmlichen Computersystemen. Um mit ihrer Umwelt interagieren zu können müssen sie verschiedenste Sensoren und Aktuatoren überwachen und bedienen können.

Abhängig von der Umgebung ihres Einsatzgebietes treten für eingebettete Systeme unter anderem erhöhte mechanische, elektrische sowie thermische Anforderungen auf. Der Einsatz in raueren Umgebungen wie Fahrzeugen oder Fabrikhallen stellt höhere mechanische Anforderungen an die Systeme als in anderen Gebieten. Um den Belastungen standzuhalten müssen die Systeme deutlich robuster sein und beispielsweise Stöße und Vibrationen dauerhaft widerstehen können. Auch die vorhanden Platzverhältnisse sowie geometrischen Begebenheiten müssen beachtet werden. In der Regel begrenzt die Umgebung den maximalen Energieverbrauch sowie die Versorgungsspannung. Häufig müssen sie in bereits existierende Strukturen eingebettet werden und mit den dortig verfügbaren Ressourcen auskommen. In einigen Einsatzgebieten können extremere Temperaturen auftreten. Diese muss die Hardware aushalten. Zudem macht der Einsatz in zum Beispiel besonders isolierten Gehäusen den Abtransport von Abwärme schwierig. Als Beispiel für Einsatzgebiete mit hohen Temperaturanforderungen sind der automotive oder auch industrielle Bereich zu nennen. TriCore Prozessoren werden dort unter anderem eingesetzt. In der aktuellsten Ausführung sind sie in einem Temperaturbereich von -40°C bis +125°C einsetzbar.

Der Einsatz in gewissen Gebieten birgt hohe Anforderungen an die Zuverlässigkeit und Ausfallsicherheit. So hätte beispielsweise das Versagen der Steuerung einer Bremsanlage oder eines Kernreaktors katastrophale Konsequenzen. Es muss sichergestellt werden, dass die Zuverlässigkeit maximal ist. Sollte es dennoch zu einem Ausfall kommen muss ein verlässlicher Notbetrieb möglich sein.

Einsatzgebiete wie die Hinderniserkennung eines automatisch gesteuerten Fahrzeugs erfordern eine Reaktion in einem festgelegten Zeitraum. In diesem Zusammenhang spricht man von Echtzeitsystemen. Diese Echtzeitsysteme lassen sich nochmals in drei Kategorien einteilen auf die nachfolgend kurz eingegangen wird; harte-, feste- und weiche Echtzeitsysteme. Harte Echtzeitsysteme definieren sich durch harte Zeitschranken, deren nicht einhalten, wie bereits angesprochen, zu schwerwiegenden Folgen führen kann. Diese sind in der Regel nicht mehr korrigierbar. Feste Echtzeitsysteme zeichnen sich dadurch aus, dass das Verpassen der Zeitschranke das Ergebnis wertlos macht, es jedoch zu keinen katastrophalen Folgen kommt. Ein Beispiel wäre die Positionserkennung, die bei einem automatischen Fahrzeug zu einer falschen/längeren Route führen könnte. Dies ließe sich unter Umständen später noch korrigieren. Bei weichen Echtzeitsystemen handelt es sich bei der Zeitschranke eher um eine Richtlinie als um eine Grenze. Sollte diese überschritten werden ist eine gewisse Abweichung noch akzeptabel. Als mögliches Beispiel könnte man die Anzeige eines Sensors betrachten, der regelmäßig aktualisiert wird. Sollte sich dies ab und an etwas verzögern ist dies innerhalb eines gewissen Akzeptanzbereiches tolerierbar. Für Echtzeitsysteme ist vor allem die exakte zeitliche Vorhersagbarkeit ausschlaggebend. Sollte eine Berechnung teilweise schneller fertig sein so muss trotzdem immer vom schlimmsten Fall ausgegangen werden. Eine weitere wichtige zeitliche Anforderung ist die langfristige, unterbrechungsfreie Verfügbarkeit der Leistung des Systems, auch ohne Zeit für Reorganisation für interne Datenstrukturen oder ähnliches.

Zu weiteren Anforderungen zählen beispielsweise ökonomische Anforderungen. Häufig sind die eingesetzten Systeme ein Massenprodukt und müssen somit möglichst kosteneffizient sein. Dabei gilt es zu beachten, dass alle nötigen Anforderungen erfüllt werden. Ein übertreffen der Anforderungen kann wiederum den Preis unnötig erhöhen, vgl. [Ud09, 12].

2.3 Unterschiede

Aufgrund der bereits vorgestellten Anforderungen ist es offensichtlich, dass sich die einzelnen eingebetteten Systeme je nach Einsatzgebiet stark unterscheiden können. In diesem Kapitel werden noch einige weitere Unterschiede aufgezählt und anhand von Echtzeitsystemen gezeigt welche Auswirkungen die obigen Anforderungen an die Hardware besitzen können.

Anhand der benötigten Leistung lässt sich bereits eine Einteilung in verschiedene Gruppen treffen. Hierzu lassen sich die Prozessoren anhand ihrer Wortbreite, Taktung und Kernanzahl unterscheiden. Wird nur wenig Performance benötigt so ist bereits ein kleiner 8-Bit Prozessor ausreichend. Für komplexere Anwendungen bedarf es ggf. auch einer 32-Bit Architektur oder wie heutzutage in vielen Smartphones üblich sogar 64-Bit mit mehreren Kernen und hohem Takt. Ebenfalls unterscheiden sich die einzelnen Systeme in der Speicherausstattung, sowie in der Art des verfügbaren Speichers. Dies betrifft sowohl Arbeitsspeicher als auch Festwertspeicher (ROM). Insbesondere bei letzterem gibt es je nach Bedarf unterschiedliche Varianten wie Flash Speicher, MROM, PROM, EPROM, EEPROM, vgl. [Ud09, 5].

Weitere Möglichkeiten sind die zugrunde liegende Architektur sowie der Befehlssatz. Dabei gibt es unter anderem Implementierungen, die der von Neumann, Harvard oder auch Berkley Architektur folgen, sowie ob die Befehlssätze eher dem RISC, CISC oder einem hybriden Ansatz nachgehen. Sollten Echtzeitanforderungen vorliegen sind unter Umständen gewisse Architekturbestandteile nicht so einfach implementierbar wie für andere Einsatzgebiete. Hierzu zählen zum Beispiel alle Bestandteile, die die Performance verbessern können, dies jedoch nicht garantieren. Darunter fallen unter anderem Caches, Pipelines oder auch Sprungvorhersagen. Im Fall von automatischen Caches kann nie angenommen werden, dass sich die benötigten Werte wirklich zum erforderlichen Zeitpunkt im Cache befinden. Eine Lösung für das Problem ist beispielsweise dem Programmierer Zugriff auf den Cache zu bieten, sodass diese gezielt eingesetzt werden können. Ähnlich verhält es sich mit Sprungvorhersagen, da diese lediglich spekulativ sind. Wie bereits beschrieben, ist bei echtzeitkritischen Systemen immer vom schlimmsten Fall auszugehen, somit sind für die Implementierung dieser Bestandteile ggf. zusätzliche Maßnahmen nötig, vgl.[SR94, 9f.].

3 Architekturen

Mikrocontroller werden meist in Familien organisiert. Gemeinsam ist den Mitgliedern der Familie in der Regel der Prozessorkern. Anhand anderer Komponenten wie Speicher, E/A Schnittstellen und weiteren Modulen unterscheiden sie sich. Durch verschiedene Kombinationen wird meist mit einer einzelnen Familie ein breiter Bereich abgedeckt. Nachfolgend werden einige relevante Architekturen von verschieden Herstellern betrachtet. Weiterführende Informationen finden sich in den Unterlagen der Hersteller.

3.1 ARM Cortex

Die ARM-Architektur von der gleichnamigen Firma wurde ursprünglich 1983 von der Firma Acron entwickelt und später als ARM Limited ausgelagert. Bei der Architektur handelt es sich um eine RISC 32-Bit Architektur, seit der neusten Generation ARMv8 aus dem Jahr 2011 64-Bit. ARM stellt selbst keine Prozessoren her, sondern lizenziert diese an andere Hersteller wie den später noch vorgestellten Herstellern, sowie im Smartphone und Tablet Markt tätigen Unternehmen wie Apple oder Qualcomm. Insbesondere im Markt für diese mobilen Geräte ist ARM sehr weit verbreitet. Die ARM Prozessoren lassen sich in 4 Serien gliedern: Cortex-A/R/M sowie SecurCore. Letztere ist speziell für sicherheitskritische Anwendungen. Innerhalb der einzelnen Cortex Serien gibt es weitere Unterteilungen in einzelne Familien. Diese sind teilweise wiederum für bestimmte Aufgaben konzipiert(Performance vs. Effizienz). In Tabelle 1 findet sich jeweils eine möglichst aktuelle Version zu der bereits Daten vorliegen.

Tab. 1: ARM Cortex Familien siehe [AR]

Familie	Bit	Architektur	Festwertspeicher (kBytes)	Schreiblesespeicher (kBytes)	Maximal Takt (MHz)
Cortex-A73	64	ARMv8-A	-	-	2.8G
Cortex-R5F[2]	32	ARMv7-R	4096	512	330
Cortex- M7[3]	32	ARMv7-ME	2024	384	300
SC-300 [4]	32	ARMv7-M	<2048	<50	60

Die Cortex-A Serie bietet die höchste Performance und wird beispielsweise bei Smartphones eingesetzt. Cortex-R ist für harte Echtzeitsysteme gedacht. Unter anderem werden sie im Automobil Segment oder im Storage Bereich eingesetzt. Cortex-M wurde wiederum für kleine stromsparende Mikrocontroller konzipiert. Sie werden häufig in kleinen mobilen Bereichen verwendet wie Smartwatches oder auch intelligenter Kleidung.

[2] Daten beziehen sich auf die Texas Instruments Hercules Cortex-R5F Mikrocontroller
[3] Daten beziehen sich auf die Atmel SAM E/S/V 70 Familien
[4] Daten beziehen sich auf die ST ST33 Familie

3.2 Atmel AVR

Die ersten Mikrocontroller der AVR Architektur, AT90s1200, wurden im Jahr 1997 eingeführt. Diese zeichneten sich unter anderem durch eingebaute Flash Speicher und kostenlose Entwicklertools aus. Bei den Prozessorkernen dieser Familie handelt es sich um 8-Bit RISC Kerne, die eine modifizierte Harvard Architektur nutzen. Später folgten noch angepasste 16-Bit Kerne sowie im Jahr 2006 eine AVR32 Bit Architektur. Diese verwendet eine andere Architektur. Controller dieser Familien finden sich zum Beispiel auf den ARDUINO Boards. Eine weitere Besonderheit der Architektur war die Anpassung an die höhere Programmiersprache C noch während der Entwicklung der Architektur, vgl. [My90]. Des Weiteren stellt Atmel noch Mikrocontroller basierend auf der ARM Architektur sowie der MSC51 Architektur von Intel her. In Tabelle 2 sind einige Informationen zu den Mikrocontrollerfamilien angegeben. Im 8-Bit Segment bietet Atmel die Reihen ATtiny und ATmega an. Die Mitglieder der ATtiny Reihe befinden sich am unteren Rand des Leistungsspektrums. Sie besitzen weniger Speicher und Schnittstellen im Vergleich zu den ATmega. Sie sind in Modellen von 0,5-16KB Flash Speicher erhältlich. Die Controller der Baureihe ATmega besitzen 4-256KB Flash Speicher. Der Prozessortakt beträgt bei beiden bis zu 20MHz wobei 1.0 MIPS pro MHz erreicht werden.

Tab. 2: AVR Mikrocontrollerfamilien siehe [At]

Familie	Bit	Festwertspeicher (kBytes)	Schreiblesespeicher (kBytes)	Serielle E/A (Anzahl)	Maximal Takt (MHz)
AVR 8	8	0,5-256	1/8-8	1-2	20
AVR 8/16	8/16	8-384	2-32	4-12	32
AVR 32	32	16-512	128	10	66

Der Hersteller gibt als Einsatzgebiete für beide Familien neben allgemeinen Tätigkeiten auch noch Lichtsteuerung an, bei der kleineren Familie außerdem noch Motorsteuerung und bei der größeren die Steuerung von LCDs. Im Bereich von 8/16 Bit gibt es die ATxmega Baureihe. Diese erreicht ebenfalls wieder 1.0 Mips pro MHz. Zum Einsatzgebiet macht der Hersteller dieselben Angaben wie bei der ATmega Serie. Die Controller der 32 Bit Familie erreichen 1.5 Mips pro MHz und besitzen als Einsatzgebiet ebenfalls allgemeine Tätigkeiten.

3.3 Infineon TriCore

Die erste Generation der TriCore Architektur brachte Infineon im Jahr 1999 unter der Bezeichnung AUDO (Automotive Unified-Processor) auf den Markt. Der Name beschreibt dabei die 3-in-1-Architektur. Diese kombiniert die Vorteile sowohl von RISC, MCU (Microcontroller Unit) als auch DSP (Digitaler Signalprozessor). Sie wurde überdies insbesondere für Echtzeitsysteme optimiert, vgl. [Te04, 313]. TriCore basierte Mikrocontroller werden neben der Automobilindustrie unter anderem auch in der Mess- und Regeltechnik eingesetzt.

Tab. 3: Infineon Tricore siehe [Ina]

Familie	Bit	Festwertspeicher (kBytes)	Schreiblesespeicher (kBytes)	Serielle E/A (Anzahl)	Maximal Takt (MHz)
TriCore 1.3.1	32	1M-4M	48-244	6-10	180
TriCore 1.6.1	32	512-8M	56-2,75M	6-10	300
TriCore 1.6.2	32	<16M	<6M	6-10	300

Die aktuellste Ausführung der Architektur ist die Version 1.6 von 2011. Diese wird in kleinere Unterarchitekturen aufgeteilt. Aktuell verfügbar ist die Version 1.6.1, die Version 1.6.2 befindet sich aktuell in Entwicklung. In Tabelle 3 sind einige Informationen zu den 3 Versionen angegeben. Bei der Version 1.6.1 gab es bis zu 3 Kerne. Bei Version 1.6.2 sind bis zu 6 geplant.

3.4 Intel

Eine der ersten Mikrocontrollerfamilien ist die MCS-48 von Intel aus dem Jahr 1976. Bei dieser handelt es sich um CISC-Prozessoren mit Harvard Architektur und 8-Bit. Bereits 1980 wurde die Nachfolgefamilie MCS-51 vorgestellt. Einer der bekanntesten Vertreter ist der 8051. Im 16-Bit Segment bot Intel die MCS-96 Familie an (8096). Auch wenn Intel seine komplette Mikrocontroller-Sparte im Jahr 2007 eingestellt hat, nach wie vor werden insbesondere die Controller der MCS-51 Familie von vielen verschiedenen Herstellern eingesetzt und produziert. Durch die Lizenzierung an andere Hersteller erreichte Intel eine sehr gute Marktverteilung.

Viele andere Hersteller bieten teils modifizierte Nachfolgearchitekturen an, zum Beispiel Infineon und Atmel. Diese zeichnen sich unter anderem durch eine deutlich gesteigerte Performance aus. Während beispielsweise bei der MCS-51 Architektur noch mindestens 12 Takte pro Instruction nötig waren sank dies auf teilweise lediglich eine. In Tabelle 4 finden sich einige Details zu den 8 und 16 Bit Familien sowie deren Nachfolger von Intel MCS251 sowie MCS296.

Tab. 4: Einige Intel Mikrocontrollerfamilien siehe [BU10, 92] und [Inb]

Familie	Bit	Festwertspeicher (kBytes)	Schreiblesespeicher (Bytes)	Serielle E/A (Anzahl)	Maximal Takt (MHz)
MCS-51	8	4-32	128-512	0-1	33
MCS-251	8	0-16	512-1024	1-2	24
MCS-96	16	8-56	232-1.5k	1-2	50
MCS-296	16	-	2k	1	50
Quark	32	32-384	8k-80k	3	32

Im Bereich von 32-Bit hatte Intel die PXA-Sparte im Angebot. Diese beruht auf der ARM Architektur und wurde an die Firma Marvel verkauft. Aktuell bietet Intel Mikrocontroller

basierend auf ihrer x86 Architektur an. Hierzu zählen unter anderem Quark und Atom Prozessoren. Letztere versuchte Intel angepasst auch im Smartphone Markt zu etablieren. Dies wurde jedoch aufgegeben. Atom Prozessoren werden weiterhin in kleinen Laptops oder auch Windows Tablets eingesetzt.

4 Vergleichbarkeit

Ein Vergleich verschiedener Mikrocontroller ist beispielsweise auf Grund der in den Abschnitten 2.2 und 2.3 vorgestellten Anforderungen und Unterschiede in der Regel nur in einem ähnlichen Einsatzgebiet sinnvoll, da die Auswirkungen dieser sehr stark sein können. Anhand der im Abschnitt 3 gezeigten Daten sowie weiteren Daten der Hersteller lassen sich die Architekturen bereits grob miteinander vergleichen.

Ähnlich wie bei anderen Produkten sollten die Mikrocontroller alle nötigen Eigenschaften und Anforderungen erfüllen und dabei möglichst kostengünstig sein. Insbesondere die Leistung des Mikrocontrollers spielt dabei eine wichtige Rolle. Zum Vergleich dieser bieten sich zum einem typische Benchmarks an. Hierbei sollte der spätere Einsatzzweck (Programmcode) beachtet werden, damit möglichst aussagekräftige Ergebnisse erreicht werden. Für den Bereich von eingebetteten Systemen gibt es beispielsweise extra angepasste Benchmarks von EEMBC [EE] oder auch den freien MiBench [Mi], vgl.[BU10, 16]. Häufig gibt es jedoch recht wenig frei zugängliche Benchmarkergebnisse und nur wenige Hersteller veröffentlichen selbst Ergebnisse. Sollten Ergebnisse veröffentlicht werden so handelt es sich dabei meist um die prozentuale Verbesserung der neuen Architektur zur vorherigen Architektur eines Herstellers. Mit diesen Informationen ist ein Vergleich unterschiedlicher Hersteller jedoch schwierig.

Eine weitere Möglichkeit sind verschiedene analytische Methoden. Als Maßzahlen lassen sich unter anderem MIPS (Million Instructions Per Second) und MFLOPS (Million Floatingpoint Operations Per Second) verwenden, vgl.[BU10, 12f.]. Insbesondere erstere werden von vielen Herstellern angegeben, meist als Mips pro MHz. Sollten die verfügbaren Informationen nicht ausreichend sein, bieten viele Hersteller Test- oder auch Evaluierungsboards an, anhand von diesen können, sofern nötig, eigene Vergleiche erstellt werden.

5 Zusammenfassung

Die vorliegende Arbeit beschreibt den allgemeinen Aufbau von eingebetteten Systemen und gibt zusätzlich einen Überblick zu aktueller Hardware aus diesem Bereich. Hierzu wurden zu Beginn einige Komponenten eingebetteter Systeme betrachtet. Der Fokus lag dabei auf Mikrocontrollern sowie deren Bestandteile. Nachfolgend wurden einige Anforderungen an Mikrocontroller vorgestellt. Hierzu zählen Anforderungen an die Schnittstellen, Anforderungen der Umgebung (mechanische, elektrische, thermische), Anforderungen an die

Zuverlässigkeit, zeitliche Anforderungen (Echtzeitsysteme, zeitliche Verfügbarkeit). Die Erfüllung dieser Anforderungen kann die Architektur der Systeme beeinflussen. Dies wurde im anschließenden Abschnitt am Beispiel von Echtzeitsystemen aufgezeigt. Zudem wurden weitere mögliche Unterschiede erläutert. Diese umfassen unter anderem die grundlegende Architektur, Art und Menge der Speicher sowie allgemein die Leistung.

In Abschnitt 3 wurden Architekturen von verschiedenen Herstellern (ARM, Atmel AVR, Infineon TriCore, Intel) betrachtet. Abschließend wurde auf die Vergleichbarkeit verschiedener Architekturen eingegangen. Neben einem Vergleich über Herstellerdaten bieten sich auch Leistungsvergleiche über Benchmarks und andere analytische Methoden wie Maßzahlen an. Diese werden jedoch nicht von jedem Hersteller veröffentlicht. Zudem gilt es zu beachteten für eingebettete Systeme bzw. für das gewünschte Szenario passende Kriterien zu wählen, um aussagekräftige Ergebnisse zu erhalten. Ein Vergleich für verschiedene Einsatzgebiete konzipierte Systeme besitzt ggf. wenig Aussagekraft.

Die Bedeutung von eingebetteten Systemen wird in Zukunft noch weiter steigen insbesondere im Hinblick auf Themen wie Internet of Things wodurch immer mehr Produkte mit Mikrocontrollern und ähnlicher Hardware ausgestattet werden. In diesem Zusammenhang spielt unter anderem die Kommunikation der verschiedenen Systeme und deren Sicherheit eine wichtige Rolle. Dies wird auch von vielen Experten als ein aktueller Knackpunkt in Mark Weiser Vision angesehen. Eine weitere Entwicklung im Bereich eingebetteter Systeme ist der ähnlich wie aktuell im Desktop Bereich zu beobachtbare Trend zu immer mehr Prozessorkernen.

Literaturverzeichnis

[AR] ARM: . https://www.arm.com/products/processors (Stand 23.05.17).

[At] Atmel: . http://www.atmel.com/products/microcontrollers/avr/default.aspx (Stand 23.05.17).

[Bä10] Bähring, Helmut: Anwendungsorientierte Mikroprozessoren. Springer Berlin Heidelberg, 2010.

[BU10] Brinkschulte, Uwe; Ungerer, Theo: Mikrocontroller und Mikroprozessoren. Springer Berlin Heidelberg, 2010.

[EE] EEMBC: . http://www.eembc.org (Stand 06.06.2017).

[Ina] Infineon: . http://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-tm-microcontroller/channel.html?channel=ff80808112ab681d0112ab6b64b50805 (Stand 23.05.17).

[Inb] Intel: . http://www.intel.de/content/www/de/de/embedded/products/quark/overview.html (Stand 23.05.17).

[Ma08] Marwedel, Peter: Eingebettete Systeme. Springer, 2008.

[Mi] MiBench: . http://vhosts.eecs.umich.edu/mibench// (Stand 06.06.2017).

[My90] Myklebust, Dr. Gaute: The AVR Microcontroller and C Compiler Co-Design. ATMEL Corporation, 1990.

[SR94] Shin, K.G.; Ramanathan, P.: Real-time computing: a new discipline of computer science and engineering. Proceedings of the IEEE, 82(1):6–24, 1994.

[Te04] Technologies, Infineon: Semiconductors E2. Publicis, 2004.

[Tu02] Turley, Jim: , The Two Percent Solution, 12 2002.

[Ud09] Udayashankara, V: 8051 Microcontroller: Hardware, Software & Applications. Mcgraw Hill Education (India) Private Limited, 2009.

[We91] Weiser, Mark: The Computer for the 21st Century. Scientific American, 265(3):66–75, January 1991.

Betriebssysteme und Programmiersprachen

Eingebettete Betriebssysteme - Unterschiede und Herausforderungen gegenüber „herkömmlichen" Betriebssystemen

Tobias Treutner[1], Manuel Nagler[2]

Abstract: Die Anforderungen, die an ein eingebettetes Betriebssystem gestellt werden, unterscheiden sich stark von den Anforderungen an ein normales Betriebssystem. In dieser Arbeit gehen wir zunächst auf die einzelnen Anforderungen ein und zeigen anschließend auf, wie die Betriebssysteme TinyOS, Contiki und RIOT an diese herangehen. Zum Schluss werden wir die Einsatzbereiche dieser drei eingebetteten Betriebssysteme eingrenzen.

Keywords: RIOT, Contiki, TinyOS, Eingebettete Betriebssysteme

1 Einleitung

Betriebssysteme für das Internet of Things (IoT) müssen andere Anforderungen erfüllen als Betriebssysteme, die wir von unserem Heimcomputer kennen. Häufig sind „Computer" im IoT nicht an das Stromnetz angeschlossen, sondern laufen im Batteriebetrieb. Sie verfügen nur über eine geringe Menge an Speicher und mit Taktraten von nur einigen MHz sind sie auch um ein Vielfaches langsamer als ihre großen Vorbilder. Ressourcenintensive Berechnungen wie das Rendern von Videos sind aber auch gar nicht Aufgabe der Computer im IoT. Vielmehr werden sie z. B. zur Bildung von Sensornetzen verwendet oder in Smart Home Anwendungen eingesetzt, um kleinere Aufgaben zu erledigen oder Sensoren auszulesen. Hier liegt der Fokus vielmehr auf Sicherheit, sparsamem Umgang mit Speicher und Energie, sowie Hardwareunabhängigkeit. In dieser Arbeit beschränken wir uns auf Betriebssysteme, die in Sensornetzen eingesetzt werden.

2 Anforderungen an ein eingebettetes Betriebssystem

In diesem Kapitel beschreiben wir kurz die Anforderungen an ein Betriebssystem hinsichtlich der Sicherheit, Ressourcenverbrauch, Scheduling, Multithreading, Netzwerkfähigkeit und Hardwareunabhängigkeit.

[1] LMU München treutner@cip.ifi.lmu.de
[2] LMU München manuel.nagler@campus.lmu.de

2.1 Sicherheit

In Sensornetzen spielt Sicherheit eine große Rolle. Meist stellt das Betriebssystem dem Kernel und den Anwendungen den gleichen Stack zur Verfügung. Programme für eingebettete Betriebssysteme werden meist in C oder C ähnlichen Programmiersprachen entwickelt. Vor allem in Multithreading Systemen stellen Zeigermanipulationen oder das Überschreiten von Arraygrenzen ein besonderes Risiko dar. [FAD10]

2.2 Ressourcenverbrauch

Sensornetze bestehen aus dutzenden Geräten, die über Funk miteinander kommunizieren. Bei diesen Geräten ist der Stromverbrauch ein entscheidender Faktor, da häufig Batterien als Energiequelle genutzt werden und deren Wechsel einen großen Aufwand bedeutet. Daher sollte das System ein Energiemanagement besitzen, um die MCU[3] und die Peripherie effizient in den Ruhemodus versetzen zu können. Programm- und Arbeitsspeicher sind aufgrund von Kosteneinsparungen und Platzmangel nur minimalistisch angelegt. [KSW16] Einige Betriebssysteme kommen schon mit weniger als 4 kB ROM und 1 kB RAM zurecht. [Ba13]

2.3 Scheduling

Der Scheduler bestimmt die Reihenfolge, in der einzelne Tasks der CPU zugewiesen werden. Zu den Zielen eines herkömmlichen Schedulers gehören die Maximierung des Durchsatzes und der Ressourcenauslastung, eine faire Rechenzeitzuteilung sowie die Minimierung der Latenz. Letzteres ist im Bezug auf Echtzeitfähigkeit von entscheidender Bedeutung. Nicht alle Betriebssysteme können eine deterministische Reaktionszeit garantieren und sind somit auch nicht echtzeitfähig. [FAD10]

2.4 Multithreading

In Multithreadingsystemen verursachen Kontextwechsel und Scheduling viel Overhead. [KC07] Da Sensorknoten mit Batterien oder Solarzellen betrieben werden, implementieren die einzelnen Betriebssysteme verschiedene Ansätze, um den Overhead möglichst gering zu halten und Energie zu sparen.

[3] Microcontroller Unit

2.5 Netzwerkfähigkeit

Auch in der Netzwerkübertragung ist die Energieeffizienz oft der ausschlaggebende Faktor. Die meisten Betriebssysteme unterstützen neben den gängigen Netzwerkprotokollen IPv4 und IPv6 den IEEE Standard 802.15.4. Dieser wurde extra für einen energieeffizienten Nachrichtenaustausch auf der Bitübertragungs- und MAC-Schicht für Sensornetze entwickelt. [IE13]

2.6 Hardwareunabhängigkeit

Auf dem Markt gibt es zahlreiche Mikrocontroller verschiedener Hersteller. Diese MCUs setzen sich aus einer CPU, Speicher und interner Peripherie zusammen. Einige Betriebssysteme laufen nur auf einer bestimmten Hardware, andere lassen sich durch Ändern des Quellcodes an neue Hardware anpassen. Eine weitere Anforderung an ein eingebettetes Betriebssystem ist die Portabilität, welche dem Aufwand und der Anzahl der neu zu codierenden Quellcodezeilen entspricht. [KSW16]

3 Betriebssysteme und deren Eigenschaften

In diesem Kapitel werden wir die Betriebssysteme Contiki, TinyOS und RIOT betrachten und darauf eingehen, wie diese an die einzelnen Anforderungen aus dem vorherigen Kapitel herangehen.

3.1 Contiki

Die Idee des schwedischen Programmierers Adam Dunkel war es, ungewöhnliche Dinge mit dem Internet zu verbinden. Daher entwickelte er im Jahr 2001 den uIP Stack, eine leichtgewichtige Implementierung des TCP/IP Stacks für Mikrocontroller. Da der uIP Stack aber in vielen eingebetteten Systemen nicht benutzt wurde, brachte er 2003 das Open Source Embedded System Contiki heraus. [Th] Contiki wurde in C implementiert und ist ein ereignisbasiertes Betriebssystem. Es setzt sich aus Kernel, Libraries, Program Loader und verschiedenen Prozessen zusammen. Ein Prozess ist entweder eine Anwendung oder ein Service, welcher Dienste für andere Anwendungen bereitstellt. Einer der großen Vorteile von Contiki gegenüber anderen eingebetteten Betriebssystemen ist das Laden, Entladen oder Updaten von Prozessen zur Laufzeit. Das System muss weder neu gelinkt noch neugestartet werden. [DGV04]

3.1.1 Sicherheit

Contiki beinhaltet keine eigenen Sicherheitsmechanismen, die eine Anwendung bezüglich Speicherzugriffsverletzungen prüfen. Der Source Code wird unter einer 3-clause BSD-style Lizenz vertrieben. Es ist jedem eine eigene Abwandlung des Quellcodes erlaubt[4]. Prozesse können nicht direkt kommunizieren, sondern der Nachrichtenaustausch muss immer über den Kernel erfolgen.

3.1.2 Ressourcenverbrauch

Eine Minimalinstallation kommt mit knapp 2 kB RAM und 30 kB ROM zurecht und benötigt im Vergleich zu seinen Konkurrenten TinyOS und RIOT erheblich mehr Speicher. Dafür beinhaltet schon die Installation einige Features, z. B. GUI, Multithreading, Libraries, TCP/IP und IPv6. Contiki ist ein ereignisgesteuertes Betriebssystem, welches sehr sparsam mit dem Arbeitsspeicher umgeht, da alle Threads auf nur einem Stack arbeiten. Des Weiteren kann auf einen Locking Mechanismus verzichtet werden, der die einzelnen Stacks vor einem unbefugten Zugriff schützt. Der Energiesparmodus wird in Contiki durch den anwendungsspezifischen Teil des Systems implementiert. Der Eventscheduler stellt für die einzelnen Anwendungen Informationen zur Länge der Eventqueue bereit. Falls die Warteschlange leer ist, fahren diese dann den Prozessor herunter. [DGV04]

3.1.3 Scheduling

Contiki verfügt über einen leichtgewichtigen Scheduler, der Prozesse zyklisch oder nach Eintreten eines Events aufruft, und einen Abrufmechanismus (polling mechanism). Ist ein Thread einmal dem Prozessor zugewiesen, kann er nicht mehr unterbrochen werden und läuft bis zum Ende durch[5]. Der Kernel unterscheidet zwischen asynchronen und synchronen Events. Asynchrone Events werden in die Queue des Schedulers eingereiht und zu einem späteren Zeitpunkt dem Zielprozess zugewiesen.

Synchrone Events hingegen werden sofort ausgeführt, sodass der Zielprozess sofort dispatched wird. Nach der Ausführung des Eventhandlers des Zielprozesses wird der vorherige Prozess weiter ausgeführt.

Der Abrufmechanismus kann hoch priorisierte Events vor der Ausführung eines asynchronen Events einreihen. Dies wird in der Regel für hardwarenahe Prozesse verwendet. [DGV04]

[4] `http://www.contiki-os.org/license.html`
[5] ausgenommen Protothreads - siehe Multithreading

3.1.4 Multithreading

Für eingebettete Betriebssysteme gibt es in der Regel zwei Entwicklungsansätze, Threadbasiert und Ereignisgesteuert.

Threadbasiert Da der Stack jedes Threads im Hauptspeicher abgelegt werden muss, ist aufgrund der Speicherknappheit eines Mikrocontrollers nur eine beschränkte Anzahl an gleichzeitigen Threads möglich. Die Vergrößerung des RAMs ist keine Option, da dies erhöhte Hardware- und Energiekosten zur Folge hätte.

Ereignisgesteuert Ereignisgesteuerte Systeme haben zwar nur einen globalen Stack, doch muss der gesamte Programmablauf als Zustandsmaschine betrachtet werden. [DGV04]

Contiki verwendet Protothreads, eine Mischung aus beidem. Mit Hilfe einer Library wird Multithreading auf den ereignisbasierten Kernel aufgebaut. Protothreads haben den Vorteil, dass keine weiteren Stacks angelegt werden müssen und nur ein Overhead von zwei Bytes pro Thread entsteht. Doch es handelt sich um kein echtes preemptives Multithreading, da dem Programm Sprungmarken hinzugefügt werden müssen, an denen es unterbrochen werden kann. [Du06]

3.1.5 Netzwerkfähigkeit

Contiki unterstützt neben den gängigen Netzwerkprotokollen IPv4 und Ipv6 auch die speziell für Sensornetze entwickelten Protokolle RPL[6][Wi12] und den uIP Stack. uIP ist eine reduzierte Version von TCP/IP und ermöglicht den Nachrichtenaustausch mit 8-Bit Mikrocontrollern. Es unterstützt das TCP, UDP und das ICMP Protokoll. Um Speicher einzusparen, werden alle Nachrichten in einen globalen Puffer zwischengespeichert. Das jeweilige Programm wird von Contiki über TCP/IP benachrichtigt, sodass die Nachricht in seinem lokalen Speicher gesichert werden kann. [DGV04]

3.1.6 Portabilität

Contiki wurde bereits auf einige Hardwareplattformen, u. A. MSP430 oder Atmel AVR, portiert. [Co] Nach Angaben des Herstellers dauerte die Portierung für den Zilog Z80 nur einen Tag, die für den Atmel AVR sogar nur einige Stunden. *Aufgrund der Aufteilung des Contikisystems müssen nur der Bootcode, die Gerätetreiber, der gerätespezifische Teil des Program Loaders und der Stackswitching Code der Multithreading Library umgeschrieben werden. Für den Kernel und die Serviceschicht bedarf es keiner Änderung.*[7] [DGV04]

[6] Routing Protocol for Low power and Lossy Networks
[7] eigene Übersetzung

3.2 TinyOS

TinyOS war eines der ersten Betriebssysteme für Sensornetze und wurde im Jahr 2000 von UC Berkeley entwickelt. Anfangs wurde das System nur von einigen wenigen Akademikern zur Wireless Sensor Network Forschung verwendet und heruntergeladen. Heutzutage ist TinyOS eines der am häufigsten verwendeten eingebetteten Betriebssysteme mit etwa 25 000 Downloads pro Jahr. TinyOS und dessen Programme werden in nesC, einem low-level C-Dialekt, implementiert. [Le12] Es hat in der Minimalversion einen sehr geringen Speicherverbrauch von unter 300 Bytes RAM und 4 kB ROM. Des Weiteren ist es mit zahlreichen Werkzeugen (u. A. zur Datenerfassung), Sensoren, Treibern und Netzwerkprotokollen ausgestattet. [FAD10]

3.2.1 Sicherheit

TinyOS verfügt über keine Schutzmechanismen bezüglich der Speichersicherheit. Jedoch gibt es den Sicherheitscompiler „Save TinyOS“, der auf dem C-Compiler Deputy beruht. Save TinyOS prüft den Code beim Kompilieren auf Typsicherheit sowie auf Zeiger- und Arrayfehler. Des Weiteren fügt es Überprüfungsroutinen in den Programmcode ein, mit denen Fehler zur Laufzeit festgestellt und behoben werden können. [FAD10] Der Source Code ist Open Source und wird unter einer New BSD Lizenz vertrieben. Die Abwandlung des Source Codes ist jedem gestattet und muss nicht offengelegt werden. [FA]

3.2.2 Ressourcenverbrauch

Mit 300 Bytes RAM und 4 kB ROM ist TinyOS das kleinste der drei Betriebssysteme in unserem Vergleich. Es ist ein ereignisbasiertes Betriebssystem, welche weniger Arbeitsspeicher verbrauchen als threadbasierte Betriebssysteme. Um Energie zu sparen, können die Funkverbindung sowie der Prozessor heruntergefahren werden. Mit dem Befehl HPLPowerManagement.Enable() wird der Energiesparmodus aktiviert. Dadurch wird der Prozessor so oft wie möglich bis zum nächsten Clock-Takt in den Ruhemodus versetzt.[PKK07]

3.2.3 Scheduling

In den ersten Versionen wurde TinyOS mit einem schlichten Scheduler nach dem FIFO-Prinzip ausgestattet. FIFO ist ein unfairer Scheduling Ansatz, da auch kurze Programmabläufe hinter längeren warten müssen. Deshalb ist in den neueren TinyOS Versionen[8] ein EDF (Earliest Deadline First) Scheduler implementiert. Da in TinyOS ein ereignisbasiertes

[8] ab Version 2.1

System umgesetzt wurde und Threads immer bis zum Ende ausgeführt werden, ist es trotz EDF kein echtzeitfähiges System. [FAD10]

3.2.4 Multithreading

Erst ab der Version 2.1 unterstützt TinyOS Multithreading. Die Threads werden TOS Threads genannt und basieren auf einem Package, das das Programmierkonzept von Threads mit der Effizienz eines ereignisbasierten Kernels vereinen soll. Application Level Threads können sich gegenseitig unterbrechen, Tasks und Interrupt Handler sind ununterbrechbar. Systemaufrufe werden nie durch die Threads selbst ausgeführt. Der Thread muss einen Task an den Kernelthread stellen, der dann den aktiven Thread unterbricht und einen Systemaufruf tätigt. Dies hat den Vorteil, dass nur der Kernel TinyOS Code ausführt. [FAD10] TOS Threads allokieren dynamisch Speicher fester Größe für die Thread Control Blocks auf dem Stack. Context Switches und Systemcalls sind sehr effizient und verursachen einen Overhead von weniger als 0,92%. [Kl09]

3.2.5 Netzwerkfähigkeit

Anfangs unterstützte TinyOS zwei Protokolle, Dissemination und TYMO. Dissemination ist vergleichbar mit einem Broadcast, der eine Nachricht an alle Netzwerkteilnehmer sendet. Es wird vor allem von Administratoren zur Netzwerkkonfiguration verwendet. TYMO basiert auf DYMO[9] und ist ein Routingprotokoll für mobile Ad-Hoc Netzwerke. [FAD10] Mit TinyOS Version 2 hielt das 6LoWPAN Protokoll Einzug. Diese implementiert den IPv6 Stack des IEEE 802.15.4 Standards für TinyOS.[Ha07]

3.2.6 Portabilität

TinyOS beruht auf einem drei-schichtigen Hardwareabstraktionsmodell, welches sich aus den Schichten

- Hardware Presentation Layer (HPL): Hardwarenahe Komponenten
- Hardware Adaptation Layer (HAL): Hardwarespezifische Abstraktionen
- Hardware Interface Layer (HIL): Hardwareunabhängige Anwendungen

zusammensetzt. Die drei Schichten bieten mehr Flexibilität als die Zweischichtenmodelle anderer Betriebssysteme und erreichen eine gute Balance zwischen Leistung und Portabilität.

[9] Dynamic MANET On-demand Routing

TinyOS läuft u. A. auf dem Atmel ATmega128, Texas Instruments MSP430 sowie dem Intel XScale PXA271 und kann von verschiedenen Plattformen aus programmiert werden. [Pl]

3.3 RIOT

RIOT hat seinen Ursprung im FeuerWhere Projekt der FU Berlin. Diese entwickelte 2008 ein echtzeitfähiges Betriebssystem, um Feuerwehrmänner im Einsatz zu überwachen. Es bildete sich ein Fork aus dem auch heute noch existierenden Betriebssystem FeuerWare, um IETF Protokolle in das System zu implementieren. Dadurch wurde es möglich, das Betriebssystem im Internet of Things zu verwenden. [RIa]

3.3.1 Sicherheit

Durch seinen ursprünglichen Einsatzzweck als Betriebssystem für eine Feuerwehrsoftware wurde RIOT als echter Mikro-Kernel implementiert, da so Fehler in Komponenten des Systems, welche nicht Teil des Kernels sind, nicht zum Absturz des gesamten Systems führen. Nur die wichtigsten Systemkomponenten wie Scheduling, Interprozesskommunikation und Synchronisierung sind Teil des Kernels. [WSS09] Des Weiteren ist RIOT LGPL lizensiert und der Quellcode ist auf Github einzusehen. Hierdurch wird die Sicherheit von RIOT noch weiter ausgebaut, da jeder Entwickler in den Quellcode blicken kann und so eventuell vorhandene Fehler oder sogar Sicherheitslücken schnell gefunden und behoben werden können. In Kombination mit der Echtzeitfähigkeit machen diese Eigenschaften RIOT zu einem sehr sicheren Betriebssystem für das IoT.

3.3.2 Ressourcenverbrauch

Mit einem Bedarf an etwa 1,5 kB RAM und ca. 5 kB ROM benötigt RIOT nur minimal mehr Speicher als TinyOS und ist damit ein sehr ressourcenschonendes Betriebssystem. [RIa] Dies wird mitunter durch den modularen Mikro-Kernel erreicht, da nicht benötigte Systemkomponenten eingespart werden können, was Speicher- und Energieverbrauch senkt. Der Energieverbrauch wird weiterhin durch RIOTs Scheduler gesenkt, der das gesamte System bei jeder Gelegenheit in den Ruhemodus versetzt und erst durch einen externen Interrupt wieder aktiviert werden kann. Sogar RIOTs Timer wurde auf Energieeffizienz optimiert. Dieser kommt ohne periodische Interrupts aus und stört dadurch den Ruhemodus des Systems nicht. [WSS09]

3.3.3 Scheduling und Multithreading

RIOT unterstützt präemptives Scheduling und Echtzeitbetrieb durch einfache Prioritätenvergabe. Jedem Task wird dazu eine Priorität zugewiesen. Der Scheduler stellt dann dem Prozess mit der höchsten Priorität die gesamte Rechenleistung des Systems zur Verfügung. Kommt ein neuer Task mit höherer Priorität hinzu, so wird dieser sofort bearbeitet. Wenn mehrere Tasks die gleiche Priorität haben, wird die Rechenleistung unter diesen gerecht aufgeteilt. Dadurch wird Multithreading äquivalent zu gängigen Betriebssystemen unterstützt.

Der Echtzeitbetrieb wird durch gezielte Prioritätenvergabe ermöglicht. Tiefere Systemkomponenten (z. B. Treiber) müssen mit einer höheren Priorität laufen als Benutzerprozesse. Durch dieses einfache System lassen sich nicht eingehaltene Fristen im Echtzeitbetrieb schnell auf zwei Ursachen eingrenzen. Entweder ist das System für den einzelnen Task bereits zu schwach dimensioniert, oder die Load des Systems war zu hoch und es mussten noch andere Tasks mit selbiger oder sogar höherer Priorität zur gleichen Zeit ausgeführt werden.

Mit der niedrigsten Priorität läuft ausschließlich der Idletask des Betriebssystems. Dieser versetzt RIOT in den Ruhemodus, falls dies momentan möglich ist. [WSS09]

3.3.4 Netzwerkfähigkeit

Alle gängigen Netzwerkstandards des IoT werden von RIOT unterstützt. Insbesondere der 6LoWPAN Standard, der auch für Contiki und TinyOS verfügbar ist, kann in RIOT verwendet werden. Dadurch wird eine energiesparende Netzwerkkommunikation möglich. Weitere unterstützte Protokolle sind IPv6, RPL und UDP. [RIa]

3.3.5 Portabilität

RIOT ist bereits für ein großes Spektrum an Plattformen verfügbar. Unter anderem lässt es sich unter Linux virtualisieren, wodurch das Entwickeln von Software für RIOT ohne spezielle Hardware ermöglicht wird. Im Gegensatz zu TinyOS und Contiki unterstützt RIOT sowohl C als auch C++ ohne Einschränkungen. [RIa] Benötigt man RIOT für eine noch nicht unterstützte Hardware, lässt sich das Betriebssystem einfach portieren. Durch den modularen Aufbau von RIOT kann ein Großteil auf neuen Systemen wiederverwendet werden. Einzig die hardwarespezifischen Teile des Betriebssystems, z. B. Pin-Belegungen, müssen auf die neue Plattform portiert werden. Anwendungssoftware kann problemlos weiterverwendet werden. [RIb]

4 Fazit und Schluss

Wie in Tabelle 1 erkennbar, unterscheiden sich die drei Betriebssysteme in unserem Vergleich stark in den unterstützten Features. RIOT ist in der Kategorie „unterstütze Features“ Spitzenreiter. Durch die Unterstützung von ANSI C und C++ müssen sich Programmierer nicht einschränken und es findet durch den modularen Aufbau auch auf kleinen Systemen Platz. Es ist als einziges Betriebssystem echtzeitfähig und erfüllt höchste Sicherheitsanforderungen. In sicherheitskritischen Anwendungen ist daher RIOT unsere erste Wahl.

	RAM	ROM	C	C++	Multithreading	MMU	Modularität	Echtzeitfähig
Contiki	<2 kB	<30 kB	Teilweise	Nein	Teilweise	Ja	Teilweise	Teilweise
TinyOS	<1 kB	<4 kB	Nein	Nein	Teilweise	Ja	Nein	Nein
RIOT	~1,5 kB	~5 kB	Ja	Ja	Ja	Ja	Ja	Ja

Tab. 1: Vergleich der Betriebssysteme[RIa]

Falls RIOT bereits zu viel Speicher verbraucht und auf dessen einzigartige Features verzichtet werden kann, ist durch TinyOS eine gute Alternative geboten. Allerdings muss man bei TinyOS auf Echtzeitfähigkeit verzichten und seinen Programmierstil aufgrund der Programmiersprache nesC stark anpassen. Da in Sensornetzen der Trend zu immer größeren abgedeckten Flächen und kleineren Sensorknoten geht, sollte TinyOS genau hier eingesetzt werden. Dadurch können die Kosten für große Sensornetze verringert werden.

Ist geringer Speicherverbrauch eher zweitrangig, kann auch Contiki verwendet werden. Mit 30 kB ROM verbraucht es allerdings sechsmal so viel Speicher wie RIOT in der Minimalversion und wird dadurch für viele Systeme schlichtweg zu groß. Der Grund für den vergleichsweise enormen Speicherverbrauch liegt hauptsächlich an Contikis fest integrierter Zusatzsoftware, darunter z. B. eine grafische Oberfläche. Desweiteren unterstützt Contiki als einziges dieser drei Betriebssysteme das laden und entladen von Prozessen zur Laufzeit. Werden solche Features in TinyOS oder RIOT ebenfalls benötigt, relativiert sich Contikis großer Speicherbedarf, wodurch es bedenkenlos eingesetzt werden kann.

Wir haben in dieser Arbeit festgestellt, dass für eingebettete Betriebssysteme viele verschiedene Anforderungen existieren. Nicht alle Betriebssysteme sind für jeden Einsatzzweck geeignet. Es muss auf der verwendeten Hardware lauffähig sein und sollte alle benötigten Features unterstützen. Es wurden in dieser Arbeit auch nicht alle Betriebssysteme behandelt, die für Sensornetze geeignet sind. Sollte keines der hier erwähnten Betriebssysteme infrage kommen, so stehen mit z. B. Linux oder auch ARM mbed OS weitere Alternativen zur Auswahl.

Literaturverzeichnis

[Ba13] Baccelli, E.; Hahm, O.; Gunes, M.; Wahlisch, M.; Schmidt, T. C.: RIOT OS: Towards an OS for the Internet of Things. In: 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). S. 79–80, April 2013.

[Co] Contiki Hardware. `http://www.contiki-os.org/hardware.html`, Zugegriffen am 08.06.2017.

[DGV04] Dunkels, A.; Gronvall, B.; Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: 29th Annual IEEE International Conference on Local Computer Networks. S. 455–462, Nov 2004.

[Du06] Dunkels, Adam; Schmidt, Oliver; Voigt, Thiemo; Ali, Muneeb: Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In: Proceedings of the 4th international conference on Embedded networked sensor systems. Acm, S. 29–42, 2006.

[FA] FAQ - TinyOS Wiki. `http://tinyos.stanford.edu/tinyos-wiki/index.php/FAQ`, Zugegriffen am 08.06.2017.

[FAD10] Farooq, Muhammad Omer; Aziz, Sadia; Dogar, Abdul Basit: State of the Art in Wireless Sensor Networks Operating Systems: A Survey. In (Kim, Tai-hoon; Lee, Young-hoon; Kang, Byeong-Ho; Ślęzak, Dominik, Hrsg.): Future Generation Information Technology: Second International Conference, FGIT 2010, Jeju Island, Korea, December 13-15, 2010. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 616–631, 2010.

[Ha07] Harvan, Matúš: A 6lowpan Implementation for TinyOS 2.0. 6. Fachgespräch Sensornetzwerke, 802:109, 2007.

[IE13] : IEEE Approved Draft Standard for Local and metropolitan area networks - Part 15.4: Low-Rate Wireless Personal Area Networks (WPANs). IEEE P802.15.4REVi/D09, April 2011 (Revision of IEEE Std 802.15.4-2006), S. 1–311, April 2013.

[KC07] Kim, Hyoseung; Cha, Hojung: Multithreading optimization techniques for sensor network operating systems. In: European Conference on Wireless Sensor Networks. Springer, S. 293–308, 2007.

[Kl09] Klues, Kevin; Liang, Chieh-Jan Mike; Paek, Jeongyeup; Musăloiu-E, Răzvan; Levis, Philip; Terzis, Andreas; Govindan, Ramesh: TOSThreads: Thread-safe and Non-invasive Preemption in TinyOS. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. SenSys '09, ACM, New York, NY, USA, S. 127–140, 2009.

[KSW16] Kietzmann, Peter; Schmidt, Thomas C.; Wählisch, Matthias: RIOT – das freundliche Echtzeitbetriebssystem für das IoT. In (Halang, Wolfgang A.; Unger, Herwig, Hrsg.): Internet der Dinge: Echtzeit 2016. Springer Berlin Heidelberg, Berlin, Heidelberg, S. 43–52, 2016.

[Le12] Levis, Philip: Experiences from a Decade of TinyOS Development. In: OSDI. S. 207–220, 2012.

[PKK07] Phani, AMRVA; Kumar, D Janakiram; Kumar, G Ashok: Operating systems for wireless sensor networks: A survey technical report. Department of Computer Science and Engineering., Indian Institute of Technology Madras, Chennai, India, 2007.

[Pl] Platform Hardware - TinyOS Wiki. http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware, Zugegriffen am 08.06.2017.

[RIa] RIOT - The friendly Operating System for the Internet of Things. http://riot-os.org/, Zugegriffen am 08.06.2017.

[RIb] RIOT OS - Porting Guide. https://github.com/RIOT-OS/RIOT/wiki/Porting-Guide, Zugegriffen am 08.06.2017.

[Th] The Contiki Community. http://www.contiki-os.org/community.html, Zugegriffen am 08.06.2017.

[Wi12] Winter, Tim: RPL: IPv6 routing protocol for low-power and lossy networks. 2012.

[WSS09] Will, H.; Schleiser, K.; Schiller, J.: A real-time kernel for wireless sensor networks employed in rescue scenarios. In: 2009 IEEE 34th Conference on Local Computer Networks. S. 834–841, Oct 2009.

Realtime Operating Systems

Christopher Schütze[1], Jonas Dellinger[2]

Abstract: Today, there are hundreds of operating systems which are focusing on different requirements. One type of operating systems are realtime operating systems (RTOS).
This paper describes how RTOS differentiates from other operating systems such as 'off-the-shell' UNIX distributions and which hard criteria make an operating system realtime compatible. We will clarify the meaning of *realtime* and what the differences are of hard-, soft- and firm-realtime in order to set a base for the comparison with non-RT operating systems.
Furthermore, choosing the right RTOS for a given requirement set is not an easy task. Various fields of industry need different sets of requirements. Each RTOS distribution implemented one or another realtime requirement in different way and for a different field of use. Thus it is a hard decision to select the most suitable RTOS for the given conditions.
With focus on hard realtime, we classify RTOS based on two characteristics, scheduling and realtime clock, to make them comparable to each other for the avionic and medical field of use.

Keywords: realtime operating systems, rtos, freertos, qnx, rtai, medical realtime, avionic realtime, scheduling, realtime clock

1 Introduction

Until today, a lot of different types of operating systems evolved. One of the most common in our everyday life, the multiprocessor system, focuses on performance and throughput by trying to divide tasks into parallelizable smaller tasks. This way of handling tasks is perfect for personal computers and servers. Aside of those, there also exist various embedded systems. Some obvious examples are smart-phones, washing machines and control units in modern cars. When look deeper into automobile components, there exist several safety-critical constraints which are not covered by a general purpose operating system (GPOS). For example, one of the components is the anti-lock braking system (ABS), which prevents the wheels from blocking during an emergency braking. When this system fails to respond within a predefined time span, it also fails to prevent the accident.
This time critical requirement leads us to a special kind of operating system (OS), which is called RTOS. By adapting system components such as the scheduling algorithm, it guarantees to process tasks and respond to incoming system events and interrupts as quickly as possible. So the task scheduler is one part how you can classify an OS, especially an

[1] Ludwig-Maximilian-Universität C.Schuetze@campus.lmu.de
[2] Ludwig-Maximilian-Universität J.Dellinger@campus.lmu.de

RTOS. Other characteristics are the supported platform architectures or how the system manages the resources such as memory allocation.
But the essential question is, what does realtime exactly mean for an RTOS and what are the crucial differences to a GPOS. The intuitive answer would be, that there is no delay in execution. Which leads us to the next question: How does an RTOS achieve this?
In the following sections we will have a deeper look at what realtime is, clear how we can differentiate a RTOS from a GPOS and what the benefits or drawbacks are. Furthermore, we will show by which characteristics we can classify an RTOS. In the trailing sections we will have a short look at some areas of application and select some requirements. Thus we can put a selection of well-known RTOS in order depending on the listed requirements and characteristics.

2 Background

This section will first introduce *realtime* and its signification. If you look up realtime, the first hits will get you to a description of realtime in detail and the differences between three different kinds of realtime, *hard*, *soft* and *firm*. So we pick that up and show how those kinds set apart. Afterwards we will compare a GPOS with RTOS by taking a deeper look at some requirements. This shows us why a GPOS is not suitable for an realtime use case. Finally, in this sections we will take a couple of characteristics, how an RTOS can be classified and how each characteristic can distinguish from on concrete RTOS implementation to another.

2.1 Realtime: Hard vs. Soft vs. Firm

There are three different kinds of realtime: *hard*, *soft* and *firm*. While all three follow the same idea of computing multiple tasks with a specified time constraint, they differ in handling missed deadlines. As soon as a deadline miss can cause catastrophic failures, it's called a hard deadline. Furthermore, as soon as a RT system contains at least one task with a hard deadline, it's considered a hard RT system. For example, a traffic light system should be treated as a hard RT system. If one light fails to switch to red in time, it could cause an accident, a catastrophic failure. On the contrary, a RT system is called soft when there is no task with a hard deadline. An example is multimedia streaming. The system should not fail to stream a media because some frames where unable to be processed. So instead of a system failure, it will more likely reduce your quality of service (QoS). To upkeep this QoS, soft RT systems introduce upper boundaries for missed deadlines. One of them could be *no more than x misses within time interval t*. In our multimedia example, we could also define *only* 10 *dropped frames every* 100 *frames*. Firm RT systems are equal to soft RT systems with one minor difference: If the result was overdue i.e. computed after its task's deadline , it's marked as invalid and will not be used in the future.
Because this paper will focus on hard realtime systems only, we will use the term realtime to refer to hard realtime in the following.

2.1.1 Operating System in general

A GPOS, such as Linux, MACOS or MS Windows suggests a bit different meaning of realtime. While we are using it, we don't want that any upcoming event (e.g. mouse pointer movement) blocks the current application. So the OS is optimized to parallelize every task and try to reach a maximum CPU workload. This will be achieved by best-fitting scheduling algorithms. The benefit is that we can run multiple applications *at the same time* and use them without the system hanging up. As more as the OS is able to run tasks (pseudo) parallel, the user perceives the OS as fast. Maybe the user would consider saying the OS is working in realtime. But, as we explained in the previous section, it is not. Because a GPOS is not optimized to handle critical incoming events which has to be processed immediately and has to return a result until a time deadline. It is focusing on fast response for *each event* which occurs.

2.1.2 OS quality criteria and performance features

Not surprisingly, both kinds of OS need to operate with high performance. But the requirements are different if we take a more detailed look at it. As is was mentioned before, it is very important for a GPOS to have a short response time. Which means that no matter how many processes are queued or in the ready-to-run state, the system must respond as quickly as possible to a request. Because a user doesn't want to wait for a requests' response for 5 seconds if he is expecting the response within half a second (e.g. quit a server connection). With respect to that, the system is handling events or rather interrupts in a fairly manner. Each queued process gets a fixed CPU time slice, so everything the system has to get done will be completed as *soon* as possible. This results in frequent process switches or context switches which take a few milliseconds (msec). Because of that, the compute time of a process could be spreaded over a minute even though it only needs a fraction of a minute. This is not really a problem if it is not necessary to get the process' result until a fixed deadline. Furthermore, it is acceptable to lose data as a consequence if a process is not completed within a excepted time. Obviously, we can summarize that this kind of scheduling behavior is not the best-fitting one for an RTOS. A OS does not only differentiate from an RTOS by the scheduling behavior, but also in some other provided (system) services. While for an OS a graphical user interface (GUI) is absolutely indispensable, it is just an optional feature for an RTOS. As a consequence, a big part of a GPOS gets cut off and the system gets stripped to become more compact and suitable for micro-controllers. But not only software parts are stripped, also hardware resources could be limited in comparison to a GPOS hardware setup (e.g. RAM, disk space). Finally, we can say, GPOS distributions are all-round solutions for the every day use. Where timing problems, blocking processes and limited resources gets handled by a GPOS easily, it could be a (huge) problem or even a disaster in the case of an RTOS.

2.2 Classification

An RTOS can by classified by several system characteristics. In this section we'll have a detailed look at two important characteristics: scheduling in realtime systems and realtime clock.

2.2.1 Scheduler and Algorithms

Before we discuss different realtime scheduling algorithms, we will give a brief overview of how those can be classified as seen in figure 1.
In general, we have to distinguish between *static* and *dynamic* scheduling. A static scheduler generates its dispatch table at compile-time. To achieve an operating and feasible dispatch table, it has to analyze all tasks for their characteristics, for example the worst case execution time (WCET). At run-time, this scheduler will be deterministic and its dispatcher will require virtually no overhead. A dynamic scheduler, on the other hand, decides which tasks to dispatch at run-time. While this is especially useful for tasks with irregular computation workload, the scheduler is non-deterministic and could require a lot more overhead than a static one.
Furthermore, we can distinguish between *preemptive* and *non-preemptive* scheduling. A preemptive scheduler is allowed to interrupt the current running task and dispatch another, while a non-preemptive scheduler has to wait until the running task finishes its work or is blocked.[Ko11]

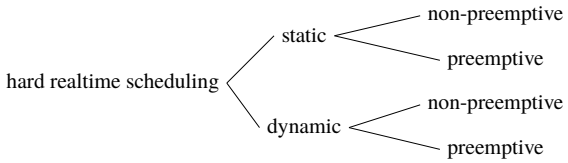


Fig. 1: Classification Hierarchy (Inspired by [Ko11, Fig. 10.1])

Schedule Feasibility In GPOS scheduling, it's all about fairness and throughput. However, since we're in a hard realtime environment, our focus is on meeting every tasks' deadline no matter what happens. Consider two tasks, which have the following attributes: The *computation time* is the same as their deadline and they both run at the same *periodic interval*. Now, we can see that there is no feasible way to schedule both so they can meet their required deadlines. We can only see this because we made the assumption that their computation time is equal to their deadline. But how can we even measure the computation time? One way would be *integration* testing. The system will be run with various different test data and will be checked for missed deadlines. This may work for smaller systems, but will fail once multiple programmers work on isolated components. Another way is

static analysis, often called offline-configuration. It will analyze each task and calculates the WCET based on the generated assembly code. By referring to the CPU manual, it's possible to get the duration of every instruction. Nonetheless, it gets more complicated when you include branches or for-loops (JUMP Instructions) in your code. In the end, it has to give a clear answer to the question: Is it possible to schedule all tasks without missing deadlines?

Priority Scheduling One category of RTOS suitable schedule algorithms is called *priority driven*. Every algorithm in this category assigns a priority to each task. With this information, every time a new task requests computing time, the scheduler will dynamically compare its priority with the current running task. If the incoming tasks priority is higher, the current task will be preempted and is replaced by the new one. There are also two different types of priority driven algorithms: *static* and *dynamic* ones (They should not be mistaken with the ones defined in 2.2.1). An algorithm is static, or *fixed*, when the task priorities are only set once and will not change within run-time. In the contrary, the algorithm is called dynamic when task priorities can be changed.
C. L. Liu and J. W. Layland [LL73] introduced a method called *fixed priority scheduling*. Priorities are assigned statically by an algorithm called *rate monotonic* (RM), where every task gets a unique priority based on its period. A shorter period task will get a higher priority than a task with a longer period. If two tasks have the same period, it will be decided randomly which one gets the higher priority. With it, they also defined necessary environment boundaries:

- Hard deadline tasks requests are periodic, with a constant interval
- Each task needs to finish before the next request for the same one arrives (deadline is equal to the interval)
- Tasks don't depend on each other
- Each task's run-time is constant and does not change
- Non-periodic tasks are special and don't have hard deadlines

At first, those boundaries seem very limiting. It doesn't allow spontaneous events and constant run-time is often hard to achieve, but in the end we can make the following statement: Let n be the amount of tasks, C_i the worst execution time of task i and T_i the period of task i:

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) \leq n(2^{\frac{1}{n}} - 1) \tag{1}$$

If this equation holds, then this task set is schedulable, i.e every task will be able to finish before its deadline. If not, then the result can't be implied. Meanwhile, studies improved the static analysis by taking more arguments into account. For example, M. Joseph and P.

Pandya [JP86] introduced RT_i, the *worst-case response time* of task i.

$$RT_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{RT_i}{T_j} \right\rceil C_j \tag{2}$$

where $hp(i)$ returns a set of tasks which have a higher priority than task i. The equation seems hard to solve since RT_i is on both sides. However, after this technical report [TH95] it can be solved by using a recurrence relation. If every tasks RT_i *is lower or equal the tasks* deadline, the scheduling will succeed. This analysis is also called the *exact analysis*, since there are cases of task sets where it will confirm feasibility while the analysis of C. L. Liu and J. W. Layland is not able to provide a distinct result.
An example for a dynamic preemptive priority based algorithm is *earliest deadline first* (EDF). As soon as a task finishes or a new task is released, it will utilize a process queue and search for the process closest to its deadline, which then will be scheduled for execution. Being in the same environment, there also exists a feasibility condition as follows:

$$\sum_{i=1}^{n} \left(\frac{C_i}{T_i} \right) \leq 1 \tag{3}$$

In both formulas 1 and 3, the right side represents overall CPU utilization. So when comparing EDF with RM, we're looking at 100% and 69% (for $n \to \infty$). Based on this, Dertouzos [De74] proofed, that as soon as a task set is schedulable by any algorithm, it is schedulable by EDF.

Aperiodic Tasks In the previous paragraph we had a look at some different algorithms and ways to schedule tasks. However, to give a distinct answer on feasibility, all of them required the tasks to be periodic with a deadline equal to the interval. But in a realtime world we often have to deal with indeterministic events, or in a different context, *aperiodic* tasks. When we consider an aperiodic task with no boundaries relating inter-arrival time, it's simply not possible to make any assumption about the feasibility of the corresponding task set. In this case, it would be necessary to define a *minimum* inter-arrival time. With this boundary, we can now again assume a worst case scenario and calculate scheduling feasibility based on that.

2.2.2 Realtime-Clock

The use of an RTOS suggests a reference to something in the real world. Though we need to keep the system in sync with the time of the real world - *realtime*. To achieve this constraint, a *realtime clock* (RTC) is essential. In the following we will explain what an RTC is and why it is indispensable for an RTOS.
The *realtime clock* is not a software part of the OS. Thus, we need a piece of hardware to

get a real world referencing time. The **clock hardware** is build out of an crystal oscillator, a counter and a holding register. 'A piece of quartz crystal [...] can [...] generate a periodic signal of very great accuracy' [TB14, p.388]. Depending on the chosen crystal the signal is 'in range of several hundred megahertz (MHz) to a few gigahertz (GHz)' [TB14, p.388]. On each signal the counter gets decremented down to zero, that results with an interrupt. Depending on the operational mode, the clock stops after an interrupt until it is started again (*one-shot-mode*) or periodic interrupts occur (*square-wave-mode*). These interrupts are called *clock ticks*. The holding register is used to copy its value into the counter after each interrupt. By example for a 500-MHz crystal the representing value of the holding register could be between $2\,nanoseconds\,(nsec)$ and $8.6\,nsec$ with a (unsigned) 32-Bit register. Thus, it is rather not only a simple clock, but more a programmable clock. The **clock software** is a part of the OS, e.g. as a system driver. It provides services to get or adjust the value of *nsec* for each clock tick, maintain the time of the day and preventing processes from monopolizing the CPU.

As we know the timing constraint in an RTOS is important, it should be plainly why the RTC is indispensable. The core software parts (e.g. the scheduler) do not know anything about the passing time such as *nsecs*, rather more they understand the clock tick and know what to do based on it. Indeed the clock tick is no time unit but it is referencing to one by which the indirect dependency from the RTOS to the realtime is noticeable. This and additionally the accuracy of the generated time proves the importance and indispensability of an RTC for an RTOS.

3 RTOS Fields of use and Requirements

In this section we will take a look at two application areas where an RTOS is absolutely essential. In both areas the software requirements are regulated by industry standards which have to be followed. In the next sub sections we will take a view at one concrete implementation for each chosen industry area, because both of the selected use cases are highly safety/time critical applications in two different but extreme environments.

3.1 Avionic

Avionic systems are controlled and supervised by aviation safety organization like EURO-CONTROL or International Civil Aviation Organization (ICAO). Both are focosing on keep a high safety level. This sector is huge and is divided into a lot of subcategories. Thus, we simply choose ACAS [Wi04] which is one of a very critical and high important system.

The Airborne Collision Avoidance System (ACAS) is installed, in respect to the every day increasing air traffic, to detect critical approaches to the own aircraft and prevent air

collisions. Basically safety distances are required for an aircraft in each direction to detect a critical situation. Additionally to the air traffic controller on the ground and the pilot in the aircraft, the ACAS looks several times per second for other aircrafts in the surroundings with sensor measurements. Depending on the own and the others altitude, speed, direction and descending or climbing rate, the system calculates if potentially threats exist on the basis of time of closest approach. In case of a threat, the system looks up for a conflict resolution in a optimized table to give a resolution advisory (RA) to the pilot to clear the conflict and prevent an air accident.
Because of the high speed of an aircraft, all these actions need to be done in seconds. Well known aircrafts approximately fly with 900 kilometer per hour which means 250 meter per second. Any calculation in the system that is not meeting the deadline could result in a delayed RA and maybe end up in a disaster.[KHC12] [KD07]

3.2 Medical

Systems in the health sector are highly depending on the medical and the (electro-) technical state of research. But as in the avionic sector many of those are life critical applications, which have to be running in extreme environments. As for avionic system, the challenge is to run even at -60 degrees and at a high air pressure. Some medical systems need to be intact within the human body. Many of the requirements for medical devices are regulated, amongst others, in the standard IEC 62304 for medi-technical software. One of those medical system that complies these requirements is a pacemaker system.

The Pacemaker is not only a device in the humans chest to keep the heart working. As we can read in [GO09] *Formal specification of a cardiac pacing system* the pacemaker supports different modes (permanent, bradycardia therapy, temporary) and in the [pac16] *Pacemaker Specification from Medtronic* we can see that it also can store the monitored data. Obviously a pacemaker is life-sustaining, but why does it have to be a *realtime system*? This should be clear if we have a look at the details of this system.
The different modes indicates how the pacemaker has to work, e.g. should the heart be stimulated frequently (permanent mode) or adaptive (bradycardia therapy). In the *bradycardia mode* the device has to measure the heart frequency within milliseconds, create an electrocardiogram (ECG) and analyze it by search spikes. Spikes in the cardiogram could be a trigger for a stimulation to the heart over the connected electrodes. Two of the pacing parameters are the pulse width and voltage. Both demonstrate how high the accuracy have to be in that case. These two parameter define the pacing strength and the timespan of the heart stimulation. The voltage can be configured (depends on the patient) to be between $0.13\,and\,5.00\,volt$. Furthermore, the pulse width parameter accepts values between $0.09\,and\,1.00\,msec$.
If these values aren't precise enough and the system fails to meet the time deadline, e.g.

send the pulse with an pulse width of $0.15\,msec$, the heart could run to fast and result in an uncomfortable feeling for the patient.

4 Discussion

In this section we take a look at, based on the classification 2.2.1, how some RTOS distributions solves the scheduling problem to achieve the *hard realtime* goal. Therefor, we will give a short introduction into the concrete scheduler implementation and which features the single one provides.

4.1 RTAI

The Realtime Application Interface (RTAI) is less a type of an RTOS, it is more a patch for GPOS to provide realtime functionality. With the installation of RTAI, the system underneath gets an additional scheduler installed with whom hard realtime constraints can be achieved. So if the developer does not use RTAI function to extend the base thread structure, the created thread will never meet realtime deadlines because it is scheduled based on the original scheduling algorithm of the patched GPOS. In case of the created thread is extended to the realtime thread structure, it get scheduled based on the set thread priority (as normal) and the scheduling policy which is set to FIFO, by default. But can also be set to Round-Robin (RR) to keep all queued equal prioritized threads in balance.
The challenge and a potential trap is, that the developer has exactly know what to do, since it is possible to call 'normal' linux services, which are not following realtime restrictions, out of the realtime thread context.

4.2 FreeRTOS

FreeRTOS is a small real time operating system which can be run on multiple different architectures. It's mainly used on single core CPUs and offers a minimal feature set for e.g scheduling, inter-process communication and memory allocation.
Regarding our scheduling classification, it's using a dynamic priority driven algorithm *highest priority first*. Since priorities can be set statically and dynamically, it also needs to support priority-equal tasks which are in a ready state. In this case, it will process those tasks with the RR strategy. This is unusual for realtime systems, since RR focuses on fairness and will require a lot of overhead-heavy context switches. Generally speaking, FreeRTOS gives tasks a lot of control over the scheduler, which is not seen in a lot of other OS. For example, every task has the ability to call $vTaskSuspendAll$. With this call, scheduler context switches will be disabled. The current task will be the executing one until he either finishes his work or calls $xTaskResumeAll$.

4.3 QNX

QNX Neutrino is one of the most well known RTOS used in several industry sector, e.g. medical. Basically the scheduler provides a priority-driven preemptive functionality, which means that a thread with a higher priority becomes *READY* the current running get preempted immediately. Because it exist one thread queue for each priority all thread with the same priority needs to be schedules as well. Therefor it is possible to set the *scheduling policy*. But a special feature in QNX is, that the policy can be set per-thread instead of system-wide. The developer can select out of three policies: *FIFO, RR, Sporadic*. A **FIFO-Thread** runs until it is blocked, finished or preempted as illustrated in figure 2. In case of preemption, the thread will be put at the first place of the queue.

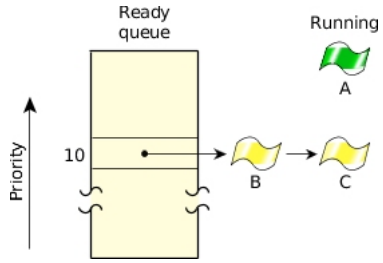


Fig. 2: FIFO: A is running until blocked or finished. (Source: [QN] Documentation)

As the figure 3 illustrate, a **RR-Thread** runs until its timesslice is consumed and get put at the end of the queue. The **Sporadic-Thread** is scheduled by RM-Scheduling-Algorithm

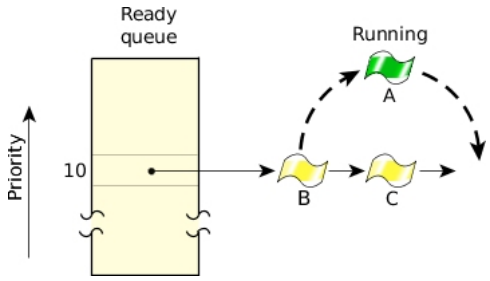


Fig. 3: Round Robin: Thread A ran until it consumed its timeslice; the next READY thread (Thread B) now runs. (Source: [QN] Documentation)

(see 2.2.1). Thus each thread get set a unique priority depeding on is period.

4.4 Are these RTOS hit the medical or avionic requirements?

Now we want to check how we can classify the three previously mentioned RTOS into the likewise mentioned fields of use based on the knowledge provided by this paper.

Through the hard conditions required by safety and life critical applications, we can disqualify RTAI. Because of potential traps the developer could go into during the development with the RTAI, this kind of RTOS (if we want to call it that) is not suitable enough. Besides it is not certifiable without a huge effort to achieve the special industry requirements. Therefor RTAI is not officially certified. But this is absolutely necessary for medical and avionical applications.
For systems like an ACAS 3.1 or a pacemaker 3.2 the IEC published the standard IEC 61508 [IE00], which defines and describes requirements of functional safety for electronic systems in critical use cases. For those cases *SafeRTOS* exists, which is a part of FreeRTOS. How FreeRTOS, or in that special case SafeRTOS, QNX is certified by TÜV SÜD [TUa] as well, that it is in accordance with the necessary for IEC 61508 [IE00] at a safety integrity level (SIL)[Sc11] of 3 - the highest level that can be achieved. Thus we can naively assert that both, FreeRTOS/SafeRTOS and QNX are suitable for applications in the avionical sector.
As for the avionical application, there also exists regulatory standards for medical use cases. The QNX vendor offers a specialized and IEC 62304 [IE06] compliant distribution for medical applications. However, FreeRTOS/SafeRTOS also have been certified against EN 62304 (german version of IEC 62304 [IE06]) by TÜV SÜD [TUb]. We can also say that both RTOS are hitting the requirements for the considered fields of use. These assessments are also supported by the official statements on the single ones webpage.
Summarized, excluding RTAI, FreeRTOS/SafeRTOS and QNX are, based on their certifications, suitable for application development in the medical or avionical field of use. Even if third party vendors exist that offer more specialize RTOS, though their base RTOS is one of the listed. Finally, to get a high detailed classification, it is absolutely necessary to do further research.

References

[De74] Dertouzos, Michael L.: Control Robotics: The Procedural Control of Physical Processes. In: IFIP Congress. pp. 807–813, 1974.

[GO09] Gomes, Artur Oliveira; Oliveira, Marcel Vinicius Medeiros: Formal specification of a cardiac pacing system. In: International Symposium on Formal Methods. Springer, pp. 692–707, 2009.

[IE00] IEC, IEC: 61508: 2000 Functional safety of electrical/electronic/programmable electronic safety related systems. International Electrotechnical Commission, Geneva, 2000.

[IE06] IEC, IEC: 62304: 2006 Medical device software–software life cycle processes. International Electrotechnical Commission, Geneva, 2006.

[JP86] Joseph, Mathai; Pandya, Paritosh: Finding response times in a real-time system. The Computer Journal, 29(5):390–395, 1986.

[KD07] Kuchar, JE; Drumm, Ann C: The traffic alert and collision avoidance system. Lincoln Laboratory Journal, 16(2):277, 2007.

[KHC12] Kochenderfer, Mykel J; Holland, Jessica E; Chryssanthacopoulos, James P: Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[Ko11] Kopetz, Hermann: Real-time systems: design principles for distributed embedded applications. Springer Science & Business Media, 2011.

[LL73] Liu, Chung Laung; Layland, James W: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1):46–61, 1973.

[pac16] Micra Transcatheter Pacing System Specification, 2016.

[QN] QNX - Real Time Operating System. http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.prog%2Ftopic%2Foverview_PRIOR.html, Stand: Juni 2017.

[Sc11] Schröder, Hans Christian; Günther, André; Klingler, Karsten; Leidel, Thomas: Safety Integrity Level Der richtige Weg zur funktionalen Sicherheit? VGB powertech, p. 87, 2011.

[TB14] Tanenbaum, Andrew S.; Bos, Herbert: Modern Operating Systems. Prentice Hall Press, Upper Saddle River, NJ, USA, 4th edition, 2014.

[TH95] Tindell, Ken; Hansson, Hans: Real time systems and fixed priority scheduling. Technical Report Tech. rept., 1995.

[TUa] TUEV, TÜV SÜD: , Funktionale Sicherheit nach EN 61508. http://www.tuev-sued-stiftung.de/uploads/images/1339742090447928560703/funktionalesicherheit-61508.pdf.

[TUb] TUEV, TÜV SÜD: , Medizingerätesoftware nach IEC 62304. http://www.tuev-sued.de/produktpruefung/branchenloesungen/medizinprodukte/pruefung-bewertung/pruefung-aktive-medizinprodukte/iec-62304/software-fuer-medizinprodukte, Stand: Juni 2017.

[Wi04] Williams, Ed: Airborne collision avoidance system. In: SCS. volume 4, pp. 97–110, 2004.

Verbreitete Programmiersprachen für eingebettete Systeme und deren Besonderheiten

Philipp Posovszky[1]

Abstract: Der Überblick über die Sprachen zum Entwickeln von Programmen für eingebettete Systeme ist schwierig und eine Einordnung der Sprache zur Eignung in eine bestimmte Problemstellung ebenfalls. Zum einen gibt es viele verschiedene Programmiersprachen (C, C++, Python, Java, Ada . . .) die eingesetzt werden können und zum anderen verschiedene Anforderungen an das eingebettete System selbst. Diese Arbeit beschäftigt sich mit einem allgemeinen Überblick und die Besonderheiten der verbreitesten Programmiersprachen. Der Fokus liegt dabei auf C, C++ und Ada. Eingegangen wird hierbei nicht nur auf die Eigenschaften der Programmiersprachen, sondern auch auf mögliche Probleme im Umgang mit diesen Sprachen im eingebetteten Umfeld, mit Hauptaugenmerk auf Echzeitsysteme. Zudem wird kurz eine Alternative zu universellen Programmiersprachen beschrieben: programmieren mit ***D**omain **S**pecific **L**anguages* (DSL). Die Ausarbeitung schließt mit einer kurzen Zusammenfassung und dem aktuelle Entwicklungsstand der Sprachen ab.

Keywords: Programmiersprachen; Assembler; C; C++; Ada; DSL; eingebettete Systeme; Echzeitsysteme

1 Einleitung

Während der letzten Jahrzehnte haben eingebettete Systeme einen festen Platz in unserer Gesellschaft eingenommen. Ein Leben ohne diese Technologie wäre heutzutage undenkbar. Die Einsatzgebiete sind dabei vielfältig: Von der Mikrowelle, Zeitschaltuhr einer Straßenlaterne, Automotoren, medizinischen Geräten wie Herzschrittmachern bis hin zur Kernspaltung in Atomkraftwerken. Dabei müssen hohe Performanz Anforderungen und Ausfallsicherheit gewährleistet sein. Das breite Anwendungsgebiet führt zu immer weiter steigenden Anforderungen an Mikrocontroller, deshalb werden eingebettete Systeme in der Entwicklung der Hardware, als auch bei der Software, immer komplexer. Zudem können diese Systeme of nur sehr klein dimensioniert werden, was Einschränkungen bei den Ressourcen (z.B. Arbeitspeicher, Rechenleistung, Kühlung) zur Folge hat. Auf diese begrenzten Ressourcen muss ein besonderes Augenmerk bei der Entwicklung von Programmen für eingebettete Systeme gelegt werden. Eingebettete Systeme sind oft als Echzeitsysteme (eng. *Realtime-Systems*, kurz RT-Systems) ausgelegt. Eine andere Form sind Universalsysteme (eng. *General-Purpose-Systems*, kurz GP-Systems). Ein RT-System

[1] Ludwig-Maximilians-Universität München, Kommunikationssysteme und Systemprogrammierung, Professor-Huber-Platz 2, 80539 München, Deutschland, Ph.Posovszky@gmail.com

erfüllt definierte maximale Ausführungszeiten für einzelne Aufgaben, im Gegensatz sind GP-Systeme dazu ausgelegt, alle Aufgaben möglichst schnell parallel auszuführen. Häufig werden GP-Systems für ***I**nternet **o**f **T**hings* (IoT) eingesetzt, wenn diese keine Anforderungen an Echtzeit besitzen. Der *Raspberry Pi* oder Smartphones sind Beispiele für GP-Systeme. In der folgenden Arbeit wird auf die verbreitetsten Sprachen im Bereich der eingebetteten Anwendungsentwicklung für RT-Systems eingegangen, sowie einige häufig auftretende Probleme bei der Anwendung im eingebetteten Umfeld erläutert.

2 Hintergrund

Die Anwendungsgebiete von eingebetteten Systemen sind zu vielfältig um für jedes Gebiet einen eigenen Mikrocontroller zu entwickeln, weshalb moderne Mikrocontroller programmierbare Funktionsblöcke besitzen. Solche Mikrocontroller werden mit *Assembly* programmiert. Der *Assembly* Befehlssatz ist stark von der Designphilosophie des Mikrocontrollers abhängig. Beim ***C**omplex **I**nstruction **S**et **C**omputer* (CISC) werden viele mächtige Einzelbefehle eingesetzt. Bei ***R**educed **I**nstruction **S**et **C**omputer* (RISC) werden hingegen nur wenige elementare Befehle umgesetzt. Eine Portierung vom *Assembly*-Code zwischen verschiedenen Architekturen ist aufgrund der unterschiedlichen Befehlssätze nicht trivial möglich. [FL98]

Bei der Übersetzung von Assembly wird dieser direkt in die entsprechenden Maschinenbefehle (Operatoren) der Architektur umgesetzt, siehe Listing 1.

Listing 1: ARM-Assembly mit Maschinenbefehlen als HEX-Wert [Ma16]

```
MOV R3, R9          ; Verschiebt Wert von R9 in R3
E1 A0 30 09         ; Code in HEX-Werten
```

Diese Art der Programmierung wurde von Programmiersprachen der 3. Generation oder höher abgelöst. Dazu zählen die Programmiersprachen C, C++ oder Ada. Diese Programmiersprachen ermöglichen unter Zuhilfenahme eines *Compiliers*, komplexe Programmanweisungen in *Assembly* zu übersetzen (siehe Listing 2 und Listing 3). Durch diesen Abstraktionsschritt wird erst zum Zeitpunk des Kompilierens entschieden, mit welcher Zielarchitektur das Programm kompatibel ist.

Listing 2: C-Code [Ma16]

```
int total;
int i;
total = 0;
for (i = 10; i > 0; i--) {
        total += i;
}
```

Listing 3: Erzeugter ARM-Assembly Code[Ma16]

```
        MOV   R0, #0              ; R0 accumulates total
        MOV   R1, #10             ; R1 counts down to 1
again   ADD   R0, R0, R1          ; Add i to total
        SUBS  R1, R1, #1          ; Reduce i by 1
        BNE   again               ; Jump to again if is != 0
halt     B     halt               ; infinite loop to stop
```

Für eingebettete RT-Systems muss zu jederzeit vorhersagbar sein, wie sich das Programm verhält. Dazu ist das Einhalten von *harten* und *festen Deadlines* (de. Fristen) wichtig, denn die Korrektheit eines Ergebnisses hängt nicht nur von der logischen Korrektheit, sondern auch von der Zeit ab, in der das Ergebnis erzeugt wird. Sobald eine *harte* Deadline verletzt wurde, ist ein deterministischer Ablauf nicht mehr gewährleistet. *Weiche Deadlines* dagegen haben keinen kritischen Einfluss auf den Programmablauf. Die unterschiedlichen *Deadlines* werden wie folgt definiert [SR94]:

Harte *Deadlines* - Eine Deadline ist *hart*, wenn die Konsequenz diese nicht einzuhalten katastrophale Folgen hat. Periodische Aufgaben haben normalerweise *Deadlines* mit harter Echtzeit.

Feste *Deadlines* - Eine Deadline ist *fest*, wenn das produzierte Ergebnis seine Nützlichkeit verliert sobald die Deadline überschritten wird. Das Nichteinhalten der Deadline ist nicht sehr kritisch.

Weiche *Deadlines* - Eine Frist, die weder *hart* noch *fest* ist, ist *weich*. Der Nutzen des Ergebnisses, die durch eine Aufgabe mit *weichen Deadlines* erzeugt wird, nimmt im Laufe der Zeit nach Ablauf der Deadline ab.

Aufgrund des geforderten Determinismus sind Sprachen wie Java, welche in einer komplexen *Runtime* läuft und einen nicht deterministischen *Garbage Collector* besitzt, oder Python, welche erst zur Laufzeit interpretiert wird, ungeeignet für eingebettete Systeme mit Echzeitanforderungen. In einem als GP-System ausgelegten eingebetteten System können diese jedoch eingesetzt werden. Darüber hinaus müssen einige Regeln bei dem Einsatz von Programmiersprachen beachtet werden, um die Anforderung an den Determinismus nicht zu verletzen, dazu bei den einzelnen Programmiersprachen mehr.

Weitere wichtige Designqualitäten für sicherheitskritische Programme nach ***J**oint **S**trike **F**ighter (JSF) Coding Standard* C++ [St05] sind:

Zuverlässigkeit - Der ausgeführte Code muss sich jederzeit konsistent zu seinen Anforderungen in einer vorhersagbaren Weise verhalten.

Wartbarkeit - Der Quellcode sollte konsistent, lesbar, einfach im Design, und einfach zu debuggen sein.

Portierbarkeit - der Quellcode sollte portierbar sein (z.B. nicht vom *Compilier* oder *Linker* abhängen).

Erweiterbarkeit - Anforderungen tendieren dazu, sich weiter zu entwickeln. Darauf sollte beim Design geachtet werden.

Testbarkeit - Der Quellcode sollte einfach zu testen sein.

Lesbarkeit - Der Quellcode sollte leserlich und leicht verständlich verfasst werden.

Eine Programmiersprache im eingebetteten Umfeld, sowie auch bei allen anderen Softwareprojekten, soll dabei unterstützen, diese Designqualitäten einzuhalten.

2.1 Memory Management (de. Speicherverwaltung)

Im Kontext der *Speicherverwaltung* geht es um eine effektive und effiziente Nutzung des begrenzten Arbeitsspeichers eines Systems. Insbesondere ist dies bei eingebetteten Systemen relevant, da diese in der Regel nur über wenig Speicher verfügen. Man unterscheidet generell zwischen statischer und dynamischer *Speicherverwaltung*.

Bei einer rein statischen *Speicherverwaltung* ist zum Zeitpunkt des Kompilierens bekannt wie viel Speicher benötigt wird und die Allokation findet direkt im statischen Bereich statt. Ein Freigeben dieses Speichers erfolgt erst bei Beendigung des Programmes. Bei einer dynamischen *Speicherverwaltung* geschieht die Allokation vom Speicher während der Laufzeit. Dabei muss zwischen dem Speicherbereich *Stack* (de. Stapel) und dem *Heap* (de. Haufen) unterschieden werden, definiert nach [Bl03]:

Stack (de. Stappel) - Bei dem Stack Speicher handelt es sich um einen dynamischen Speicherbereich mit einer festen Größe. Eine explizite Freigabe von Speicher ist nicht möglich, dieser wird nach dem ***F**irst **i**n Last **O**ut* (FILO) Prinzip verwaltet und automatisch beim Verlassen eines Geltungsbereich freigegeben. Das Ablegen und Entfernen von Elementen auf dem Stack ist aufgrund des FILO Prinzip sehr effizient. Der Stack wird hauptsächlich für Variablen mit einer kurzen Lebensdauer verwendet.

Heap (de. Haufen) - Bei dem *Heap* Speicher handelt es sich um einen dynamischen freien Speicherbereich: Dieser ist nur durch die Speichergrenze auf Prozessebene begrenzt. Alle auf dem *Heap* angelegten Speicherbereiche müssen explizit wieder freigegeben werden. Durch die Speicherfragmentierung beim freigeben von Objekten ist der Verwaltungsaufwand von *Heap*-Speicher höher als beim *Stack*.

Bei Verwendung des dynamischen *Heap*-Speichers gibt es keine Garantie für Laufzeit, sowie erfolgreicher Allokation von Speicher. Wenn immer es möglich ist, sollte deshalb

im eingebetteten Umfeld nach der Initialisierungsphase des Programms auf dynamische Instanziierung verzichtet werden. Ein *Garbage Collector* verhält sich ebenfalls nicht deterministisch und darf in eingebetteten Systemen mit Echtzeitanforderungen nicht eingesetzt werden. [St05]

2.2 Parallele Programmierung

Eine Programmiersprache besitzt die Eigenschaft Parallelität (engl. concurrency), wenn es möglich ist, mehrere Operationen/Befehle/Anweisungen gleichzeitig auszuführen. Dabei kann es sich um unabhängige oder abhängige Anweisungen handeln. Dies gilt für Systeme mit einer einzelnen CPU. Bei einer einzelnen CPU wird häufig auch von Pseudoarallelität gesprochen, hierbei wird durch ein schnelles abwechselndes Abarbeiten von mehrerer Aufgaben der Eindruck von echter Parallelität vermittelt. Bei modernen Multi-CPU-Systemen dagegen, kann echte Parallelität durch das Aufteilen des Programmcodes in mehrere Threads erzielt werden. Jedoch kann es durch das Ausführen von mehreren Threads zu *Raceonditions* auf Variablen im gemeinsamen Speicher kommen. Dies muss durch geeignete Techniken verhindert werden, z.B. exklusiver Zugriff auf Variablen durch Semaphore. [La79]

3 Programmiersprache C

Die Programmiersprache C wurde 1973 von Dennis Ritchie bei *Bell Laboratories* entwickelt. Bei C handelt es sich hierbei um eine Weiterentwicklung der Elternsprache B und der Großelternsprache BCPL. C ist heute eine der am weitesten verbreiteten Programmiersprachen bei allgemeinen und eingebetteten Anwendungen. Die Sprache C liegt in diversen Sprachvarianten vor (z.B. ANSI C, C89, C99 und C11. . .). C trennt den Programmcode in einen Spezifizierungsteil, die **.h*-Dateien, und in einen Implementierungsteil, die **.c*-Dateien. Eine Empfehlung aus der Industrie ist, sich bei der Entwicklung von Programmen in C, für kritische Anwendung im eingebetteten Umfeld an MISRA-C [Lt04] zu halten. Im Folgenden werden auf häufige Probleme im Umgang mit C in eingebetteten Systemen, sowie auf mögliche Lösungen eingegangen.

3.1 Speicherverwaltung

In C ist die Speicherverwaltung variabel gestaltbar. Es ist eine statische, als auch eine dynamische Speicherverwaltung möglich. Normalerweise werden Initialisierungswerte von globalen Variablen aus dem statischen Speicher geladen. Alle Variablen innerhalb eines Funktionsaufrufs werden auf dem Stack angelegt, und nach dem Verlassen des Geltungsbereiches wieder automatisch freigegeben. Dies gilt nicht für Allokationen auf

dem freien *Heap*-Speicherbereich: Dabei wird nur der Zeiger auf dem Speicherbereich auf dem Stack abgelegt. Die MISRA-C Regel 16.2 gibt vor, bei der Nutzung des *Stacks* keine rekursiven Aufrufe durchzuführen. Dies könnte zu einem *Stack Overflow*, bei welchem sich das Programm nicht mehr deterministisch verhält, oder abstürzt. Siehe Listing 4 für Beispiele verschiedener Formen der Speicherallokation. [Lt04]

Listing 4: Verschiedene Varianten der Speicherallokation in C

```
//Literal im statischen Speicher, Zeiger auf Stack
char* answer=''The answer to life the \
universe and everything: 42''
void foo(){
        int i=5; //Allokation auf dem Stack
        //Allokation auf dem Heap, Zeiger im Stack
        int *array = (int *) malloc(5 * sizeof(int));
        free(array) //Freigeben vom Heap
}
```

Wie schon erwähnt ist von der Nutzung des *Heap*-Speicherbereichs während der Laufzeit abzusehen. In C gibt es noch andere Gründe, welche das Arbeiten mit dem *Heap*-Speicherbereichs komplizierter gestalten. Der wichtigste Grund ist, dass *malloc(...)* keine deterministische Laufzeit besitzt und eine erfolgreiche Allokation nicht garantiert werden kann. Des Weiteren muss die Deallokation des nicht mehr verwendeten Speichers sichergestellt werden, siehe Listing 4 Aufruf der Funktion *free(ptr)*. Falls das nicht geschieht, entstehen Speicherlecks, wodurch das Programm abstürzt, sobald der komplette Speicher vergeben ist. Problematisch ist ebenfalls eine mehrfache Deallokation durch *free(ptr)*. Dies führt bei den meisten Implementierungen zu nicht definierten Verhalten. Der Standard MISRA-C verbietet in Regel 20.4 die Verwendung von dynamischer *Heap* Allokation während der Laufzeit, sowie alle Standardbibliotheken von C, die diese Allokation verwenden. Siehe Listing 5 für ein Beispiel zum Zusammenfügen von Zeichenketten in C, ohne Verwendung von Standardbibliotheken *string.h*.

Listing 5: Zusammenfügen von Text ohne Standard Bibliothek

```
char s1[10]="Hallo ",s2[4]="du\n",i, j;
for(i = 0; s1[i] != '\0'; ++i);   // Ermittlung Laenge s1
for(j = 0; s2[j] != '\0'; ++j, ++i){
        s1[i] = s2[j]; // Zusammenfuegen
}
```

3.2 Parallele Programmierung

Bei der Entwicklung von C wurde der parallelen Ausführung von Programmcodes keine Beachtung geschenkt. Zum damaligen Zeitpunkt waren Multiprozessorsysteme noch nicht

verbreitet. Für die Parallelität werden C-Spracherweiterungen in Form von Bibliotheken benötigt (z.B. pThreads [IE13],. . .). Diese ermöglichen das Ausführen von mehreren Threads und deren Synchronisation. Seit der C-Variante *C11 ISO/IEC 9899:2011* [IS11] werden Threads native unterstützt, sowie ein Speichermodell über die Bibliothek *stdatomic*. Diese Bibliothek verhindert, dass mehrere Threads gleichzeitig auf die selbe Variablen zugreifen können.

3.3 Besonderheiten

Eine Besonderheit von C ist die Möglichkeit, innerhalb des C-Codes, *Assembly* zu verwenden, siehe Listing 6. Dadurch ist es möglich einen hardwarenahen Code innerhalb von C zu schreiben. Dies wird unter anderem häufig in der Entwicklung von Treibern verwendet.

Listing 6: Inline *Assembly* für ARM in C

```
asm("mov r0,r0"); /* NOP Beispiel */
```

4 Programmiersprache C++

C++ wurde von Bajrne Stroustrup bei *AT&T* als Erweiterung der schon vorgestellten Sprache C entwickelt. C++ ist eine universelle Programmiersprache, die das Design und ein reiches Typsystem, sowie leichtgewichtige Abstraktionen in den Vordergrund stellt. C++ wird für eine große Vielfalt von Projekten eingesetzt (von Mikrocontroller bis hin zu großen verteilen Anwendungen). C++ trennt den Programmcode in einen Spezifizierungsteil, die **.h*-Dateien, und in einen Implementierungsteil, die **.cpp*-Dateien. Eine Empfehlung aus der Industrie ist, sich beim Entwickeln von C++ Programmen für kritische Anwendung im eingebetteten Umfeld, an den *Standard JSF C++* [St05] zu halten. Im Folgenden wird auf häufige Probleme im Umgang mit C++ in eingebetteten Systemen und deren möglichen Lösungen eingegangen, angelehnt an den *Standard JSF C++* [St05]. C ist eine echte Teilmenge der Sprache C++ und somit vollständig kompatibel. [St13]

4.1 Speicherverwaltung

Da C++ auf C aufbaut, bietet auch C++ die Möglichkeit der statischen und dynamischen Speicherverwaltung. Normalerweise werden Initialisierungswerte von globalen Variablen aus dem statischen Speicher geladen. Alle Variablen innerhalb eines Funktionsaufrufs werden auf dem Stack angelegt und nach dem Verlassen des Geltungsbereiches wieder automatisch freigegeben. Dies gilt nicht für Allokationen auf dem freien *Heap*-Speicherbereich: Dabei wird nur der Zeiger auf dem *Stack* abgelegt. Die Speicherressourcen für Objekte werden in C++ über ***R**esource **A**cquisition **I**s **I**nitialization* (RAII) verwaltet. Damit ist die

Durchführung der Speicherallokation innerhalb des Konstruktors und die Deallokation im Dekonstruktor gemeint. Am Ende des Geltungsbereiches eines Aufrufs wird garantiert, dass alle Dekonstruktoren der Objekte aufgerufen werden. [St13]

Wie bei *MISRA-C* ist nach *Standard JSF C++* [St05] Regel 206 eine dynamische *Heap* Allokation nur in der Initialisierungsphase erlaubt. Es ist dabei zu beachten, dass Bibliotheken aus der ***S**tandard **T**emplate **L**ibrary* (STL) im Hintergrund dynamische *Heap*-Allokationen tätigen können. Dies muss durch Übergabe eines *Allocators*, der sich eines bei der Initialisierung definierten *Storage Pool* bedient, verhindert werden. Des Weiteren ist nach Regel 119 der Aufruf von Rekursionen aufgrund von möglichen *Stack Overflows* untersagt. Für eine einfache Allokation eines *Storage Pools* während des Kompilierens siehe Listing 7.

Listing 7: Erstellung eines statischen *Storage Pool* in C++

```
class Item {
        public:
                char buf[40];
};
Item itemPool[100];
...
  new(&itemPool[0]) Item;
  new(&itemPool[1]) Item;
```

Zusätzlich existieren in C++ *Smart Pointer*, welche das Entstehen von Speicherlecks verhindern: Sobald ein *Smart Pointer* nicht mehr referenziert wird, wird dieser automatisch aus dem Speicher entfernt. [IS12]

4.2 Parallele Programmierung

Wie bei C unterstützt die Sprache C++ von Beginn an keine Parallelität. Die Parallelität wurde bisher in C++, wie in C, über Bibliotheken (z.B. pThreads [IE13],. . .) umgesetzt. Jedoch wird seit der Variante *ISO/IEC 14882:2011 C++11* ein Threadmodell nativ unterstützt. [IS12]

4.3 Template Programmierung

Templates ermöglichen es, generisch zu programmieren. Der C++ Template Mechanismus erlaubt die Verwendung von Typen als Parameter in der Definition von Klassen, Funktionen oder Typ-Aliase. Dabei entsteht kaum *Overhead* und die Programmierung ist vergleichbar mit einem selbst geschriebenem Code in Laufzeit und Speichereffektivität.[St13] Im Listing 8 ist ein einfaches Template zum Dreieckstausch dargestellt.

Listing 8: Template für Tauschfunktion in C++

```
template <class T>
void generic_swap(T& left, T& right)
{
        T temp =  left;
        left = right;
        right = temp;;
}
```

Durch den Einsatz von Templates wird die Übersichtlichkeit, Lesbarkeit, Wartbarkeit und Wiederverwendbarkeit vom Quellcode stark erhöht. Im Gegensatz zu C ist es hier nicht notwendig, mehrere Varianten für verschiedene Typen bereitzustellen (z.B. diese aufwendig über Makrokonstrukte zu erzeugen).

4.4 Besonderheiten

Besonderheiten von C gelten auch für C++. Des Weiteren bietet C++ die Möglichkeit zu Ausnahmebehandlungen, um auf Fehler reagieren zu können. Diese sollen nach *Standard JSF C++* [St05] Regel 208 aber nicht verwendet werden, da diese durch Kompilier noch nicht ausreichend unterstützt werden. Außerdem bietet C++ das Konzept der Polymorphie. Richtig eingesetzt kann Polymorphie zu einer besseren Abstraktion der Programmstruktur beitragen. Mehrere Regeln des *Standard JSF C++* [St05] schränken die Nutzung der Polymorphie ein.

5 Programmiersprache Ada

Ada wurde vom *US Department of Defense* im Jahr 1974 mitfinanziert. Das Ziel dabei war, eine spezielle auf eingebettete und Echtzeit-System ausgelegte Sprache zu entwickeln, welche einfach zu warten und zu erlernen ist. Die Sprache hat sich in folgenden Bereichen stark durchgesetzt: Flugsicherung, Sicherheitseinrichtungen der Eisenbahn, Waffensystemen, Raumfahrt, Medizin und in der Steuerung von Kernkraftwerken. Die Sprache ist modular ausgerichtet und erlaubt die Implementierung von unabhängigen Modulen. Ada trennt den Programmcode in einen Spezifizierungsteil, die **.ads*-Dateien, und in einen Implementierungsteil, die **.ada*-Dateien. Im Folgenden wird auf die Eigenschaften der Programmiersprache Ada und deren Besonderheiten eingegangen. [BN81]

5.1 Speicherverwaltung

Die Arbeit mit statischer und dynamischer Speicherverwaltung ist in Ada möglich. Ada erlaubt ein *Garbage Collector*. Wie bei C/C++ bereits erwähnt, ist bei Programmen im

eingebetteten Umfeld von einem *Garbage Collector* abzuraten, da dieser nicht deterministisch ist. Siehe Listing 9 für Beispiele zur statischen und dynamischen *Stack* Initialisierung. Im Gegensatz zu C/C++ ist es in Ada möglich, Variablen dynamischer Länge auf dem *Stack* abzulegen. Der *Heap*-Speicher wird in Ada nun explizit verwendet. Wie in der Empfehlung für den *Heap*-Speicher bei C++, werden in Ada native *Storage Pools* verwendet. Außerdem bietet Ada diverse Schutzvorkehrungen gegen Programmierfehler. Es wird bei *Stack Overflows*, als auch bei Zugriffen auf ein *Array* außerhalb der Grenzen, ein Fehler geworfen. Dieser Fehler kann von der Ausnamebehandlung verarbeitet werden. [JB12]

Listing 9: Verschiedene Varianten zur Speicherallokation in Ada [JB12]

```
//Days_In_Month ist statisch
type Month is (Jan, Feb, Mar, ... , Nov, Dec);
Days_In_Month: array (Month) of Integer :=
 (Jan => 31, Feb => 28, Mar => 31, Apr => 30,
 May => 31, Jun => 30, Jul => 31, Aug => 31,
 Sep => 30, Oct => 31, Nov => 30, Dec => 31);
//Factor auf Stack
function double (X: Float) return Float is
Factor: constant Float := X*2;
begin
        return X;
end double;
```

5.2 Parallel Programmierung

Ada bietet die Möglichkeit der parallelen Ausführung von logisch parallelen Threads, sogenannte Tasks, die kontrolliert zusammenarbeiten. Kontrolliert bedeutet, dass einzelne Tasks völlig parallel zueinander ablaufen. Task können mittels *Rendezvous* Punkten im Code synchronisiert werden. Ferner gibt es die Möglichkeit, einen Task auf Eingaben und Ressourcen warten zu lassen. Die Kosten für die Taskverwaltung sind hierbei etwa so hoch wie das Aufrufen einer Funktion auf einem einzelnen Prozessor. [BN81]

5.3 Besonderheiten

Ada-Kompilier sind einer strengen Validierung unterworfen, deshalb hat sich Ada vor allem in sicherheitskritischen Bereichen durchgesetzt. Zudem bietet Ada die Möglichkeit der Ausnahmebehandlung. Die Sprache ist sehr streng typisiert. Alle Typ-Überprüfungen finden während des Kompilierens statt. Einige Formen von Laufzeittypisierungen, wie z.B. Längenangaben von Arrays, sind möglich. Eine implizierte Typisierung ist in Ada nicht möglich. Wie in C++ ist es auch in Ada möglich mit generischen Typen zu arbeiten, siehe Listing 10 für Dreieckstausch in Ada. [BN81]

Listing 10: Generische Tauschfunktion in Ada [**?**]

```
generic
  type E_Type is private;
procedure Generic_Swap (Left, Right: in out E_Type);
procedure Generic_Swap (Left, Right: in out E_Type) is
  Temporary : constant E_Type;
begin
  Temporary := Left;
  Left := Right;
  Right := Temporary;
end Generic_Swap;
```

6 Domänenspezifische Programmiersprachen

Eine ***D**omain **S**pecific **L**anguages* (DSL) ist eine meist deklarative Sprache, die einen ausdrucksstarken Fokus auf ein bestimmtes Problemgebiet bietet. In den meisten Fällen übersetzt eine DSL einen Aufruf in eine *Subroutine* aus einer Bibliothek, die durch eine andere Programmiersprache bereitgestellt wird. DSL kann als eine Ansicht verstanden werden, die Implementierungsdetails einer Bibliothek versteckt. Der Einsatz von DSL bringt sowohl Risiken, als auch neue Möglichkeiten mit sich. Vorteile bei der Verwendung einer DSL sind: Domänenexperten können einfach neue Codes entwickeln, einfache Wiederverwendbarkeit, validieren und optimieren auf Problemebene und vereinfachtes Testen. Nachteile sind der große Aufwand und hohe Kosten bei der Entwicklung und Wartung einer DSL. Außerdem ist das Ermitteln des richtigen Anforderungsprofils an eine DSL schwierig. Bei zu umfangreichen Anforderungen entwickelt sich eine DSL schnell zu einer Universalsprache. Ein Beispiel für eine DSL Sprache ist die Datenbankabfragesprache ***S**tructured **Q**uery **L**anguage* (SQL). [vDKV00]

7 Zusammenfassung

Die etablierten Programmiersprachen C, C++ und Ada werden stetig weiterentwickelt und durch Bibliotheken oder Sprachvarianten ergänzt. Aktuell ist C die mit Abstand am weitesten verbreitete Sprache bei eingebetteten Systemen. Jedoch hat C++ mit dem neuen Sprachstandard C++11 große Schritt unternommen, um eine komfortablere und qualitativ bessere Programmierung zu ermöglichen. Wie der *Technical Report on C++ Performance* [DA06] verdeutlicht, steht C++ in Sachen Performanz heutzutage C in nichts nach. Die Verwendung von Speicherpools erlaubt es, Probleme wie die *Heap*-Allokation zu vermeiden. Andere Sprachen wie Ada oder allgemein DSL haben weiterhin eine Existenzberechtigung. In Ada ist es möglich, besonders sicherheitskritische Anwendungen zu schreiben, die durch einen validierten *Compilier* einfach portierbar sind. DSL erlauben Domänenspezialisten

schnell neue Programme zu schreiben, ohne Kenntnisse über die unterliegende Sprache zu besitzen. Solange man sich an die jeweiligen Empfehlungen aus den Standards hält, ist es in jeder der erwähnten Programmiersprache möglich, qualitative, sichere und stabile Codes zu schreiben.

Literatur

[Bl03] Blunden, B.: Memory Management: Algorithms and Implementation in C/C++. Windows programming/development. Wordware Pub., 2003.

[BN81] Brender, R. F.; Nassi, I. R.: What is Ada? Computer, 14(6):17–24, June 1981.

[DA06] Dave Abrahams, Howard Hinnand, Dietmar Kühl Dan Saks Bill Seymour Bjarne Stroustrup Detlef Vollmann: Technical Report on C++ Performance. Bericht, ISO/IEC, 2006.

[FL98] Flik, T.; Liebig, H.: Mikroprozessortechnik: CISC, RISC Systemaufbau Assembler und C. Springer-Lehrbuch. 1998.

[IE13] IEEE: Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7. IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013), S. 1–3906, April 2013.

[IS11] ISO: ISO/IEC 9899:2011 Information technology — Programming languages — C. International Organization for Standardization, Geneva, Switzerland, December 2011.

[IS12] ISO: ISO/IEC 14882:2011 Information technology — Programming languages — C++. International Organization for Standardization, Geneva, Switzerland, Februar 2012.

[JB12] John Barnes, Ben Brosgol: Safe and Secure Software - An invitation to Ada 2012. AdaCore, 2012.

[La79] Lamport, Leslie: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. IEEE Transactions on Computers C-28, 9:690–691, September 1979.

[Lt04] Ltd, MIRA: , MISRA-C:2004 Guidelines for the use of the C language in Critical Systems, Oktober 2004.

[Ma16] Mazidi, Muhammad Ali; Naimi, Sarmad; Naimi, Sepehr; Chen, Shujen: ARM Assembly Language Programming & Architecture (Volume 1). MicroDigitalEd.com, 2016.

[SR94] Shin, K.G.; Ramanathan, P.: Real-time computing: a new discipline of computer science and engineering. Proceedings of the IEEE, 82(1):6–24, 1994.

[St05] Stroustrup, B.: , The JSF air vehicle C++ coding standards, 2005.

[St13] Stroustrup, B.: The C++ Programming Language. Pearson Education, 2013.

[vDKV00] van Deursen, Arie; Klint, Paul; Visser, Joost: Domain-specific Languages: An Annotated Bibliography. SIGPLAN Not., 35(6):26–36, Juni 2000.

Kommunikationsprotokolle und deren Herausforderungen in eingebetteten Systemen

Long-range communication protocols for ISO/OSI Layers 1 and 2

Emil Fürniss [1], Krist Stoja [2]

Abstract: As the Internet of Things (IoT) expands, new low power wide area (LPWA) approaches are needed to meet the requirements of this sector. As IoT-requirements differ significantly from those of conventional connected devices such as computers and cellphones, the existing GSM/UMTS based wireless solutions are less than ideal. In response, many competing technologies have emerged in an attempt to balance power efficiency, range, signal robustness and quality of service. However, these solutions employ a wide range of different technologies in order to achieve the desired results, each with their own advantages and disadvantages. This has resulted in a very fragmented market, upon which this paper hopes to shed a little light by providing a general overview.

One solution based on existing cellular technology is being promoted by 3GPP, a consortium of cellular network providers who have adapted their technology to produce a solution called NB-IoT. However, the focus of this paper lies with the multitude of mostly proprietary LPWA solutions that have emerged: Most notably LoRa, Sigfox, Weightless and Ingenu. These offer solutions which were developed from the ground up to connect the budding IoT, often using unlicensed bands and proprietary modulation techniques in order to achieve the necessary range, low cost, and signal robustness.

This overview should not only help understand the current market, but also give an idea of the wide range of capabilities and applications that this LPWA paradigm enables.

Keywords: Internet of Things (IoT); Low-power Wide Area Networks (LPWAN); Machine-to-Machine (M2M) Communication

1 Introduction

The connected world is on the brink of an explosion. The so-called Internet of Things (IoT) describes the growing trend of connecting utilitarian devices such as sensors, household devices, manufacturing equipment, power grid networks, traffic control systems, transportation networks, and many more to the internet, thus allowing a level of control and monitoring hitherto unheard of. This explosion in connected devices will, by some estimates, double the number of devices connected to the internet by the early 2020s as seen in figure 1 [Ce16a].

As more and more devices are being connected to the internet, the need has arisen to develop a new set of solutions that meet the specific requirements of IoT-communications. After all,

[1] LMU Munich, Institute of Informatics, Oettingenstr. 67, 80538 Munich, Germany, emil.fuerniss@campus.lmu.de
[2] TU Munich, Department of Informatics, Boltzmannstr. 3, 85748 Munich, Germany, krist.stoja@tum.de

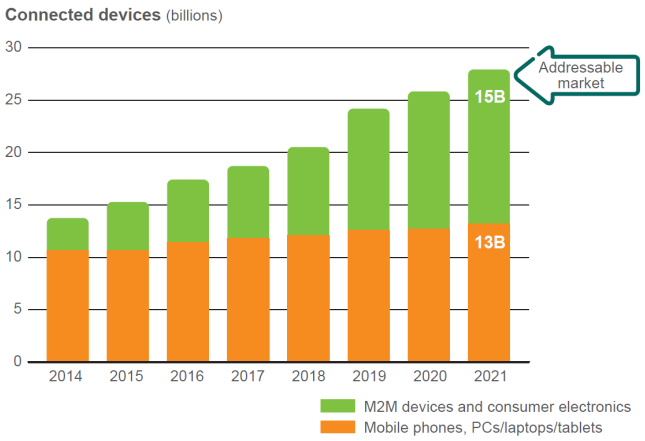


Fig. 1: Predicted growth in connected devices [Ce16a]

these devices have very different requirements than traditional wireless devices, such as mobile phones; most require very low data bandwidth as they are only transmitting simple data, such as sensor data. Many are also required to operate off a single battery for many years at a time, and so must be extremely energy efficient. Finally, the sheer number of devices expected to go online and a limited radio spectrum forces the issues of per-unit-cost, spectral efficiency and resistance to interference to be considered as well. And while the idea of low-power networking is not new, most technologies such as Bluetooth or Zigbee have sincere range limitations. Any attempts to implement longer range networks based on such systems have usually involved multi hop architectures. This severely limits efficiency, latency and scalability [Ba07].

With these issues in mind, several competing low-power wide-area network (LPWAN) solutions have been developed. Choosing between the wide ranges of technologies is complex and depending on their needs, the users must decide what is best for them in terms of features offered, and the benefits they have. They need to find a balance between power consumption, costs, range, capacity and directionality.

This paper will explore some of the technologies available and address some of their differences and applications. First, we will give a brief overview over some of the relevant background technologies which underpin wireless communications. Then we provide a more detailed explanation of several important technologies, namely Sigfox, LoRa, Weightless, Ingenu, and NB-IoT, before comparing them directly. Finally we provide two real-world examples where such systems have been implemented before concluding with a future outlook.

2 Background technologies and challenges

Before diving into the specifics of existing implementations, it is worth looking at some of the underlying ideas and technologies which are being employed in the various LPWAN solutions. As this is still a relatively new field and the requirements are so different from those of the past, there are many different solutions, often with wildly different approaches to the same problems. The result is a fragmented spectrum of technologies, each with their own advantages and disadvantages. [BDK16]

2.1 Frequency bands and modulation

All wireless telecommunications use some section of the radio spectrum. For our purposes, this spectrum is divided into two major segments: licensed frequency bands which are the exclusive domain of the respective license holder, and unlicensed frequency bands which are free to use, but require the user to adhere to strict power and interference guidelines. [Ce16b].

In order to fulfill the low-cost requirement, many systems choose to operate in the unlicensed spectrum, as the licensed spectrum is very expensive. Specifically, most LPWANs operate in the unlicensed ISM bands (industrial, scientific, and medical radio band). Additionally, the main advantage of a licensed bands, namely the ability to use it in an unrestricted and competition-free manner, is less relevant to LPWANs, as the data rates of most machine-to-machine (M2M) communications are so low, especially compared to cellular data traffic [SIC16].

Irrespective of the bands being used, there is always a need for very robust signal modulation. This is down to two main reasons: To accommodate the noise and interference generated by other users, and the signal degradation caused by sending low power signals over long distances. These two factors result in an exchange: quiet rural areas, which are mainly restricted by the natural signal degradation, allow for high transmission distances (usually 10-50 km), but noisy urban environments will only allow for shorter transmission distances (usually 3-10 km) [Ce16b]. While the specifics of the various modulation techniques applied across the various solutions are beyond the scope of this paper, the question of how wide a channel to use is worth a brief overview.

2.2 Comparing wide- and narrowband

When designing radio communication standards with energy efficiency, spectral efficiency and signal robustness in mind, two distinct options as to how to spread the signal across the available spectrum have emerged:

One option is to use a wide channel of frequencies several hundred kHz wide in combination with spread spectrum modulation. Depending on the scenario, the channels' width can offer very flexible data rates, but if throughput is not a priority, it can be traded for superior resistance to external interference and noise - a requirement which is especially important for IoT-devices operating in unlicensed bands [VZZ15, Ad17].

The alternative is to use narrow band modulation. It attempts to maximize spectral efficiency by restricting itself to a narrow channel, usually less than 25 kHz. This allows for multiple narrow band networks to operate efficiently in the same area, as they can use separate channels and noise interference is minimal [KH16]. Some implementations take this idea even further by limiting their channels to a few hundred Hertz, then usually designated Ultra Narrow Band (UNB). The downside is that each such channel can only carry very low volumes of data, to the point where a channel can even be forced to operate unidirectionally [RKS17].

3 Comparison of existing technologies

The following section aims to provide a more detailed insight into the various LPWA network solutions. Nonetheless, all are chasing the same objective: finding the ideal balance of LPWA functionality, such as bandwidth, power consumption, range, and costs.

3.1 Sigfox

Sigfox is one of the oldest and biggest LPWA technologies in IoT and is an entirely closed and proprietary network. Its network utilizes a low power and ultra-narrow band connection, which only allows for very low data transmission rates: Each device can send up to 140 messages per object per day, with a payload of 12 bytes for message. However, Sigfox claims that this is ideal for many machine to machine type communications, as many applications in all areas of IoT (home and consumer goods, healthcare, transportation, security, retail, etc.) do not require a lot of data bandwidth.

Its UNB radio modulation (Sigfox operates within a 0.2 kHz band) and the low transmission speed of only 100 to 600 bits per second allows for long distance transmissions while maintaining good resistance to interference due to very low noise, and allowing up to 1 million devices to be connected to a single gateway at once. Additionally, the signals are very good at penetrating solid objects, making it possible to connect systems which are inside buildings or buried underground. However, because it was developed with uploads as the priority, Sigfox is not designed for use cases where downloads are of any greater importance. Early versions were even limited to upload only and to this day, the extreme uplink limitations mean that Sigfox does not support firmware updates through its network [RKS17].

In order to reduce energy consumption and costs even further, Sigfox has consolidated the software based management of its entire system into a cloud based solution. This also means that if a consumer wants to use Sigfox, they are obliged to buy access to the Sigfox infrastructure and cloud platform as a service. It uses a single-hop star network topology, and the devices are not attached to a specific base station but blindly send their messages to all base stations within their range. The system then analyzes all received signals and chooses the one with the best reception [KH16, RKS17]. This mode of operation is popular among LPWAN solutions and is also used by LoRa, as it enables the networks to receive the best signal possible, irrespective of unforeseen outside disturbances. This also keeps the initial link to the device as energy efficient as possible and considerably increases the networks scalability and serviceability, as devices do not need to be linked to specific base stations or gateways.

The technology is mostly used in Western Europe (868 MHz) and some parts of the USA (900 MHz) [KH16], while pilot programs are in progress in South America and Asia [Co17].

3.2 LoRa & LoRaWAN

Contrary to some of the other technologies, the LoRa Alliance follows a relatively open approach as its hardware can be purchased and deployed as a third-party network. The rights to produce LoRa hardware were sold to the American Semtech Corporation in 2012, which is now the only company licensed to produce and sell the chip sets.

LoRa is the physical layer technology and employs a proprietary wide band Chirp Spread Spectrum (CSS) radio modulation technology, which helps signal resistance to interference, requires less power, and gives its protocols the possibility to transmit data with signal strengths below the noise floor. The noise and interference concerns are especially relevant, as LoRa usually operates in unlicensed ISM-Bands. One of the MAC layer protocols implemented on the top of its transmission modulation technology is LoRaWAN, a simple but robust unslotted ALOHA based server side implementation of multiple access protocol designed to minimize collisions. Studies evaluating the real world performance and reliability of LoRa and LoRaWAN have indicated a package delivery ratio of 96.7% [RKS17].

The LoRa network architecture has three main components in addition to the devices themselves, as illustrated by figure 2: the application server, the network server and the gateways. The gateways are a transparent bridge for bidirectional transmissions between end devices and the central server, where the customer-specific logic is built in. In order to enhance communication between end devices and to increase the capacity of the gateway, different frequency channels and data rates are used, which do not interfere with each other.

LoRaWAN also defines three levels of service, depending on the needs of different use cases: Class A must be implemented by every device and enables basic, but highly efficient

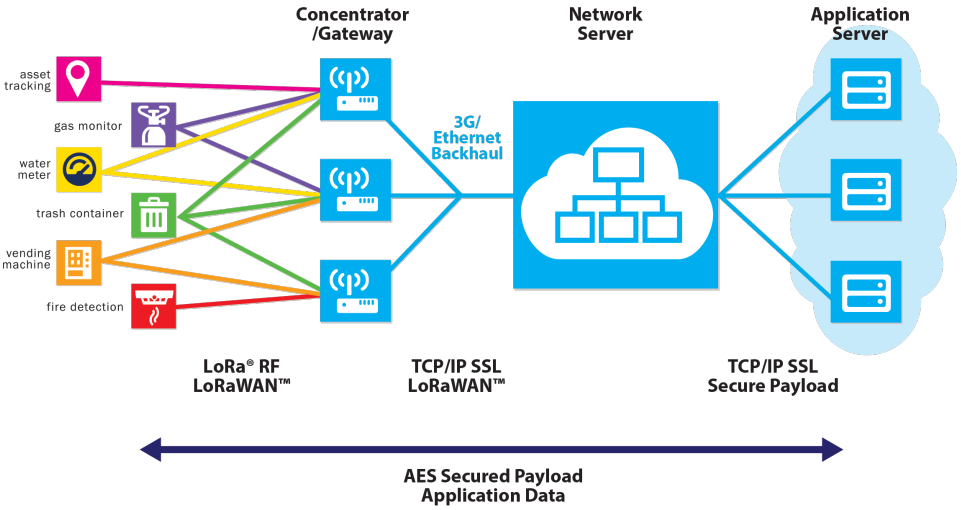


Fig. 2: The basic LoRa network architecture [Lo15]

ALOHA-based bi-directional communication, whereby the downlink is restricted to two short windows after each uplink. Class B is an extension which enables regular scheduled receive windows for downlink messages thanks to time-synchronization with the gateway. Finally, Class C is a further extension which keeps the receive window open continuously unless the device is transmitting [VZZ15].

LoRaWAN uses different frequency bands depending on the region. It uses the 863-870 MHz and 433 MHz bands in Europe, 902-928 MHz in the USA, 915-928 MHz in Australia, and 779-787 MHz and 470-510 MHz in China [VZZ15].

3.3 Weightless

Weightless is another open standard whose developers, a company from the UK, were recently acquired by Huawei [VZZ15]. In order to support different modalities and use cases, Weightless offers three different protocols: Weightless-N, Weightless-W and Weightless-P. Weightless-W is the most niche of the three, as it is only of interest for areas where the TV white space spectrum is available. Weightless-N is the middle ground, intended for narrow-band solutions such as sensor based networks. It operates as a competitor to Sigfox and uses a similar modulation type. Meanwhile, Weightless-P is the flagship of the three, offering the greatest range and bandwidth [Wh17].

As shown in the comparison chart (figure 3), Weightless-P not only offers higher performance than Weightless-N, but can also operate in full duplex mode. Both Weightless-P and

	Weightless-N	Weightless-P	Weightless-W
Directionality	1-way	2-way	2-way
Feature set	Simple	Full	Extensive
Range	5km+	2km+	5km+
Battery life	10 years	3-8 years	3-5 years
Terminal cost	Very low	Low	Low-medium
Network cost	Very low	Medium	Medium

Fig. 3: Comparison of Weightless protocols [Wh17]

Weightless-N use a narrow 12.5 kHz band and combine two types of modulation in order to achieve maximum power efficiency and signal integrity: Gaussian Minimum Shift Keying (GMSK), which optimizes the efficiency of the amplifier, and Quadrature Phase Shift Keying (QPSK), which eliminates most of the interference.

Using both Frequency Division Multiple Access (FDMA) and Time-division Multiple Access (TDMA) channel access methods in ISO/OSI layer 2, it claims to offer an optimal capacity for up-link dominated traffic from a large number of devices [RKS17].

3.4 Ingenu

When Ingenu started in 2008 under the name ”On-Ramp Wireless” it was mainly focused on utilities. Over time it extended into oil and gas applications, and in 2015 it changed its name and received support from big names in the cellular industry such as Verizon and Qualcomm, shifting its focus to making a public machine wireless network dedicated to the Internet of Things [Fu17].

Ingenu uses a patented technology called Random Phase Multiple Access (RPMA) modulation for its uplink transmissions. RPMA is an implementation of Direct Sequence Spread Spectrum (DSSS) modulation and a variation on the classic Code Division Multiple Access (CDMA), resulting in wide band communication which is resistant against outside interference and allows for multiple devices to transmit at the same time. Ingenu is also something of an exception as it operates in the 2.4 GHz band. This frequency range was chosen because in addition to it being globally available and free of charge, regulations in many regions allow for higher energy signals in this range compared to other frequency ranges, which Ingenu claims can cancel out the enhanced signal degredation caused by the the higher frequency. In comparison to LoRa and Sigfox, operating at 2.4 GHz enables Ingenu to provide higher transmission rates (up to 624 kbps download speeds and up to 156 kbps upload speeds), but at the cost of higher power consumption, which is a problem mostly for the receiving device. Another downside to this band is more spreading loss from obstructions, which may be balanced out by the higher power signals. Its network architecture is also different to that of Sigfox and LoRaWAN, as each device is connected

to a specific base station, necessitating a handover when moving between base stations [RKS17].

3.5 NB-IoT

Defined by the 3rd Generation Partnership Project (3GPP), a collaboration of traditional cellular network operators, Narrow Band IoT (NB-IoT) is an attempt to adapt GSM/UMTS technology and hardware to the requirements of the IoT market. It is the newest standard on the market, with the 3GPP completing its standardization in mid 2016 [St17]. Technically NB-IoT is only one of 3 IoT standards defined by the 3GPP, but both the LTE enhancements for Machine Type Communications (eMTC) and Extended Coverage GSM (EC-GSM) have significantly higher bandwidth and power requirements, which results in them occupying a different segment of the market and are not suitable for direct comparison with Sigfox, LoRa, and Ingenu.

However, NB-IoT operates in existing LTE bands, often alongside other LTE services, but is limited to around 250 kbps downlink and 20 to 250 kbps uplink. Its deployment is very flexible as a simple software update can enable existing base stations to support NB-IoT communications. Also, it can be deployed in a 180 kHz wide LTE band, a 200 kHz GSM band or a LTE guard band. These bands are all wide compared to UNB, but narrow compared to conventional LTE services, which frequently operate in 20 MHz wide bands. The 3GPP claim that NB-IoT can result in battery lifetimes of up to 10 years with a 5 Wh battery and allow up to 50.000 devices per cell [Fl16, RKS17].

4 Comparison

As we can now see, the LPWAN market is very fragmented, with many similarities and differences between technologies. As a result, it offers a wide range of choices which have to be filtered and matched to the induvidual use case. Before comparing the solutions directly from a practical point of view, it must be noted that there have been very few industry deployments or independant reviews of Weightless-P, Ingenu, and NB-Iot, and very little has been published about the few deployments that do exist. This makes it difficult to objectively compare these technologies. Sigfox and LoRa are a little more established and thus easier to compare.

The most obvious performance indicators are listed below in table 1 and give an initial overview. The main point of difference in this table are the data rate and, to a lesser degree, the range of each solution. This makes it appear as a fairly straight-forward choice, but there are many more subtle differences to consider.

While Sigfox is a highly consolidated solution which allows customers to buy into an existing ecosystem, LoRa is an open ecosystem and requires the implementation of an individual

network and user-specific hardware. The initial deployment costs are thus much higher with LoRa than with Sigfox, though the flexibility of developing and owning a tailored network is surely an advantage to some users. This is one area where the lack of industry deployments make the other solutions especially hard to compare.

Also hard to compare, even for Sigfox and LoRa, is the real-life battery life of the end devices. There has been very little independant research in this very important area and most solutions claim a similar battery life, usually 8-10 years . At the time of writing and to the knowledge of the authors, no independant study comparing the battery performance has taken place.

Meanwhile, range and penetration of solid objects are strong points of Sigfox, with its robust modulation making the connection inside buildings or underground possible, while also being resistant to interference. LoRa on the other hand offers a flexible data rate which can be orders of magnitude higher than that of Sigfox. It also allows for fully symmetical communications, while Sigfox only has marginal downlink capacities. Weightless-P claims a flexible data throughput to rival LoRa, while Ingenu and NB-IoT have the highest uplink rates.

With respect to topology, all solutions presented use a similar star topology, but the way the end devices connect to the gateways are quite different. Sigfox and LoRa end devices are not connected to specific gateways. Instead, their signal is recieved by all gateways in range and the strongest signal is then chosen. This allows the end device to easily move between gateways. Ingenu end devices in comparison, are connected to a specific gateway and require a handover when moving between them.

The final differentiating factor to be discussed here, is the frequency band in which each technology operates. In this respect, Ingenu stands out from the crowd as it is the only one to operate at 2.4 GHz. This compares to the mainly sub-GHz operating frequencies of LoRa, Sigfox, and Weightless, though if this design decision carries any real-world benefits has yet to be independantly confirmed. Other than NB-IoT with its basis in cellular technology, all other IoT-specific solutions operate in unlisenced bands.

	Sigfox	LoRa	Weightless-P	Ingenu	NB-IoT
Range	10 km (urban), 50 km (rural)	5 km (urban), 15 km (rural)	2 km (urban), 10 km (rural)	15 km (urban)	10 km (urban)
Data rate	600 bps (up), 100 bps (down)	0.3 - 37 kbps (symmetrical)	0.2 - 100 kbps (symmetrical)	624 kbps (up), 19.5 kbps (down)	250 kbps (symmetrical)
Battery life	10 years	10 years	10 years	10 years	10 years

Tab. 1: Basic performance comparison of LPWAN solutions [RKS17, VZZ15]

5 Real-world deployment examples

LPWA technology naturally has many applications in the private sector. However, the nature of wide area systems is of particular interest to many challenges faced by the public sector, as the nature of the public sector involves providing services and monitoring to large areas and entire populations. As such, the concept of a smart city has been gaining popularity, describing the idea of enhancing a cities efficiency by connectiong and controlling many aspects of public infrastructure. In this section, we will be giving two examples of cities that have deployed such systems [Ce16b].

5.1 inteliLIGHT LoRa Streetlight Control Solution

Beginning in late 2015, the small Hungarian town of Szada, located just outside Budapest, played host to a trial deployment of a LoRa-based street lighting control developed by inteliLIGHT. The aim was to verify the functionality, feasibility and reliability of such a smart city system in convenient proximity to the capital, before attempting deployment on a larger scale. The initial setup, as illustrated in figure 4, included around 500 street lights connected by wire in small groups to an inteliLIGHT LoRa controller. These controllers communicate with a LoRa gateway, which in turn communicates with a LoRaWAN network server which is linked to the client. According to inteliLIGHT, installing all the hardware only took 3 days and the central gateway was installed on the highest suitable building in the town in order to provide optimal coverage.

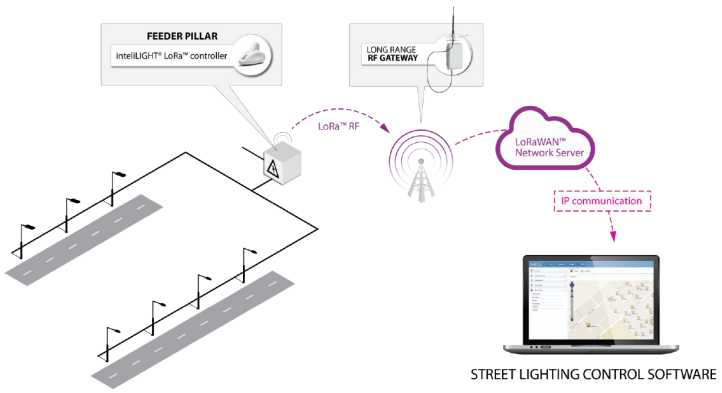


Fig. 4: Network architecture of the inteliLIGHT system [In16]

inteliLIGHT claims up to 80% energy savings by providing the ability to control lamps individually and that each gateway can connect up to 20.000 controllers, thus providing excellent scalability. [In16]

5.2 Environment monitoring in Getaria for tourist purposes with Sigfox

In the picturesque coastal town of Getaria in the Spanish province of Gipuzkoa in Basque Country, tourism is a major contributor to the local economy. In order to assist the development of the tourism sector, they decided to monitor various environmental factors such as air pollution, noise levels and water quality across the municipality, allowing them to react faster to disturbances and issues that might affect the attractiveness of the town. To accomplish this, they employed a Sigfox-based network of sensors, implemented by the Spanish companies Libelium, a manufacturer of sensor hardware, and Nexmachina Solutions, a company specializing in monitoring and managing wireless sensor networks. The initial deployment includes the monitoring of noise levels at 10 leisure locations, with a focus on monitoring the night life in the city center. The water quality monitoring focuses on pollutant spills that affect the beaches, port and rainwater locations. Finally, the air quality monitoring is an attempt to ensure European standards for CO_2, NO_2, O_3, SO_2 and particle values in areas of risk [Ge17].

6 Conclusion

Given the rapid expansion of the IoT-market outlined in the introduction, we can understand the importance of developing the solutions outlined above. We have seen that by focusing on the specific requirements of IoT-communications and the need to connect millions of new devices through the various networks, LPWANs are able to offer solutions tailored to the very specific needs of this booming industry. The diversity of these technologies should allow them to cover the wide spectrum of IoT use cases, and allow potential users to choose the specification which best suits their needs.

However, it has become clear than more independant review and real-world experiences are needed to confirm and compare the performance claims made by the individual developers, especially in relation to the critical metric of battery life.

The myriad of subtly different solutions also produce a very fragmented market, which is in dire need of standardization for the sake of clarity. This results in a future of LPWAN which is not easy to predict, especialy with new standards like NB-IoT entering the market. Financially backed by big cellular network providers and running on existing hardware, they pose a potential threat to current dominant LPWAN providers such as LoRa and Sigfox, who are forced to build an entire new hardware infrastructure from scratch.

Despite that, it is the opinion of the authors that technologies such as Sigfox and LoRa are unlikely be substituted any time soon. They offer a wide range of advantages and already have many devices connected through their technology, thus proving their worth in the market. They are sure to have a big influence on the future of IoT and its countless applications.

References

[Ad17] Adelantado, F. et al.: Understanding the Limits of LoRaWAN. IEEE Communications Magazine, 55(1), 2017.

[Ba07] Baronti, Paolo et al.: Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. Computer communications, 30, 2007.

[BDK16] Boulogeorgos, A.; Diamantoulakis, P.; Karagiannidis, G.: Low Power Wide Area Networks for Internet of Things Applications: Research Challenges and Future Trends. arXiv:1611.07449v1 [cs.NI], 2016.

[Ce16a] Cellular Networks for massive IoT, Ericsson White Paper, 2016.

[Ce16b] Centenaro, M. et al.: Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios. IEEE Wireless Communications, 23, 2016.

[Co17] Comparison of LPWAN Technologies. https://www.digi.com/blog/iot/lpwan-technology-comparison, Accessed: 04/06/2017.

[Fl16] Flore, Dino: 3GPP Standards for the Internet of Things. GSMA MIoT, 2016.

[Fu17] Future of Work: Ingenu Machine Network. http://fortune.com/2015/09/11/ingenu-machine-network/, Accessed: 04/06/2017.

[Ge17] Getaria: environment monitoring for a smart tourist destination. http://www.libelium.com/getaria-environment-monitoring-for-a-smart-tourist-destination/, Accessed: 04/06/2017.

[In16] InteliLIGHT LoRa Case Study, 2016.

[KH16] Kalfus, Radim; Hégr, Tomáš: Ultra Narrow Band Radio Technology in High-Density Built-Up Areas. In: International Conference on Information and Software Technologies. Springer, pp. 663–676, 2016.

[Lo15] LoRaWAN What is it? A technical overview of LoRa and LoRaWAN, LoRa Alliance White Paper, 2015.

[RKS17] Raza, Usman; Kulkarni, Parag; Sooriyabandara, Mahesh: Low power wide area networks: An overview. IEEE Communications Surveys & Tutorials, 19, 2017.

[SIC16] Sanchez-Iborra, R.; Cano, M.: State of the Art in LP-WAN Solutions for Industrial IoT Services. sensors, 5, 2016.

[St17] Standardization of NB-IOT completed. http://www.3gpp.org/news-events/3gpp-news/1785-nb_iot_complete, Accessed: 04/06/2017.

[VZZ15] Vangelista, Lorenzo; Zanella, Andrea; Zorzi, Michele: Long-range IoT technologies: The dawn of LoRa. 2015.

[Wh17] Which Weightless Standard? http://www.weightless.org/about/which-weightless-standard, Accessed: 04/06/2017.

Mittelstreckenprotokolle der ISO/OSI Schichten 1 und 2

Lukas Grob[1]

Abstract:

Keywords: IEEE 802.15.4; Zigbee; Bluetooth Low Energy; eingebettete Systeme; Internet of Things

[1] Ludwig-Maximilians-Universität München l.grob@campus.lmu.de

IP und IPv6 im Internet der Dinge

Michael Hegel[1], Emre Bolat[2]

Abstract: Um Computer miteinander kommunizieren zu lassen, werden genaue Vorgaben benötigt. Eines der bekanntesten Protokolle ist das „Internet Protocol Version 4“ (IPv4), welches, aufgrund der Adressknappheit, durch die 1995 standardisierte Version 6 (IPv6) ersetzt werden soll. Doch auch wenn das IPv6-Protokoll viele Probleme beseitigt, ist es nicht geeignet, um Geräte mit geringer Leistung zu verbinden, deshalb wurde das „IPv6 over Low power Wireless Personal Area Network“ (6LoWPAN) Protokoll eingeführt. Wir verdeutlichen in diesem Paper die Funktionsweise von IPv6 und 6LoWPAN. Aufgrund ihrer unterschiedlichen Einsatzgebiete mussten bei der Spezifikation verschiedene Designentscheidungen getroffen werden, die wir in diesem Paper verdeutlichen. Um den Einschränkungen in Netzwerken, bestehend aus Embedded Systems, zu entgegnen, wurden bei 6LoWPAN unter anderem ein Adaption Layer, eine Header Komprimierung und eine Fragmentierung von Paketen eingeführt. Wir wollen mit unserer Arbeit einen ersten Einblick in die Netzwerkkommunikation von Embedded Systems und deren besondere Herausforderungen geben.

Keywords: Embedded Systems, IPv6, 6LoWPAN, Internet of Things (IoT), Netzwerkkommunikation

1 Einleitung

In den letzten Jahrzehnten haben sich die Geräte nicht nur im Design verändert, sondern auch von der Funktions- und Nutzungsweise. Ein Nutzer scheint schon fast überflüssig, betrachtet man die sogenannten Embedded Systems, die selbstständig untereinander kommunizieren. Dies geschieht innerhalb eines eigens geschaffenen Systems - dem Internet der Dinge (IoT). Definiert wird dieses als eine Art Vernetzung von alltäglich genutzten Embedded Systems, die über das Internet Daten empfangen und senden können. [Vgl. 1, S. 6] Es wird davon ausgegangen, dass im Jahre 2025 75 Milliarden Geräte an das IoT angebunden sein werden. [Vgl. 1, S. 5] Doch aufgrund der begrenzten Ressourcen von solchen Embedded Systems muss sich deren Kommunikation, von der normaler Computer, die meist die Protokolle IPv4 oder IPv6 nutzen, unterscheiden. Es gibt zwar bereits seit geraumer Zeit Lösungen, die es erlauben, leistungsschwache Geräte untereinander kommunizieren zu lassen. Der große Nachteil hierbei ist jedoch, dass keine Kommunikation über das Internet möglich ist. An dieser Stelle kommt 6LoWPAN zum Einsatz, welches einen Nachrichtenaustausch mit IPv6-Geräten außerhalb des lokalen Netzwerks erlaubt. Dieses Paper soll zunächst Grundlagen des IPv6 Standards und anschließend die Prinzipien und Abläufe des 6LoWPAN Protokoll darlegen.

[1] Email: michihegel@gmail.com
[2] Email: emre-bolat@hotmail.de

2 Internet Protocol Version 6 - IPv6

Das weitverbreitete Internetprotokoll IPv4 wurde in den 1980er Jahren entwickelt und benutzt 32-Bit-Adressen, was ungefähr vier Milliarden Adressen entspricht. Die Weltbevölkerung im Jahre 1980 zählte knapp 4,5 Milliarden Menschen. Demnach stand bereits früh fest, dass IPv4 keine endgültige Lösung darstellt und die letzten IPv4-Adressen zwischen 2009 und 2011 durch die Internet Assigned Numbers Authority (IANA) vergeben werden. [Vgl. 2] Abhilfe schaffte das im Jahre 1995 standardisierte Internetprotokoll IPv6, das trotz seinem 22-jährigen Bestehen für viele Menschen immer noch eine Neuheit darstellt. Anders als IPv4 deckt dieses Protokoll einen Adressraum von 2^{128} Adressen ab. Damit können also 340 Sextillionen IPv6-Adressen vergeben werden und somit ist nicht nur das Problem der IPv4-Adressknappheit gelöst, sondern auch die Menschheit für eine lange Zeit mit Internet-Adressen versorgt.

2.1 Adressierung mit IPv6

Notiert werden IPv6-Adressen, wie auch IPv4-Adressen, nach dem CIDR Verfahren. Sie bestehen demnach aus einer Netzadresse und einem Präfix, das, falls notwendig, mit einem „/“ angehängt wird. Das Präfix wird dazu benutzt, um das Subnetz zu kennzeichnen. [Vgl. 3, S. 4]

```
2001:0db8:0001::/48
```

IPv6-Adressen werden als 32 Hexadezimal-Ziffern dargestellt, die als 8 Blöcke von jeweils 4 Hexadezimal-Ziffern durch Doppelpunkte getrennt werden. Zudem können, wie im folgenden Beispiel verdeutlicht, führende Nullen weggelassen werden und genau ein Block von Nullen kann durch zwei Doppelpunkte ersetzt werden. [Vgl. 3, S. 3]

```
2001:0db8:0001:0000:0000:0000:BBBB:0011
2001:0db8:1111::BBBB:11
```

Jedem Netzwerk-Interface wird durch die IPv6-Adressierung mindestens eine, in der Regel jedoch drei Adressen zugewiesen, welche in Unicast, Multicast und Anycast gegliedert werden können.

2.1.1 Unicast-Adressen

Eine *Unicast-Adresse* bezieht sich auf einen einzigen Knoten bzw. ein einziges Interface, baut eine Punkt-zu-Punkt-Verbindung mit der Zieladresse auf und leitet das Paket nur an das Interface derselben Adresse weiter. Unicast-Adressen gelten international, sind providerbasiert und beginnen mit dem Präfix 001. Durch die Vergabe verschiedener Präfixe in den jeweiligen Arten der Adressierung wird das Routing simplifiziert. Die

Entscheidungsfindung kann nunmehr allein durch dieses Präfix stattfinden. Der danach folgende, maximal 45 Bit große Bereich, wird auch als *Global Routing Prefix* bezeichnet und dem öffentlichen Bereich zugesprochen, auf dem sich im ersten Feld Informationen zur internationalen, im zweiten zur nationalen Registrierungsbehörde und im letzten zum Anbieter finden. [Vgl. 4, S. 1] Das erste dieser drei Felder besitzt eine feste Größe von 5 Bit, die der restlichen beiden können von den jeweiligen Behörden selbst entschieden werden und haben demnach eine variable Größe. Die *Subscriber-ID* speichert die Informationen über die Einrichtung, von der die Provider die Adresse beziehen und ist gleichzusetzen mit der IPv4-Netz-ID. Abschließend folgt die *Subnet-ID*, die das Subnetz kennzeichnet, und die *Interface-ID*, für die oftmals die MAC-Adresse eines Gerätes, die dieselbe Größe wie das Feld besitzt, benutzt wird. [Vgl. 3, S. 7]

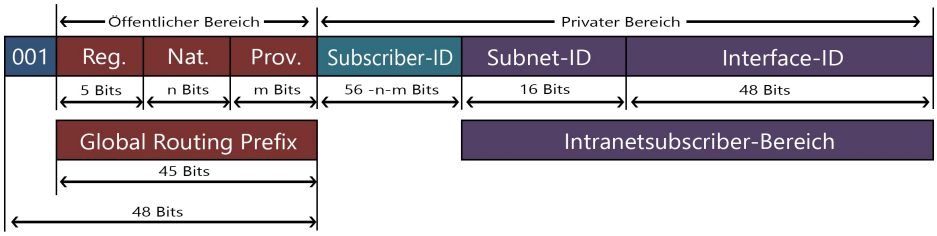


Abb. 1: Struktur einer Unicast-Adresse für Provider [Vgl. 4]

Eine andere Art von Unicast-Adressen stellen verbindungslokale oder auch Link-Local-Unicast (LLU) Adressen dar, die hauptsächlich bei der Autokonfiguration benötigt werden. Pakete können mit diesen Adressen nicht weitergeleitet werden und sind nur im Local Access Network (LAN) erreichbar, da sie keine Informationen zu Subnetzen enthalten und deshalb nur in isolierten Subnetzen verwendet werden können. [Vgl. 3, S. 10] Alle anderen Adressräume enthalten Adressen für globale Netze, auf deren Funktionsweise aber im Rahmen dieser Arbeit nicht weiter eingegangen wird.

2.1.2 Multicast-Adressen

Die bei IPv4 benutzten Broadcast-Adressen werden im neuen Protokoll nicht mehr unterstützt. IPv6 nutzt stattdessen *Multicast-Adressen*, die dieselbe Funktion erfüllen. Eine Multicast-Adresse bezieht sich auf eine Gruppe von Knoten bzw. Interfaces und leitet das Paket an alle Interfaces der Gruppe weiter. Das Präfix dieser Adresse besteht nur aus binären Einsen, woraufhin das *Flag* Feld folgt. Annehmen kann dieses aber nur die Werte 0000 und 0001. Die ersten drei Bits enthalten demnach, wie in Abbildung 2 zu sehen, in jedem Fall Nullen. Falls die aktuelle Multicast Adresse eine permanente Adresse darstellt, steht anstelle von T die Ziffer 0, bei einer temporären Adresse eine 1. [Vgl. 3, S. 12] Im *Scope* Feld kann eingestellt werden, ob zum Beispiel ein Paket weltweit oder nur innerhalb des Subnetzes weitergeleitet wird. In den restlichen 112 Bits finden sich Informationen über die Gruppe von Knoten, an die Pakete geschickt werden sollen. [Vgl. 3, S. 12-13]

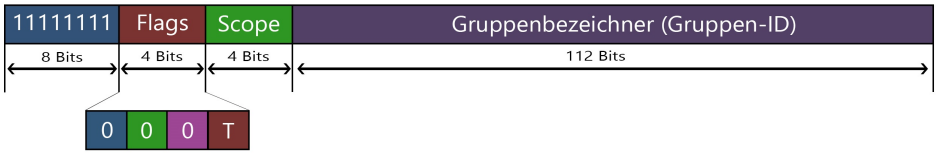


Abb. 2: Struktur einer Multicast-Adresse [Vgl. 3, S. 13-15]

Der Vorteil dieses Typs liegt darin, dass sichergestellt wird, dass Pakete nur einmalig übertragen werden und die Auswertung in der IP-Schicht vollzogen wird, um Bandbreite einzusparen. Genutzt werden diese Adressen zum Beispiel für IPTv oder Videokonferenzen zwischen mehreren Knoten. [Vgl. 3]

2.1.3 Anycast-Adressen

Eine *Anycast-Adresse* bezieht sich ebenfalls auf eine Gruppe von Knoten bzw. Interfaces, leitet aber das Paket an den nächstgelegenen Knoten der Gruppe weiter. Der Aufbau dieses Typs ist wie der einer Unicast-Adresse (siehe Abbildung 1), die jedoch von mehreren Geräten gemeinsam genutzt wird. Anycast-Adressen treten nicht häufig auf, da sie noch in der experimentellen Phase sind und deshalb nicht im *Destination Address* Feld stehen dürfen. [Vgl. 3, S. 11]

2.2 IPv6 Header

Bei der Umsetzung des *IPv6 Headers* wurde aufgrund der Zielsetzung der Entlastung der Router das Modulprinzip angewandt. Durch die Zuordnung einer festen Größe wird die Geschwindigkeit verbessert. Es muss nicht ein zuerst ein bestimmtes Feld gefunden und ausgelesen werden, um das Ende des Headers zu ermitteln. Der IPv6 Header besteht ohne Erweiterungen aus insgesamt acht Feldern und ist 40 Byte groß. Verglichen mit dem IPv4 wurde zwar die Headerlänge von mindestens 20 Byte auf 40 Byte erweitert, jedoch die Anzahl der Felder von 12 auf 8 minimiert. Der Grund dafür liegt in der Vergrößerung des Adressraumes. [Vgl. 5, S. 1] Die einzelnen Felder haben folgende Bedeutung:

- **Version:** Identifiziert die Version des Protokolls, wobei 0100 für IPv4 und 0110 für IPv6 steht, und ermöglicht so den unkomplizierten Übergang von IPv4 in IPv6 und die Zulassung neuer, experimenteller Protokolle. [Vgl. 5, S. 25]
- **Traffic Class:** Dient zur Priorisierung des Paketes. Aufgrund der vier Bit lassen sich mögliche 16 Prioritäten zuordnen. Die ersten acht Zahlen ab 0 dienen hierbei der Priorisierung von Nicht-Echtzeit-Anwendungen, wie zum Beispiel FTP oder E-Mail. Echtzeit-Anwendungen, wie beispielsweise VoIP, haben einen Anspruch auf die restlichen acht Zahlen. [Vgl. 5, S. 25]

- **Flow Label:** Ordnet das Paket einer Klasse zu, die dann vom Router anders behandelt werden kann. Das Ziel dieses noch nicht standardisierten Feldes ist es, eine Kennziffer als Hashcode zu definieren, die die Router unter anderem bei der Übertragung von Mediendateien unterstützt. [Vgl. 5, S. 25]
- **Payload Length:** Definiert die Länge des Paketes, wobei maximal 65536 Bytes an Nutzdaten übertragen werden können. Größere Daten können jedoch durch die Jumbo Payload Option weitergeleitet werden. [Vgl. 5, S. 3-4]
- **Next Header:** Verweist auf einen, keine oder mehrere vorhandene Extension Header. [Vgl. 5, S. 3-4]
- **Hop Limit:** Vergleichbar mit IPv4 TTL. Bei jedem Weiterleiten des Paketes durch einen Knoten, wird der Wert dieses Feldes dekrementiert. Bei 0 verfällt das Paket. [Vgl. 5, S. 3-4]
- **Source Address und Destination Address:** Jeweils 128 Bit großer Adressraum mit der Sender- oder Zieladresse. [Vgl. 5, S. 3-4]

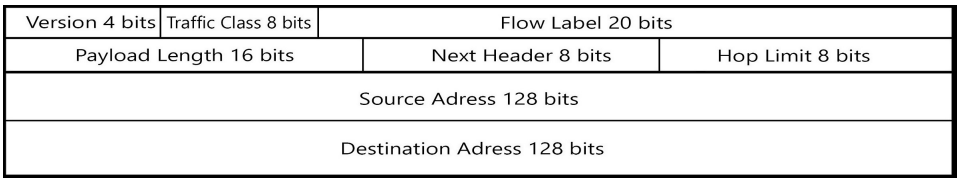


Abb. 3: IPv6 Header [Vgl 5, S. 3]

2.3 IPv6-Autokonfiguration

Die Autokonfiguration wurde unter IPv4 mit Dynamic Host Configuration Protocol (DHCP) Servern geregelt. Das Prinzip der Vergabe einer IP-Adresse an einen dem Netzwerk erstmals beigetretenen Rechner, durch einen speziellen Server, wurde für IPv6 zwar beibehalten, jedoch kann das neue Protokoll nun auch ohne DHCP, also komplett autonom, agieren, indem man über LLU-Adressen die eigene IP-Adresse von einem Router ermitteln kann. [Vgl. 6, S. 2] Die Realisierung mit DHCP in IPv6 wird auch als „Stateful Autoconfiguration“ bezeichnet, wohingegen die neuartige Lösung, die ohne weitere Protokolle auskommt, als „Stateless Autoconfiguration“ betitelt ist. [Vgl. 6, S. 6] Die Autokonfiguration findet in insgesamt sechs Schritten statt. Der erste beginnt damit, dass ein Gerät an ein Netzwerk angeschlossen wird und selbstständig eine LLU-Adresse erzeugt, deren Aufbau in Abbildung 4 zu sehen ist. Sie besteht aus dem Präfix 1111111010, 54 0-Bits und einer Interface-ID. In diesem Fall wird die MAC-Adresse des Gerätes übernommen.

Der nächste Schritt beinhaltet die Testung der generierten Adresse mittels Neighbor Discovery Protocol (NDP). Mit Hilfe des „Duplicate Address Detection“-Protokolls (DAD) wird eine sogenannte „Neighbor Solicitation“-Nachricht an alle benachbarten Geräte

Abb. 4: Aufbau einer LLU-Adresse [Vgl 3, S. 11]

gesendet, um zu ermitteln, ob die zuvor erzeugte Adresse bereits im Netzwerk enthalten ist. Falls dies zutrifft, wird eine „Neighbor-Advertisement"-Nachricht empfangen und es muss wieder ab dem ersten Schritt begonnen werden. Andernfalls wird nichts empfangen und mit dem nächsten Schritt fortgefahren, der diese Adresse dem Interface zuordnet. [Vgl. 6, S. 13-16] Damit wird nun garantiert, dass das Gerät im lokalen Netzwerk vollständig kommunizierfähig ist. Der vierte Schritt soll die Kommunikation, die bisher nur auf das LAN beschränkt ist, erweitern und global den Zugriff ermöglichen. Dies kann entweder passiv oder aktiv geschehen. Die passive Variante wartet auf die Nachrichten „Router Advertisments", bei der alle vorhandenen Router im Netz signalisieren, dass sie anwesend sind. Bei der aktiven Variante wird die Nachricht „Router Solicitation" versendet, bei der alle Router im Netz aufgefordert werden, sich per „Router Advertisments" zu melden. Falls letztendlich eine solche Nachricht empfangen wurde, wird der Router als „Default Router" deklariert. [Vgl. 6, S. 15-16] Entschieden wird nun im vorletzten Schritt, ob eine automatische Autokonfiguration möglich ist. Bei negativer Rückmeldung wird versucht auf manuelle Konfiguration bzw. DHCP zurückzugreifen. Falls eine „Stateless Autoconfiguration" möglich ist, wird zuletzt die Interface-ID aus der im ersten Schritt generierten LLU-Adresse mit dem Netzwerk-Präfix des Routers zu einer global gültigen Unicast-Adresse zusammengesetzt, mit der das Gerät nun endgültig mit dem Internet verbunden ist. [Vgl. 6, S. 16-19]

3 6LoWPAN

Für herkömmliche Netzwerke, die meistens aus vollwertigen Rechnern bestehen, ist eine Kommunikation mit Hilfe von IPv6 geeignet. Will man jedoch Embedded Systems, welche eingeschränkten Ressourcen haben, verwenden, wird man vor einige neue Herausforderungen gestellt. Solche Netzwerke bestehen aus vielen einzelnen Geräten, die oft von einer Batterie betrieben werden, eine geringe Bandbreite aufweisen, ihren Standort wechseln können und aufgrund des möglichst geringen Stückpreises über begrenzte Rechenleistung und (Arbeits-)Speicher verfügen. All diese Einschränkungen machen das IPv6 Protokoll sehr impraktikabel, was zu diversen Netzwerk Architekturen, wie beispielsweise ZigBee geführt hat. Diese hatten jedoch das Problem nicht nativ über IP zu kommunizieren, was zu einem Mehraufwand in den Gateway Router führt. All das wurde bei der Entwicklung des „IPv6 over Low power Wireless Personal Area Network", kurz 6LoWPAN, berücksichtigt. [Vgl. 7, S. 1]

In der folgenden Abbildung wird der Aufbau eines kompletten Systems mit drei Varianten dargestellt. *Simple 6LoWPAN* vereinigt Geräte, die denselben Präfix besitzen. *Ad hoc*

6LoWPAN hat die Besonderheit, nicht mit dem Internet verbunden zu sein und ohne eine Infrastruktur zu arbeiten. *Extended 6LoWPAN* verbindet mindestens zwei Edge Router mittels eines Backbone Links. LoWPANs befinden sich meist am Rand bestehender Netzwerke, wobei die Edge Router für den Datenaustausch sorgen. [Vgl. 12, S. 13-15]

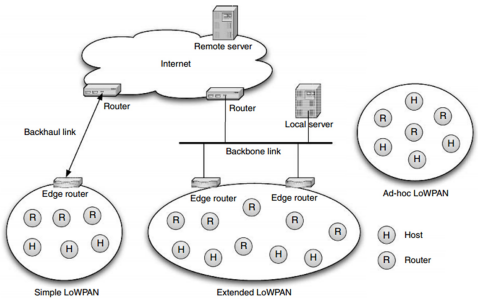


Abb. 5: 6LoWPAN Topologie [12, S. 14]

3.1 Adaption Layer

Um die Übertragung von IPv6-Paketen in IEEE 802.15.4 Frames zu ermöglichen, definiert das RFC 4944 einen Adaption Layer, welcher mit Hilfe verschiedener Header die zu übermittelnden Daten kapselt. Die unterschiedlichen Header können anhand des ersten Bytes - dem Header Type - eindeutig erkannt werden, [Vgl. 7, S. 4-5] wobei nicht immer das gesamte Byte ausschließlich der Typerkennung dient. Für den Fall, dass mehrere Header benötigt werden, müssen sie in der folgenden Reihenfolge angeordnet sein: mesh addressing header, broadcast header, fragmentation header. [Vgl. 8, S. 9] Einige der in RFC 4944 beschriebenen Header werden in den jeweiligen Kapiteln genannt und beschrieben.

3.2 6LoWPAN-Autokonfiguration

Es ist möglich, dass ein Knoten automatisch einen IPv6 Interface Identifier (IID) erstellt. Hierfür kann entweder die dem Gerät zugewiesene EUI-64 Adresse oder die 16 Bit Short-Adresse verwendet werden. Im ersten Fall wird lediglich das siebte Bit - das Universal/Local Bit -, welches angibt ob eine Adresse global „1“ oder lokal „0“ eindeutig ist, invertiert. Falls die Generierung aus einer Short Adresse stattfindet, muss diese auf 64 Bit erweitert werden, indem zunächst 16 Null-Bits an die 16 Bit lange PAN ID und anschließend die 16 Bit Short Adresse angehängt werden. Es ergibt sich folgendes Schema: [Vgl. 8, S. 13]

$$16\_Bit\_PAN\_ID : 16\_Null\_Bits : 16\_Bit\_kurz\_Adresse$$

Da die Short Adresse nur so lange gültig ist, wie eine Verbindung mit dem PAN Koordinator besteht, stellt sie einen Single Point of Failure dar. Bricht die Verbindung ab, wird

möglicherweise eine neue Short Adresse zugewiesen, wodurch der IID nicht länger gültig ist. [Vgl. 8, S. 4] Die IPv6 link-local Adresse wird anschließend durch das Anhängen des IID an das Präfix FE80::/64 gebildet. [Vgl. 8, S. 14]

3.3 Routing

Die Routingverfahren können entweder auf dem Link Layer oder dem Network Layer realisiert werden. Ersteres wird als mesh-under und letzteres als route-over bezeichnet. [Vgl. 7, S. 13]

In einem mesh-under Netzwerk werden die Routing-Entscheidungen auf Basis der Link Layer Adressen getroffen. [Vgl. 7, S. 13] Um das Weiterleiten von Nachrichten über mehrere Knoten auf Schicht zwei zu ermöglichen, wird der in Abbildung 6 gezeigte Mesh Addressing Header genutzt.

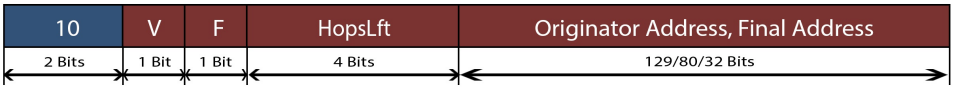


Abb. 6: Mesh Adressing Header [Vgl. 8, S. 10]

Die ersten beiden Bits zeigen an, dass es sich um einen Mesh Header handelt. Der Ursprungsknoten wird als Originator (oder Very First) und derletzten Empfängers als Final Destination bezeichnet. Da hierbei sowohl EUI-64- als auch Short Adressen zur Identifizierung möglich sind, zeigen Bit drei und vier an, welche der beiden Optionen jeweils auf den Originator und die Final Destination zutrifft. HopsLft entspricht dem Hop Limit von IPv6. [8, S. 10]

In einem mesh-under Netzwerk gibt es nur einen IP Router, den Edge Router. Um die Kompatibilität mit einigen Protokollen höherer Schichten, wie zum Beispiel der Duplicate Address Detection des IPv6-Protokolls, sicherzustellen, wird mit Hilfe des in RFC 4944 beschriebenen Broadcast Headers, eine Broadcast Domäne aufgebaut. [Vgl. 7, S. 13] Im Gegensatz dazu wird in einem route-over Netzwerk das Weiterleiten von Dateneinheiten auf dem Network Layer, mit den dort anzusiedelnden IP Adressen, realisiert. Bei diesen Verfahren stellt jeder Knoten einen IP Router dar, was es ermöglicht IP Routing Tables und Hop-by-Hop Optionen zu nutzen. [Vgl. 9, S. 1209-1210] Dies führt zu einer deutlich leichteren Integration in bestehende IP Netzwerke, jedoch muss jedes Gerät die Funktionen eines Routers, wie die bereits genannte DAD, unterstützen. [Vgl. 7, S. 13]

3.4 Fragmentierung

Um den Transport von IPv6 Paketen, deren Spezifikation eine MTU von mindestens 1280 Byte fordert, über maximal 127 Byte große 802.15.4 Frames, zu ermöglichen, müssen die IPv6-Pakete bei Bedarf fragmentiert werden. Hierfür gibt es zwei Fragmentation Header,

einen für das erste Fragment, der durch das Muster „11000“ signalisiert wird und sich von dem zweiten, in Abb. 7 zu sehenden, Header, welcher für alle weiteren Fragmente verwendet wird, zudem durch das Fehlen des Datagram Offset, unterscheidet. [Vgl. 8, S. 11]

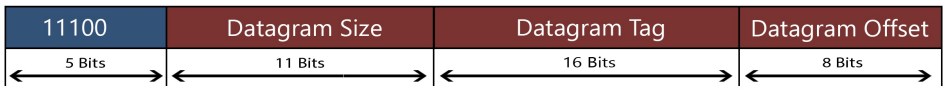


Abb. 7: Fragmentation Header [Vgl. 8, S. 11]

Bei beiden Headern geben die ersten fünf Bit den Typ an, Datagram Size entspricht der Länge des IP-Pakets vor der Link Layer Fragmentierung, Datagram Tag dient der zusätzlichen Unterscheidung von Fragmenten verschiedener IP-Pakete und Datagram Offset entspricht dem Versatz, den das Fragment im Bezug zum gesamten IP Paket hat. Dabei ist zu beachten, dass der Offset lediglich Vielfache von acht Byte angibt und somit alle Fragmente, bis auf das Letzte, eine durch acht Byte teilbare Länge haben müssen. [Vgl. 8, S. 11-12]

In Abhängigkeit des Routingverfahrens kommt es bei der Übertragung zu unterschiedlichen Verhalten. So werden bei einem mesh-under Netzwerk Fragmente unmittelbar, über eventuell unterschiedliche Routen, weitergeleitet, bis sie letztendlich an ihrem Ziel ankommen. Sind alle Fragmente am Zielknoten angekommen, werden sie wieder zu einem IPv6-Paket zusammengefügt. Falls jedoch eines der Fragmente verloren geht, muss das gesamte IPv6-Paket erneut übertragen werden. Bei einem route-over Netzwerk hingegen werden die Fragmente jeweils nur einen Hop weit geschickt, wo sie wieder zu einem IP-Paket zusammengefügt und dem Network Layer übergeben werden. Dieser überprüft, ob und wohin das Paket weitergeleitet werden muss. Falls nicht alle Fragmente empfangen werden, muss das gesamte IPv6-Paket erneut übertragen werden, diesmal jedoch nur über den letzten Hop. [Vgl. 9, S. 1209-1210]

Da in einem mesh-under System alle Pakete direkt weitergeleitet werden, kommt es sehr schnell zu einem hohen Datenaufkommen. Bei route-over Netzen hingegen muss jeder Knoten über die Ressourcen verfügen, um das gesamte Paket abzuspeichern. Fragmentierung bringt immer einen zusätzlichen Mehraufwand an Rechenleistung und zu übertragenden Daten mit sich, was nicht zuletzt zu einer verkürzten Laufzeit von batteriebetriebenen Geräten führt. Deshalb ist der effektivste Ansatz, die zu übertragenden Daten zu minimieren, um eine Fragmentierung zu umgehen.

3.5 (De-) Komprimierung

Ein 802.15.4 Frame hat eine maximale physische Größe von 127 Byte, wobei höchstens 25 Byte Verwaltungsdaten anfallen. Des Weiteren werden, für die zur Nachrichtenübertragung notwendigen Header, 40 Byte (IPv6), 8 Byte (UDP) und, falls eine AES-CCM-128-Verschlüsselung auf dem Link Layer gewünscht ist, weitere 21 Byte benötigt, womit nur noch 33 Byte für die eigentliche Nachricht bleiben. [Vgl. 8, S. 5]

Aufgrund einiger Probleme mit den in RFC 4944 beschriebenen Kompressionsverfahren LOWPAN_HC1 und LOWPAN_HC2, wie z.B. der lediglich für Link Local Kommunikation optimierten Komprimierung oder auch, dass in HC2 ausschließlich die Kompression von UDP, TCP oder ICMP vorgesehen ist, soll im Folgenden das in RFC 6282 vorgestellte, verbesserte Verfahren genauer beschrieben werden. [Vgl. 10, S. 440] LOWPAN_IPHC verwendet hierbei zwei Methoden. Zum einen wird, wie bei dem Vorgänger HC1, genutzt, dass viele Felder meistens den gleichen Wert besitzen, z.B. das Versionsfeld ist 6, oder aus einer anderen Schicht abgeleitet werden können und dies somit lediglich in einem Encoding Byte angedeutet werden muss. [Vgl. 7, S. 7-8] Außerdem wird ein shared context verwendet, um globale und multicast Adressen zu komprimieren. Hierfür wird ein Context Identifier Extension (CID) Byte eingeführt, das optional nach dem Encoding Byte folgt. [Vgl. 10, S. 440] Des Weiteren wird eine Komprimierung des Hop Limits erlaubt, falls es einem der Werte 1, 64 oder 255 entspricht. [Vgl. 11, S. 7] Außerdem ist nun eine seperate Komprimierung von Traffic Class und Flow Label möglich, sodass nicht mehr beide „0“ sein müssen. [Vgl. 7, S. 9] Das Encoding Byte von IPHC nutzt die letzten 5 Bit des Type Bytes, um zusätzliche Informationen unterzubringen und Platz zu sparen. [Vgl. 11, S. 5] Beide zusammen sehen wie folgt aus:

Abb. 8: LOWPAN_IPHC Encoding [Vgl. 11, S. 6]

Die ersten drei Bit deuten das Folgen des LOWPAN_IPHC encodings an. TF zeigt an, ob die Traffic Class und/oder das Flow Label null sind. Falls der nächste Header mit dem LOWPAN_NHC Verfahren komprimiert wurde, wird dies in dem NH Bit zum Ausdruck gebracht, während HLIM signalisiert, ob das Hop Limit auf einen häufig genutzten Wert gesetzt und komprimiert wurde oder unkomprimiert in-line übertragen wird. Ein Folgen des oben genannten CID Bytes wird durch CID angegeben. Die Art der Kompression der Source- bzw. Destination-Adress, also stateless oder context basierend, wird durch SAC bzw DAC gekennzeichnet. Multicast Adressen werden durch M-Bit identifiziert. Durch SAM, das von SAC bzw. DAM, welches von DAC und M abhängig ist, wird definiert, wie viele Bits der Source bzw Destination Adresse in-line übertragen werden. [Vgl. 7, S. 9-10]

Das bereits erwähnte LOWPAN_NHC Kompressionsverfahren, das ebenfalls in RFC 6282 beschrieben wird, folgt unmittelbar auf die in-line übertragenen IP Felder und kann genutzt werden, um beliebige weitere Header, wie IPv6 Extension oder UDP Header, zu komprimieren. Hierfür wird ein Next Header Identifier Feld genutzt, das eine variable Länge hat. Die Kompression eines UDP Headers ist ähnlich der des in RFC 4944 beschriebenen HC2-Verfahrens. Falls einer (oder beide) der zu übertragenden Ports in den Bereich 61616 - 61631 fällt, werden die ersten 12 Bits gelöscht und die verbleibenden vier Bit in-line übertragen. Die Länge kann aus einer tieferen Schicht abgeleitet werden. [Vgl. 8, S. 19-20] Außerdem ist das Annullieren der Prüfsumme möglich, falls eine höher gelegene Schicht

einen gleichwertigen Integritäts-Check der Nachricht durchführt. Dies war in HC2 nicht erlaubt und ist üblicherweise der Fall, wenn auf dem Transport oder Application Layer Sicherheitsmechanismen verwendet werden. Ein UDP Header kann mit diesem Verfahren im Idealfall bis auf 2 Bytes komprimiert werden. [Vgl. 7, S. 10-11]

In Abbildung 9 sind mit IPHC komprimierte Header verschiedener Kommunikationsarten zu sehen. Bei der Kompression des UDP Headers wird davon ausgegangen, dass sich die Ports in dem zuvor definierten Bereich befinden.

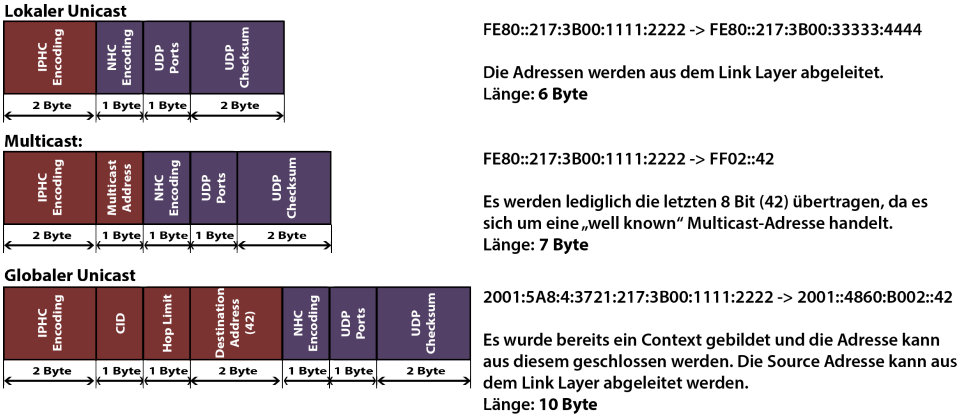


Abb. 9: Mit IPHC komprimierte Header [Vgl. 7, S. 11]

4 Fazit

Aufgrund der Beschränkungen die IPv4 mit sich bringt, ist ein neuer Standard nahezu unausweichlich. Auch wenn die Adaption von IPv6 nun schon mehr als zwei Jahrzehnte andauert, ist dieses Protokoll höchstwahrscheinlich das zukünftige Verfahren um handelsübliche PCs zu vernetzen. Das Ziel bei der Ausarbeitung des IPv6 Protokolls war es jedoch nicht Ressourcen, wie Rechenleistung, Speicherplatz oder Energie, zu sparen, sondern die Übertragung von vielen Daten zu optimieren. Dadurch eignet sich IPv6 nicht für den Einsatz in Embedded Systems. Mit 6LoWPAN wurde vom IETF ein offener Standard geschaffen, der es ermöglicht große Netzwerke, bestehend aus leistungsschwachen Systemen aufzubauen. Auch wenn hierfür bereits proprietäre Protokolle existieren, fehlt diesen die Möglichkeit direkt über IPv6 zu kommunizieren, was bei einer Einbindung in das Internet zu einem Mehraufwand in den Gateway Routern führt. Es bleibt abzuwarten, ob sich 6LoWPAN gegenüber seinen Konkurrenten durchsetzen wird. Eine ähnlich lange Adaptionsphase, wie bei dem Umstieg von IPv4 zu IPv6, ist hingegen unwahrscheinlich, da Embedded Systems eine definitiv neue Art von Protokollen benötigen.

Literatur

[1] Benedikt Abendroth, Aaron Kleiner und Paul Nicholas. *Cybersecurity Policy for the Internet of Things*. White paper. Microsoft Corporation, 2017. URL: http://mscorpmedia.azureedge.net/mscorpmedia/2017/05/IoT_WhitePaper_5_15_17.pdf.

[2] Monika Ermert. *ICANN legt sich für rasche Migration zu IPv6 ins Zeug*. https://www.heise.de/newsticker/meldung/ICANN-legt-sich-fuer-rasche-Migration-zu-IPv6-ins-Zeug-145959.html. [Online; accessed 01-June-2017]. 2007.

[3] R. Hinden u. a. *IP Version 6 Addressing Architecture*. RFC 4291. Network Working Group, Feb. 2006. URL: https://tools.ietf.org/html/rfc4291.

[4] R. Hinden u. a. *IPv6 Global Unicast Address Format*. RFC 3587. Network Working Group, Aug. 2003. URL: https://tools.ietf.org/html/rfc3587.

[5] S. Deering u. a. *Internet Protocol, Version 6 (IPv6)*. RFC 2460. Network Working Group, Dez. 1998. URL: https://tools.ietf.org/html/rfc2460.

[6] S. Thomson u. a. *IPv6 Stateless Address Autoconfiguration*. RFC 2462. Network Working Group, Dez. 1998. URL: https://tools.ietf.org/html/rfc2462.

[7] PhD. J. Hui u. a. *6LoWPAN: Incorporating IEEE 802.15.4 into the IP architecture*. White paper 3. Internet Protocol for Smart Objects (IPSO) Alliance, Jan. 2009. URL: http://www.ipso-alliance.org/wp-content/media/6lowpan.pdf.

[8] G. Montenegro u. a. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944. Network Working Group, Sep. 2007. URL: https://tools.ietf.org/html/rfc4944.

[9] A. H. Chowdhury u. a. „Route-over vs Mesh-under Routing in 6LoWPAN“. In: *IWCMC '09 Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*. Jan. 2009, S. 1208–1212. URL: https://www.researchgate.net/publication/220762351_Route-over_vs_mesh-under_routing_in_6LoWPAN.

[10] Sreejesh V. K. und G. Santhosh Kumar. „6LoWPAN: Invorporating IEEE 802.15.4 into the IP architecture“. In: *India Conference (INDICON), 2012 Annual IEEE*. IEEE, Dez. 2012, S. 439–443. URL: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=023B0022D423FF370BD879434C8C68F1?doi=10.1.1.902.4837&rep=rep1&type=pdf.

[11] Ed. J. Hui u. a. *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. RFC 6282. Network Working Group, Sep. 2011. URL: https://tools.ietf.org/html/rfc6282.

[12] Shelby, Zach und Carsten Bormann. *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011.

IP-Multicast in Wireless Sensor Networks - Energy Saver or Management Horror?

Frank Lu[1], Richard Schaeffer[2]

Abstract: A wireless sensor network (WSN) is a network consisting of battery-operated sensors used to measure conditions in the surrounding areas. Often several hundreds or even thousands of nodes are necessary in such a network in order to create a stable system. The problem with conventional unicast communication between the nodes is the high consumption of resources. The idea of multicast is to spread information to a group of devices by using efficient protocols and creating special multicast groups. The sensor nodes can join these groups and be part of the multicasting network. This paper focuses on the question, if multicast routing is a reliable option for communication in WSNs. Its benefits and disadvantages are evaluated in relation to energy consumption and management issues. Multicast routing protocols such as GMR, HGMR, HRPM and CNSMR can optimize the performance of multicast in WSNs.

Keywords: Multicast; Wireless Sensor Network; WSN; efficiency; energy consumption; management; GMR; HRPM; HGMR; CNSMR

1 Introduction

The amount of transmitted data in networks has increased significantly over the past few years. Big Data Analytics has become more and more popular. The data is being processed from every aspect our life, e.g. climate conditions or social media like Facebook. [BOJC16] The collection process of the data itself poses a challenge, which cannot be taken lightly, since transmissions in a network can be quite slow and the amount of data is massive. [Si08] The following sections will focus specifically on the area of wireless sensor networks. Communication and efficient use of resources are especially important in these sensor networks due to the limited resources. The three fundamental types for communication are unicast, broadcast and multicast. Each has its own advantages and disadvantages in different use case.[Lo17] Multicast routing optimizes the communication and aims to reduce the use of bandwidth. The complexity of the protocol architecture and the poor compatibility with existing services are trade-offs that must be considered when implementing multicast. [Si08]
This paper gives an overview on the efficiency of multicast in wireless sensor networks

[1] Technical University Munich, Informatics, Arcisstraße 21, 80333, Munich, Germanyfrank.lu@tum.de
[2] LMU Munich, Media Informatics, Geschwister-Scholl-Platz 1, 80539, Munich, Germany richard.schaeffer@campus.lmu.de

and its problems. In section 2 some background knowledge on wireless sensor networks and multicast is explained. Section 3 introduces the advantages and disadvantages of using multicast in wireless sensor networks. The mentioned aspects from section 3 are discussed in section 4. Finally, section 5 will conclude the topic by summing up the discussion and suggest an answer to the question whether multicast should be used in wireless sensor networks or not.

2 Background

2.1 Wireless Sensor Networks

A wireless sensor network (WSN) is a network consisting of battery-operated sensors used to measure conditions in the surrounding areas. The costs of a wireless sensor network are low compared to a standard data network. Often several hundreds or even thousands of nodes are necessary in order to create a stable system. The cost of each node must be kept at a minimum, since so many nodes are being deployed. The nodes of a WSN are therefore severely limited in their processor's capability, storage, communication and power. [Si08]

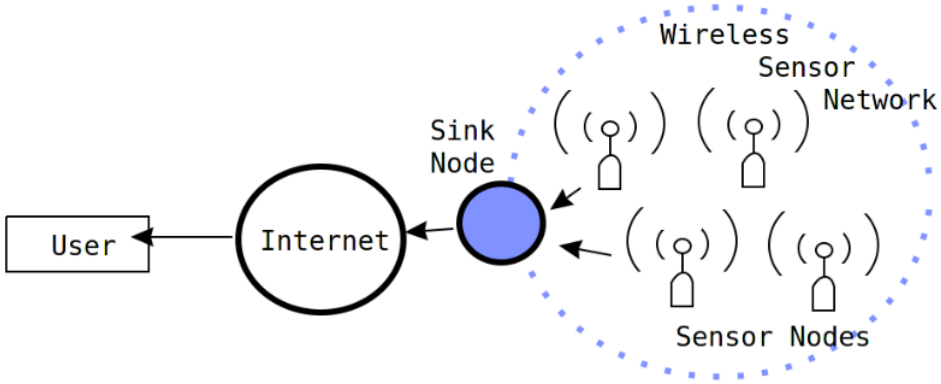


Fig. 1: Structure of a wireless sensor network

The sensors nodes periodically communicate with their neighbors via wireless links. The communication usually takes place through multiple hops. Messages are sent through neighbors to reach other nodes that are located out of their wireless range. In figure 1 the structure of a WSN is depicted. All the measured data from the sensors needs to be collected and processed. A sink node is commonly used as a base station to accomplish this task. In order to collect the data, the sink node has to send out request messages to the respective sensor nodes. It needs larger capabilities than other sensor nodes because it has to request and process a high amount of information. Sink Nodes are normally connected to the Internet, so that they can transmit the collected information. They function as the interface between the WSN and the "outside world". [Si08]

2.2 Use Cases of Wireless Sensor Networks

Wireless Sensor Networks can be used in different areas of application. For instance, they are used to observing environmental and physical phenomenons such as temperature, pressure and humidity. Overall, there are five different types of Wireless Sensor Networks: ground, underground, aquatic, multi-media and mobile WSNs. [KB16]
In a ground WSN up to thousands of nodes can be placed in an existing area. A solid and stable communication to the base station is essential, wherefore it is also important to maintain all energy resources. To support this issue, power sources such as battery or solar cells can be added to the sensor nodes. [KB16]
Unlike ground WSNs, that are formed terrestrial, underground WSNs are build inside the crust of earth, in caves, mines or other underground areas. Nodes belonging to this type of network are more expensive, because they have to insure reliable communication even through underground contents like rocks, soils and water. This fact has a direct impact on issues of deployment, equipment and maintenance. It needs to be taken into consideration when deploying a system since the sensor nodes only have batteries with limited power and recharging or replacing them can be difficult. [KB16]
Aquatic WSNs are implemented in underwater areas through transmission of acoustic waves. Due to their position underground, autonomous aquatic vehicles are used to receive the data from the sensor nodes. [KB16]
Unlike Aquatic WSNs, Multi-media WSNs own low cost sensor nodes. These are equipped with microphones and cameras used to collect data. Using a wireless connection, these nodes can also share this information. Mobile WSNs are even able to move during their interaction with the environment. Therefore, this type of WSN is used in industrial or military applications. [KB16]
Otherwise WSNs can also be deployed in defense systems (e.g. surveillance, investigation, targeting systems), forest (e.g. monitoring of animals and environment) and medical science (e.g. diagnosing patients) applications, besides of industrial applications.
Since the use cases of WSNs are very diverse, it becomes more of a challenge to develop a system that fits the needs of different scenarios. Due to all the mentioned limitations of wireless sensor networks, it has become necessary to optimize the communication inside of the network. Multicast can be vital in these networks. [KB16]

2.3 Unicast, Broadcast and Multicast

The goal of multicast is to spread information to a group of devices. A problem with conventional unicast solution is the high consumption of resources. The different communication types are shown in figure 2. In unicast every device on the network needs to establish a unique session. So whenever a sender wants to transmit a message to a group, one session for each device in the group must be established. The same message must be replicated and sent in each session. This can consume a high amount of bandwidth depending on the number of clients in the group. [Si08]

The number of sessions can be minimized when broadcast is used. This is a solution to the bandwidth problem. Since the stream is broadcast to every device in the network, only one session is needed instead of multiple sessions. So each device receiving the message will broadcast it to its neighbors. When broadcasting is used, every device in the network will receive the message, even when the device is not interested in it. [Si08]

IP Multicast solves this problem. Messages can be transmitted one-to-many, many-to-many and many-to-one. Members of the network can send *join* and *leave messages* to a multicast group. A multicast group has a group IP address. Every message sent to that address will be forwarded to all the members of the group. A sender only needs to send one message to the group address. The message will then be replicated and forwarded to the respective group members. The replication is done by network nodes or Layer 3 devices like routers. Multicasting protocols make use of the fact that some links from the source to the destinations can be shared. Optimally there will only be one message sent on each link. Multicast messages require the use of the Internet Group Management Protocol (IGMP). It allows the nodes to create, join and leave multicast groups. IGMP is specific for IPv4. In IPv6 the Multicast Listener Discovery was implemented with similar functionalities. [Si08]

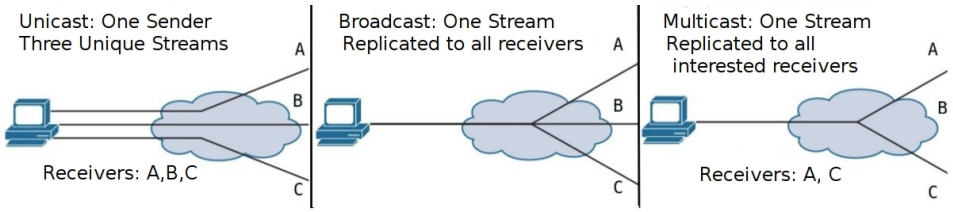


Fig. 2: Examples of Unicast (left), Broadcast (middle) and Multicast (right) [Lo17]

3 Multicast in Wireless Sensor Networks

3.1 Energy Consumption: Unicast and Multicast in WSNs

The previous section has made it clear, that in theory conventional unicast solutions take up more bandwidth than multicast. As a result the energy consumption of each node should be higher since more messages need to be sent for communication.

The study [Si08] evaluates the battery consumption in real WSN and compares unicast to multicast solutions. The test was performed using three nodes, as seen in figure 3. Two of three nodes were subscribed to the multicast group. For the network structure the star topology was used with the same signal strength between each node and the sink node. So three sensor nodes were set up with one sink node in the middle to collect their data. For the comparison of unicast to multicast three different scenarios were tested. The battery value of each sensor was requested once every minute, regardless of the scenario. Request messages were always sent from sink node to sensor node to collect the data. In the first

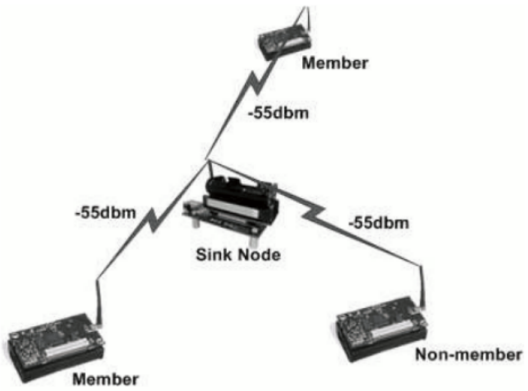


Fig. 3: Network Topology of study [Si08]

scenario the measured temperature value of the multicast sensor nodes was additionally requested every second. The request was sent to the two multicast members using unicast messages. The second scenario requested the temperature from the two multicast members at each second as well. This time multicast messages were used. In the third scenario only the battery value was requested every minute from each device. The number of messages sent each hour can be seen in figure 4. [Si08]

	Requests		Responses	
	Battery	Temp	Battery	Temp
Unicast	60	7200	180	7200
Multicast	60	3600	180	7200
Idle	60	0	180	0

Fig. 4: Messages sent in three different scenarios [Si08]

The results from the study in figure 4 show that the amount of temperature requests sent in the unicast case is 7200. In the unicast scenario the temperature was requested from the two unicast device every second. So two unicast messages were sent from the sink node to the two sensor nodes every second. Hence, the amount of messages calculated from figure 4 in an hour is:

$$2 \times 60\text{min} \times 60 \frac{\text{request msg}}{\text{min}} = 7200\text{request msg}$$

Meanwhile, in the multicast scenario the number of multicast members does not increase the number of request messages. Only one multicast message needs to be transmitted to the group instead of addressing every single group member. So the amount of request messages for the temperature is only half the amount of the unicast request messages. [Si08]

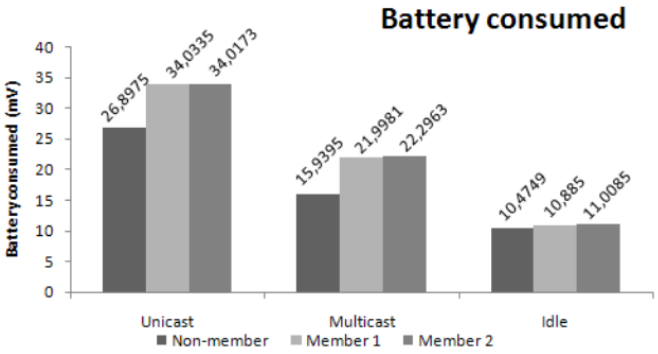


Fig. 5: Battery consumption in three different scenarios [Si08]

From the evaluation of the energy consumption in figure 5 it has become clear that the battery consumption was higher when using unicast instead of multicast. The reason for the increase is due to the higher amount of temperature requests sent. In fact, the study [Si08] concludes that multicast member nodes spent around 10.5 to 11 mV on the temperature requests. Unicast however used up around 23 mV on this task, resulting in an increase of about 12 mV. In the described scenarios the unicast solution is energetically more expensive than the multicast solution. It needs to be said, that the study only covered the use in a very simple network. These results can not necessarily be generalized for all networks.

3.2 Multicast Management in WSNs

IP-Multicast has only slowly developed in its commercial fields of application in the Internet. Due to the complexity of its protocol architecture, integrating multicast into existing services becomes difficult. Deployment, installation and management – especially at customers premises – is an important issue where the infrastructure of the network has to be considered. For instance, it can occur that firewalls do not recognize the multicast address and will refuse to accept it as a consequence. Furthermore, if the hardware in the network is too old, the component needs to be updated before it can be used in multicasting. But an update or a replacement of hardware is only profitable, if the modification is cheaper than the maintenance of the current system. [Di00]
Apart from this, the management of multicast groups is another issue. Group creations as well as sender and receiver messages need to be authorized and verified. The IP address allocation of multicast groups must be handled consistently throughout the system. Since the nodes of WSNs only communicate with each other through their WLAN inteface, security and protection mechanisms against attacks on multicast routes are of great significance. Otherwise, WSN would be highly vulnerable and the nodes could drown with alternate data.

The tools for network management are not developed far enough to handle these issues. Hence, when constructing the multicast architecture, these problems must be taken into consideration. [Di00]
Intradomain multicasting is easier to deploy and install, whereas interdomain multicast between multiple networks is more complex because it precludes control over the whole network. Finding and debugging errors in interdomain multicast becomes more and more difficult with increasing size of networks. Overall, it's not a simple issue to generalize and commercialize multicast. Unicast is cheaper to deploy than multicast. Hence, its usage is only profitable when the cost savings of bandwidth are higher than the deployment, installation and management. Thus, it is important to invent protocols, which make use of the benefits of multicast and optimizes its performance. [Di00]

3.3 Multicast Routing Protocols

As seen in the previous sections, multicast has some energetic advantages over unicast. This advantage also comes with heavy management issues. Complex protocols are required to handle these. In this section several multicast routing protocols will be briefly introduced together with their benefits and challenges.

3.3.1 Geographics Multicast Routing Protocol (GMR)

Geographic multicast routing is a fully localized routing algorithm. "Localized" means, that the full structure of the network is not relevant for each node. The purpose of the algorithm is to transmit multicast messages to multiple destinations. In GMR the nodes do not need to flood the network to figure out a suitable path. [Sa07]
Before transmitting messages the nodes will define a number of neighbors as relay nodes. Depending on the destination, the node will choose a different appropriate relay neighbor. It is necessary in GMR that all the nodes know their positions. The positions of the nodes are periodically exchanged with neighbors in radio range. GMR selects the relay node with the lowest cost at each routing step. When a sensor node transmits one multicast message, all the neighbor nodes in radio range are likely to receive the message. This phenomenon is known as the ***wireless multicast advantage*** because it reduces the overhead of sending two messages to two different neighboring relay nodes. Because of this, a GMR header must be attached to the data message so all the neighbor nodes receiving the message can realize whether they have been selected as relay nodes or not. The GMR header contains a list of all the selected relay nodes and the destinations that they are responsible for. [Sa07]
The next hop is determined at each single node based on the current network topology. This is the main advantage of GMR. It allows the algorithm to adapt to topological changes. Especially in WSNs where sensor nodes are less capable and could crash, adaptability is very valuable. The greedy neighbor selection algorithm used in GMR works fine with small

to medium amount of nodes. The problem of this algorithm becomes clear when looking at its worst case complexity taken from [Sa07]:

$$O(Dn \min(D,n)^3) \qquad D := \#\text{destinations}, n := \#\text{neighbor}$$

The run time grows exponentially once the amount of destinations and neighbor nodes increase increases. This might not be a problem with lower amount of sensor nodes, but in a more complex network with thousands of nodes, implementing GMR in that way is just not feasible. [Sa07]

3.3.2 Hierarchical Rendezvous Point Multicast (HRPM)

Hierarchical Rendezvous Point Multicast separates the network into multicast groups, which are divided into subgroups in the next step. Each of these subgroups has an access point, which can communicate to the same rendezvous point. The subgroups are managed by their access points, whereas the access points are managed by the rendezvous point. HRPM minimizes the maintenance cost of access points and rendezvous points using mobile geographic hashing. Furthermore, it also minimizes overhead by forming its hierarchy with the subgroups. Thus, the two main concepts of the protocol are the usage of hierarchy and supporting its efficiency with geographic hashing. [VG16] To send packets from a source location, a multicast group identifier has to be hashed to get the information on the location of the rendezvous point. Afterwards a list of available access points can be requested from the rendezvous point, so that the source can form overlay trees to the destinations of the access points. In the next step the data is sent by using geographic forwarding. The access points can also form overlay trees to the nodes of their subgroup and send data also by using geographic forwarding. Therefore, the access points and the source are using unicast for transmitting data over the branches of the subtrees. [Ko10]

3.3.3 Hierarchical Geographics Multicast Routing (HGMR)

Hierarchical Geographic Multicast Routing combines GMR and HRPM. It uses the same hierarchy to minimize encoding overhead as HRPM, whereas the subgroups benefit from the concept of GMR such as the high forward efficiency. HGRM forms an overlay tree from the source to the access point and an additional tree from the access point to the nodes of its subgroup. With the unicast based forwarding strategy of HRPM the data packets can be transmitted from the source to the access points. This data packets can be forward afterwards with a local multicast scheme along the multicast tree to its destination. [VG16] When a node wants to join a multicast group, it has to transmit a *JOIN* message to the rendezvous point. In the next step the node gets a value of decomposition as an answer from the rendezvous point. With this value and its position the node can use a hash function to

receive the location of the access point of its subgroup. After sending an *UPDATE* message to this access point the join process is finished. In HGMR packets are transmitted from the source to the access points by using the unicast-based forwarding strategy of HRPM, whereas each subgroup makes usage of the algorithm of GMR. Thus, HGMR benefits from both concepts. [Ko10]

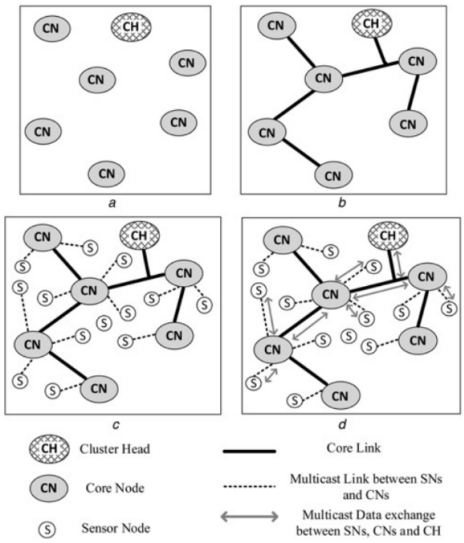


Fig. 6: Formation of CNSMR [Ma15]

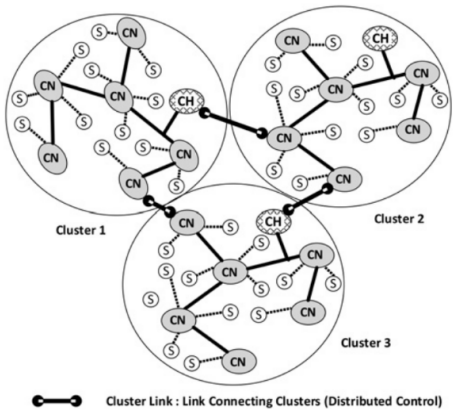


Fig. 7: CNSMR protocol across the clusters in sensor networks [Ma15]

3.3.4 Core Network Supported Multicast Routing (CNSMR)

Core Network Supported Multicast Routing takes heterogeneous nodes into consideration. CNSMR is able to differentiate between several types of nodes. The sensor networks consist of one cluster head node, several core nodes and regular sensor nodes. This set-up is shown in figure 6. Cluster head and core nodes are selected nodes with higher computing power, storage and energy resources than the regular sensor nodes. The CNSMR uses the core nodes power for its protocol. Cluster head nodes maintain the route path with core nodes and other cluster head nodes. Meanwhile, the core nodes are connected to their neighboring sensor nodes. [Ma15]
The CNSMR protocol utilizes a data gathering technique to forward data from the sensor nodes to the cluster head nodes. In conventional multicast routing, the source node needs to start the multicast session by inviting the other target nodes to the group. CNSMR avoids this by using the data gathering techniques to reduce the complexity of routing. For target nodes that are located in the same cluster as shown in figure 7, the DC-GMR approach is used, similar to the above mentioned GMR. Cluster links connect two clusters to enable

inter-cluster connectivity.
The figure 6 shows the formation of a CNSMR network. During step (a) in figure 6 powerful nodes are selected to function as core nodes and cluster head nodes. The selected core nodes together with one cluster head node form the core network of the cluster. In step (b) core nodes and the head node communicate with each other to see, if their communication is stable and whether more core nodes are necessary or not. [Ma15] The sensor nodes are selected and linked to the core nodes in step (c) to form the sensor network supported by the inner core network. Step (d) shows the multicast routing paths between the sensor, core and cluster head nodes.
In case of link failures, core nodes reduce the routing complexity by storing the routing state information. The advantage of CNSMR is the use of alternate multicast routes with less traffic. As a result, the balance of the traffic load is improved. [Ma15]

4 Discussion

The mentioned protocols above have shown weaknesses and strengths of multicast protocols. An important issue that CNSMR has managed to fix are alternating paths. It was proven in [Sa07], that GMR managed to minimize the number of total transmissions with its multicasting protocol and therefore reduced the amount of energy consumption, but only by traversing the same fixed multicast path over and over. For every delivery to the same multicast group, the same path is selected in GMR. In general the energy consumption will decrease, but the nodes that are selected for the multicast paths are consuming even more than in unicast protocols. By using alternate paths in CNSMR the traffic load is more balanced and the durability of the network increased.

Hierarchical Rendezvous Point Multicast is a scalable protocol, which minimizes the encoding overhead, an improvement compared to GMR. Its performance is not dependent on the size of the network or the density of the nodes. Nevertheless, it is energy inefficient, because packages are sent via unicast to different neighbor nodes, wherefore the demand of bandwidth increases. Hierarchical Geographic Multicast Routing is more efficient. It combines the multicast advantages of GMR with the improved scalability due to the concept of HRPM. Thus, the delay time of HGMR is smaller compared to GMR and HRPM. [VG16]

The main problem with all protocols was scalability, especially for GMR. For larger amount of nodes the time complexity substantially increases, making deployment of GMR almost impossible. The same applies for the other protocols. CNSMR is significantly improved in this regard by clustering the network into smaller parts, but this comes at the cost of storage. Alternate paths are stored in the core nodes so that packages can be routed faster. An increase in the number of nodes would also increase the necessary storage. Scaling up such would require an immense increase of the required memory. It appears that the issues of IP multicast in wireless sensor networks cannot be easily fixed without creating another.

5 Conclusion

IP-Multicast, as a method of communication between nodes, can be a suitable choice for wireless sensor networks. Multicasting messages is without a doubt more efficient than conventional unicast because the amount of messages is reduced. The nodes will therefore waste less energy on sending out and filtering messages. A downside of this is the increase of deployment and management costs. Determining the next neighbor node for a specific multicast path, can also consume a lot of computing power. Depending on the scenario and the size of the network, every protocol has its own benefits and drawbacks. IP-Multicast can perform efficiently once it is set up, but the effort of managing and adjusting the system must be taken into consideration. In the future multicast must improve in scalability and, more importantly, in usability. Multicast will not be deployed unless the benefits significantly outshine the management effort. Automatic installation and management in multicast has yet to be developed.

References

[BOJC16] Bello-Orgaz, Gema; Jung, Jason J; Camacho, David: Social big data: Recent achievements and new challenges. Information Fusion, 28:45–59, 2016.

[Di00] Diot, Christophe; Levine, Brian Neil; Lyles, Bryan; Kassem, Hassan; Balensiefen, Doug: Deployment issues for the IP multicast service and architecture. IEEE network, 14(1):78–88, 2000.

[KB16] Keswani, Kapil; Bhaskar, Anand: Wireless Sensor Networks: A Survey. Futuristic Trends in Engineering, Science, Humanities, and Technology FTESHT-16, p. 1, 2016.

[Ko10] Koutsonikolas, Dimitrios; Das, Saumitra M; Hu, Y Charlie; Stojmenovic, Ivan: Hierarchical geographic multicast routing for wireless sensor networks. Wireless networks, 16(2):449–466, 2010.

[Lo17] Loveless, John: IP multicast, Volume I: Cisco IP multicast networking. 2017.

[Ma15] Maddali, Bala Krishna: Core network supported multicast routing protocol for wireless sensor networks. IET Wireless Sensor Systems, 5(4):175–182, 2015.

[Sa07] Sanchez, Juan A; Ruiz, Pedro M; Liu, Jennifer; Stojmenovic, Ivan: Bandwidth-efficient geographic multicast routing protocol for wireless sensor networks. IEEE Sensors Journal, 7(5):627–636, 2007.

[Si08] Silva, Ricardo; Silva, Jorge Sa; Simek, Milan; Boavida, Fernando: Why should multicast be used in WSNs. In: Wireless Communication Systems. 2008. ISWCS'08. IEEE International Symposium on. IEEE, pp. 598–602, 2008.

[VG16] Verma, Kanchan; Gupta, Mansi: A Comparitive Study On Location based Multicast Routing Protocols of WSN: HGMR, HRPM, GMR. Global Journal of Computer Science and Technology, 15(8), 2016.

Protokolle der Transportschicht in eingebetteten Kommunikationssystemen

Jennifer Lauterbach[1]

Abstract: Diese Arbeit soll einen Überblick über verschiedene Protokolle der Transportschicht bezüglich ihrer Eignung für eingebetteten Systemen geben. Dabei werden sowohl die allgemeinen Anforderungen von eingebetteten Systemen betrachtet, als auch deren Ansprüche an Transportprotokolle. Weiterhin wird zum einen auf die bekanntesten Protokolle, UDP und TCP, und zum anderen auf die neueren Protokolle QUIC, SCTP, RTMPF und MPTPC eingegangen. Dazu gibt es jeweils einen Überblick und eine kritische Beurteilung, ob sie sich für eingebettete Systeme eignen.

Keywords: embedded systems; tcp; udp; quic; sctp; rtmfp; mptcp; transport layer protocols

1 Einleitung und Motivation

Aus der heutigen Zeit und Gesellschaft sind Geräte mit eingebetteten Systeme nicht mehr wegzudenken. Sie sind in den verschiedensten Lebenslagen zu finden, sei es nun in Transportmitteln, in Gebrauchsgegenständen wie Kaffeemaschinen und Beleuchtung oder in medizinischen Systemen. Diese Geräte senden und empfangen Daten, sind oft untereinander vernetzt und kommunizieren automatisch miteinander.

Der Begriff eingebettete Systeme wird nach Marwedel wie folgt definiert: „Eingebettete Systeme sind informationsverarbeitende Systeme, die in ein größeres Produkt integriert sind und die normalerweise nicht direkt vom Benutzer wahrgenommen werden." [Ma07, S. 1].

Dabei ist zu beachten, dass diese Systeme spezielle Anforderungen haben, die sich von handelsüblichen PCs unterscheiden, sich auch auf den Datenaustausch und somit die Transportprotokolle beziehen. Und mit dieser Fragestellung beschäftigt sich diese Ausarbeitung. Dabei werden die speziellen Ansprüche im nächsten Kapitel behandelt, gefolgt von der genauen Betrachtung der einzelnen Protokolle hinsichtlich der Anforderungen und Eignung für eingebettete Systeme.

[1] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland j.lauterbach@campus.lmu.de

2 Anforderungen an eingebettete Systeme und deren Transportprotokolle

2.1 Allgemeine Anforderungen an eingebettete Systeme

Eingebettete Systeme haben allgemein andere Anforderungen als handelsübliche PCs, die in diesem Kapitel dargelegt werden.

Dabei sind Verlässlichkeit, Effizienz und Echtzeit besonders relevante Punkte.

Verlässlichkeit ist wichtig, damit das System in sicherheitskritischen Momenten nicht ausfällt oder den Zeitraum zwischen Wartungsterminen ohne Probleme übersteht. Dazu gehört auch, dass das System korrekt arbeitet und keine Fehler oder Probleme verursacht. Auch Daten sollten nur für befugte Gruppen zur Verfügung stehen, damit Unberechtigte diese nicht einsehen oder verändern können [Ma07, S. 1].

Effizienz ist gerade für eingebettete Systeme besonders relevant. Zum einen ist die Energie-Effizienz wichtig, da eingebettete Systeme oft in batteriebetriebenen Geräten enthalten sind, die eine bestimme Zeit oder möglichst lange mit ihrer Energie auskommen müssen. Auch der Speicher, den so ein System hat, ist begrenzt und klein, also sollte der auf dem Gerät gespeicherte Programm-Code möglichst kurz sein und die Systemressourcen, die bei der Ausführung des Codes verwendet werden, möglichst sparsam eingesetzt werden [Ma07, S. 1 f.].

Viele eingebettete Systeme müssen außerdem in Echtzeit funktionieren, also muss die Funktionalität des Systems in einem bestimmten Zeitraum ausgeführt werden. Dabei ist auch wieder die Verlässlichkeit wichtig, denn wenn Fehler passieren, kann der Prozess ggf. nicht wiederholt werden, da dafür keine Zeit mehr vorhanden ist [Ma07, S. 4].

Weiterhin sollten die Systeme möglichst klein und leicht sein, sowie aus robusten Teilen bestehen, da portable Systeme zu unhandlich sind, wenn sie zu groß oder zu schwer sind. Die physische Robustheit ist auch gerade bei Systemen, die draußen oder unter starken Witterungsbedingungen zum Einsatz kommen, besonders wichtig, auch um die Verlässlichkeit und Verfügbarkeit zu wahren. Weiterhin sind auch geringe Kosten der einzelnen Teile für die Produzenten wichtig [LBS15, S. 120 f.].

2.2 Notwendigkeit für Transportprotokolle in eingebetteten Systemen

Nun stellt sich die interessante Frage, ob Transportprotokolle zur Datenübertragung in eingebetteten Systemen benötigt werden oder ob die Daten auch direkt ohne diese übertragen werden können. Das kommt natürlich ganz auf das eingebettete System und dessen Anwendungszweck an, aber in vielen Fällen werden Transportprotokolle für die Übertragung benötigt.

Die Transportprotokolle sind für die Übertragung von Daten, für die Kommunikation und ihren Aufbau zwischen den Kommunikationspartnern zuständig. Sie bieten außerdem verschiedene nützliche Services an. Ein Feuermelder z. B. sendet ein Alarmsignal an die Zentrale. Dabei ist es wichtig, dass er eine Empfangsbestätigung bekommt, um ansonsten das Signal erneut zu senden. In einem anderen Beispiel werden medizinische Messergebnisse übertragen, wobei es wichtig ist, dass diese aus Datenschutzgründen verschlüsselt werden. Das sind nur zwei Beispiele an nützlichen Services, die ein Transportprotokoll anbieten kann. Das heißt also, wenn eingebettete Systeme Daten übertragen wollen und bestimmte Ansprüche an die Übertragung haben, brauchen sie Transportprotokolle.

2.3 Anforderungen an Transportprotokolle in eingebetteten Systemen

Was sind also die Anforderungen an Transportprotokolle, wenn sie speziell in eingebetteten Systemen verwendet werden? Wenn man die generellen Anforderungen an eingebettete Systeme aus Kapitel 2.1 betrachtet, kann man diese teilweise auch auf Transportprotokolle übertragen.

Effizienz spielt auch bei den Transportprotokollen eine wichtige Rolle. So sollte auch die Übertragung der Daten möglichst ressourcenschonend vonstattengehen, das heißt, einfache Protokolle werden bevorzugt.

Verlässlichkeit ist auch sehr wichtig. Die Daten sollen in vielen Fällen verlustfrei und unverändert übertragen werden, unabhängig davon, ob die Veränderung mutwillig oder durch einen Übertragungsfehler geschieht. Je nach Anwendungsfall kann auch Verschlüsselung eine wichtige Anforderung sein, um unberechtigtes Lesen der Daten bei der Übertragung zu verhindern. Diese Punkte sind gerade für sicherheitskritische Systeme besonders wichtig.

Bei Echtzeitsystemen muss auch die Übertragung in einem gegebenen Zeitfenster stattfinden, also sollte hier die benötigte Zeit bei der Wahl des Protokolls berücksichtigt werden.

Die restliche Anforderungen, wie das geringe Gewicht, die Größe und Robustheit der Teile spielen für die Transportprotokolle keine Rolle. Der Preis nur in der Hinsicht, dass möglichst auf proprietäre Protokolle verzichtet werden sollte oder ein gut unterstütztes und wenig fehleranfälliges gewählt werden sollte, um die Entwicklungskosten zu reduzieren.

3 Transportprotokolle in eingebetteten Systemen

Nun folgt ein Überblick über einige Transportprotokolle und jeweils die kritische Betrachtung ihrer Eignung für eingebettete Systeme hinsichtlich der oben genannten Anforderungen.

3.1 TCP in eingebetteten Systemen

3.1.1 Überblick über TCP

TCP steht für Transmission Control Protocol und ist ein verbindungsorientiertes Ende-zu-Ende Übertragungsprotokoll, das Vollduplexkommunikation beherrscht. Es soll vor allem übertragungssicher und zuverlässig sein. Die Daten werden dabei als Byte-Stream übertragen. Die Zuverlässigkeit wird dadurch gewährleistet, dass verlorene, beschädigte oder doppelte Pakete erneut gesendet werden. Die einzelnen Pakete haben Sequenznummer, sodass Pakete in falscher Reihenfolge in die richtige gebracht werden können. Das Protokoll beherrscht außerdem eine Flusskontrolle, mit der der Empfänger die Datenrate des Senders beschränken kann. TCP erlaubt auch Multiplexing, sodass verschiedene TCP-Kommunikationen zeitgleich stattfinden können [Po81, S. 1 und 3 ff.].

Es gibt verschiedene Algorithmen für die Überlastkontrolle. Mit denen kann der Sender beschränken, wie viele Daten ins Netz gesendet werden, den Verlust von Daten entdecken und reparieren, sowie die erneute Übertragung von Daten überwachen [APB09, S. 3 - 9].

Verbindungen werden in TCP durch einen Three-Way-Handshake hergestellt. Dessen Schwäche ist die Anfälligkeit für SYN-Flooding Attacken. Dabei werden sehr viele SYN-Anfragen an den Server geschickt, aber die Verbindungen dann nicht vom Client bestätigt. Das kann zwar mit SYN-Cookies vermieden werden, diese sind allerdings nicht immer einsetzbar [Ta11, S. 560 ff.].

TCP bietet keine Möglichkeit, den Absender zu verifizieren oder die Daten zu verschlüsseln. Dafür muss noch ein anderes zusätzliches Protokoll oder Anwendung verwendet werden.

3.1.2 Betrachtung von TCP hinsichtlich der Anforderungen

TCP hat durch seinen großen Header und den Verbindungsaufbau einen hohen Verwaltungsaufwand, was für einfache und kleine eingebettete Systeme ungeeignet ist. Also eignet sich TCP auch für einfache und schnelle Übertragungen nicht, sondern nur, wenn größere Datenmengen verschickt werden und Fehlerfreiheit und Übertragungssicherheit dabei wichtig sind. Zusätzlich ist TCP weit verbreitet, sodass es keine Unterstützungsprobleme gibt.

TCP ist also gerade für Systeme, bei denen Übertragungssicherheit und Verlässlichkeit an erster Stelle stehen, gut geeignet. Dabei sollte beachtet werden, dass es keine Möglichkeit zur Verschlüsselung und Authentifizierung bietet. Dagegen gibt es für Systeme, bei denen sichere Übertragung keine große Rolle spielt, effizientere Protokolle. Durch die weitläufige Unterstützung von TCP kann das Protokoll leicht verwendet werden und da es schon lange im Einsatz ist, sind Schwächen und Angriffsmöglichkeiten mittlerweile gut bekannt.

3.2 UDP in eingebetteten Systemen

3.2.1 Überblick über UDP

UDP steht für User Data Protocol und ist ein kleines, verbindungsloses Transportprotokoll. Es ist transaktionsorientiert, bietet keine garantierte Übertragung der Pakete und hat auch keine Möglichkeit, Duplikate zu verhindern. Dass die Pakete in der richtigen Reihenfolge ankommen, wird nicht garantiert. Es bietet außerdem keine Möglichkeiten zur Fluss- oder Staukontrolle. Der UDP-Header ist nur 32 Bit lang und kann leicht gefälscht werden. Genau wie TCP hat es keine Möglichkeit zur Verschlüsselung der Daten oder der Verifizierung des Absenders [Po80, S. 1 ff.].

3.2.2 Betrachtung von UDP hinsichtlich der Anforderungen

Wird UDP nun hinsichtlich der Anforderungen betrachtet, ist die schnelle Übertragung für viele Anwendungen ein großer Vorteil. Da es verbindungslos ist und keine Verschlüsselung oder andere Sicherheitsvorkehrungen getroffen werden, entsteht kein Verwaltungsaufwand bei der Übertragung und das macht UDP sehr effizient.

Zum Beispiel für Echtzeitanwendung, bei denen viele große Mediendaten übertragen werden, wie Streaming, Online Multiplayer Games oder Voice-over-IP, ist es gut geeignet, da der Verlust von einigen Paketen weniger schlimm ist als Verzögerungen in der Übertragung.

Die fehlenden Möglichkeiten zur Fehlerausbesserung oder erneutem Verschicken von verlorenen Paketen können aber für bestimmte Anwendungen ein Problem darstellen und UDP damit für diese ungeeignet machen. Falls Verschlüsselung und Authentifizierung genutzt werden sollen, muss ein zusätzliches Protokoll dafür verwenden werden, das wiederum die Effizienz von UDP reduzieren würde.

Dafür ist UDP ein weit verbreitetes Protokoll und kann somit leicht mit verschiedener Soft- und Hardware verwendet werden.

3.3 Vergleich von TCP und UDP

Nun werden TCP und UDP verglichen, um herauszufinden, welches der beiden Protokolle für welchen Zweck besser geeignet ist. Der größte Unterschied zwischen den beiden ist, dass TCP verbindungsorientiert ist, während UDP verbindungslos ist. Bei TCP wird großer Wert auf eine übertragungssichere und zuverlässige Übertragung gelegt, während es bei UDP keine Möglichkeit gibt, eine solche Übertragung der Daten zu garantieren.

Dafür können die Daten durch den fehlenden Verbindungsaufbau bei UDP ohne Verzug übertragen werden, wohingegen die Übertragung bei TCP durch die Sicherheitsvorkehrungen und den aufwendigen Verbindungsaufbau langsam ist.

Man muss sich also speziell für seine Anwendung entscheiden, wie wichtig die schnelle bzw. verlässliche und sichere Datenübertragung ist.

3.4 QUIC in eingebetteten Systemen

3.4.1 Überblick über QUIC

QUIC steht für Quick UDP-based Internet Connection. Obwohl es auf UDP basiert, ist QUIC dennoch ein sicheres Protokoll, das seine Header und die Payload der Daten mit TLS verschlüsselt. Die Daten werden in einem bidirektionalen Byte-Stream übertragen, wobei mehrere Streams parallel laufen können [IT17, S. 3 und 6].

Alle Frames innerhalb der Pakete haben dabei eine einzigartige Sequenznummer, sodass sie geordnet und Duplikate leicht erkannt werden können. Falls ein Paket verloren geht, werden die entsprechenden Teile des Frames in einem extra Paket erneut gesendet [IS17, S. 3 f.].

Das Protokoll beherrscht außerdem zwei Möglichkeiten zur Flusskontrolle. Zum einen kann der Empfänger ein Maximum für die pro Stream empfangbare Datenrate angeben und das Limit dann nach und nach erhöhen. Zum anderen kann der Empfänger ein Maximum für den Zwischenspeicher angeben, der für alle Streams gemeinsam benutzt wird [IT17, S. 7].

3.4.2 Betrachtung von QUIC hinsichtlich der Anforderungen

Viele der Fähigkeiten von QUIC sprechen für eine gute Eignung für eingebettete Systeme. Es ist schneller als TCP ohne einen hohen Verwaltungsaufwand und bietet im Vergleich zu UDP trotzdem Übertragungssicherheit. Weiterhin hat es im Gegensatz zu den beiden Protokollen Möglichkeiten zu Authentifizierung und Verschlüsselung.

Da das Protokoll noch experimentell und in Überarbeitung ist, wird es noch nicht sehr weitläufig unterstützt und ist kaum praxiserprobt.

3.5 SCTP in eingebetteten Systemen

3.5.1 Überblick über SCTP

SCTP steht für Stream Control Transmission Protocol und ist ein verbindunsorientiertes und zuverlässiges Transportprotokoll. Es funktioniert teilweise so ähnlich wie UDP, denn

es sendet seine Daten als Nachrichtenblöcke und nicht als Byte-Stream wie TCP. Es ist außerdem fähig, einen Multistream aufzubauen, kann also mehrere unabhängige Nachrichten parallel versenden. Seine Nachrichten sind jeweils mit einer Sequenznummer versehen, sodass die Pakete geordnet werden können, aber nicht müssen, falls das nicht benötigt wird. Außerdem unterstützt es Multihoming, sodass die Nachrichten mehrere Pfade als Ziel verwenden können, falls einer ausfällt. Gegen Denial-of-Service Attacken wie SYN-Flodding ist es durch einen Four-Way-Handshake abgesichert [St07, S. 1, 5, 12 und 126].

Falls ein NAT auf dem Übertragungsweg SCTP nicht unterstützt, kann das Protokoll auch über UDP getunnelt werden [TS13, S. 1] oder eine TCP API verwenden [Bi05, S. 1]. Es wird mittlerweile von Linux gut unterstützt, kann aber unter Windows bis jetzt nur als Third-Party-Produkt verwendet werden [Ho12].

3.5.2 Betrachtung von SCTP hinsichtlich der Anforderungen

SCTP ist gerade für Systeme gut geeignet, die zwar eine sichere Übertragung wie bei TCP benötigen, aber nur eine sichere Übertragung ohne Ordnung der Pakete oder nur geordnete Pakete brauchen, ohne dass verlorene Pakete erneut gesendet werden. So wird viel Verwaltungsaufwand vermieden, wenn diese Funktionen nicht benötigt werden. Auch müssen die Daten nicht als Stream gesendet werden, sodass man mehr Freiheiten hat.

SCTP ist durch Multihoming ausfallsicherer als TCP. Wenn dem System mehrere Verbindungsmöglichkeiten zur Verfügung stehen, z. B. WLAN und ein Mobilfunknetz, kann immer zu der Verbindung mit den besten Übertragungseigenschaften gewechselt werden. Durch Multistream können die Systeme effizienter arbeiten und die Möglichkeit zur Verhinderung von SYN-Flodding Attacken ist auch vorteilhaft für eingebetteten Systeme.

Ein Problem von SCTP ist die fehlende NAT Unterstützung. Weiterhin ist es nicht so weitläufig verbreitet wie TCP und UDP und wird oft noch nicht unterstützt. Auch ist es noch nicht praxiserprobt und weitere Schwächen und Fehler werden sich daher erst in Zukunft zeigen.

3.6 RTMFP in eingebetteten Systemen

3.6.1 Überblick über RTMFP

RTMFP steht für Adobes Secure Real-Time Media Flow Protocol und ist für den verschlüsselten oder unverschlüsselten Transport von Multimedia-Daten gedacht und zwar sowohl für Client-Server als auch Peer-to-Peer Architekturen. Es ist vor allem bei Echtzeit-Datenübertragungen sehr effektiv, wie Übertragung von Audio- und Videodaten sowie für Echtzeitvideospiele. Daten können priorisiert übertragen werden, sodass die wichtigen

Daten wie z. B. Audio vor Chat-Nachrichten übertragen werden. Multistream ist auch möglich. Es nutzt UDP als Übertragungsbasis und hat deswegen eine geringe Latenz und wenig Verwaltungsaufwand und ist deswegen gut für Multimedia-Übertragungen geeignet. Dabei können die Daten nach Wunsch nach Empfang geordnet werden. Es ist insbesondere für die direkte Kommunikation von Flash-Player zu Flash-Player entwickelt worden [Th13, S. 4 ff.].

Außerdem hat es Möglichkeiten zur Fluss- und Staukontrolle [Th13, S. 67 und 89] und ist durch einen Four-Way-Handshake vor SYN-Flooding Attacken geschützt [Th13, S. 53].

3.6.2 Betrachtung von RTMFP hinsichtlich der Anforderungen

Adobes RTMFP wurde zwar mittlerweile mit dem RFC7016 zugänglich gemacht, war aber ursprünglich proprietär. Neben der Kommunikation zwischen Flash Playern ist es auch für Anwendungen gedacht, die mit Adobes AIR Framework gebaut wurden. Auch der Adobe Media Server unterstützt das Protokoll [Th13, S. 1 und 5]. Wenn man also mit Flash-Playern oder dem Framework arbeitet und Multimedia-Daten verschlüsselt und sicher übertragen will, kann man RTMFP als Transportprotokoll in Betracht ziehen. Allerdings wird auf den wenigsten eingebetteten Systemen einen Flash-Player verwendet werden, da diese entweder nicht die nötigen Ressourcen für diesen haben oder es keinen Anwendungszweck gibt.

3.7 MPTCP in eingebetteten Systemen

3.7.1 Überblick über MPTCP

MPTCP steht für Multipath TCP und ist eine Erweiterung für TCP. Im Gegensatz zu normalem TCP kann es nicht nur einen Pfad pro Verbindung nutzen, sondern mehrere gleichzeitig. Es läuft auf einem oder mehreren TCP-Subflows. Es bietet dieselben Funktionen wie TCP an, wie übertragungssichere und zuverlässige Übertragung. Falls einer Anwendung MPTCP nicht bekannt ist, verhält sich MPTCP wie normales TCP. Die Verbindung zwischen den Hosts läuft dabei wie bei TCP ab, es gibt nur eine extra Option, um festzustellen, ob der andere Hosts MPTCP beherrscht, um dann zusätzliche Subflows herzustellen. Falls dem nicht so ist, wird eine normale TCP-Verbindung aufgebaut. Auch zu einer bestehenden MPTCP-Verbindung können zusätzliche Subflows hinzugefügt werden [Fo13, S. 4 und 7].

Falls ein Paket verloren geht, könnte es zwar über irgendeinen der Subflows erneut gesendet werden, aber um die Integrität zu wahren, muss es zusätzlich auch auf dem ursprünglichen Subflow nochmal gesendet werden [Fo13, S. 32 f.].

Bei der Staukontrolle hat jeder Subflow sein eigenes Congestion Window, die aber zusammen betrachtet werden müssen, um Bottlenecks zu vermeiden und eine faire Verteilung der

Ressourcen zu gewährleisten. Es gibt dafür einen Algorithmus, der zwar nicht optimal ist, aber gut funktioniert. Es sind bereits weitere Algorithmen in Planung [Fo13, S. 33 f.].

Da der Verwaltungsaufwand bei einer MPTCP-Verbindung noch größer als bei einer normalen TCP-Verbindung ist, lohnt sich MPTCP bei sehr kurzen Übertragungen nicht. Es gibt aber momentan noch keine Heuristiken, ab wann sich MPTCP rentiert, da das Protokoll noch recht neu ist [Fo13, S. 46 f.].

Zur Zeit ist MPTCP anfällig für Flooding- und verschiedene Hijacking-Attacken. Dabei ist dies im Gegensatz zu TCP auch Angreifern von außerhalb möglich. MPTCP sollte auch mit NAT kompatibel sein, um großräumiger unterstützt zu werden. Allerdings werden bei NATing die Adressen verändert und dies sieht für das MPTCP-Protokoll aus wie ein Man-in-the-middle-Angriff. Es muss also eine andere Möglichkeit gefunden werden, die Integrität der Adressen zu wahren [Ba11, S. 8 - 14].

3.7.2 Betrachtung von MPTCP hinsichtlich der Anforderungen

MPTCP ist für mobile Geräte mit mehreren Empfangsmöglichkeiten wie WLAN und mobile Daten geeignet, sodass immer der stabilere oder schnellere Kanal genutzt werden kann. Das kann auch gerade für Geräte interessant sein, die eine stabile Verbindung benötigen, wenn sie wichtige Daten übertragen und die Verbindung dabei nicht abbrechen darf.

Für Geräte allerdings, die gar keine sichere Verbindung wie TCP benötigen oder denen die Übertragung zu langsam ist, gibt es bessere Alternativen. Der Verwaltungsaufwand von MPTCP ist höher als bei TCP, für nur kurze Übertragungen ist es daher nicht geeignet, es lohnt sich erst, wenn viele Daten übertragen werden. Dann sollten jedoch mehrere Pfade genutzt werden, damit sich der Verwaltungsaufwand lohnt.

Durch die Kompatibilität mit TCP können viele Anwendungen leicht auf MPTCP umgerüstet werden, ohne das größere Umstellungen nötig sind. Und da TCP ein sehr gut unterstützter Standard ist, hat man mit MPTCP keine großen Probleme bei der Übertragung. Allerdings ist MPTCP anfälliger für Angriffe wie Flooding- und Hijacking-Attacken als normales TCP und damit momentan noch unsicherer. Außerdem ist MPTCP zur Zeit ein experimentelles Protokoll, dessen zusätzlichen Vor- und Nachteile sich erst im weiteren Verlauf zeigen werden und das sich in der Praxis noch beweisen muss.

4 Zusammenfassung und Fazit

In dieser Arbeit wurden verschiedene Transportprotokolle hinsichtlich ihrer Eignung für eingebettete Systeme kritisch betrachtet. Dabei waren die wichtigsten Anforderungen Effizienz, Verlässlichkeit und Echtzeit. Betrachtet wurden die beiden bekanntesten Protokolle,

TCP und UDP, sowie die neuere Protokolle QUIC, SCTP, RTMFP und MPTCP. Einen Überblick über die verschiedenen Services und Eigenschaften dieser gibt Tabelle 1.

Service	**TCP**	**UDP**	**QUIC**	**SCTP**	**RTMFP**	**MPTCP**
Multihoming	nein	nein	nein	ja	nein	ja
Multistream	nein	nein	ja	ja	ja	nein
geordnete Datenpakete	ja	nein	ja	ja	ja	ja
ungeordnete Datenpakete	nein	ja	nein	ja	ja	nein
Fluss-/Staukontrolle	ja	nein	ja	ja	ja	ja
verbindungsorientiert	ja	nein	ja	ja	ja	ja
Schutz vor SYN-Flodding	nein	-	ja	ja	ja	nein
Verschlüsselung	nein	nein	ja	nein	ja	nein
Verwaltungsaufwand	hoch	gering	gering	hoch	gering	hoch

Tab. 1: Vergleich der Services der Protokolle

TCP ist durch den sicheren Verbindungsaufbau nicht sehr schnell und durch den hohen Verwaltungsaufwand auch nicht sehr effizient, dafür aber sehr zuverlässig und übertragungssicher.

UDP dagegen überträgt die Daten schnell und effizient, hat aber keine Möglichkeiten diese gegen Verlust oder Beschädigung abzusichern. Es ist für Anwendungen, für die Übertragungssicherheit wichtig ist, ungeeignet. Beide Protokolle bieten keine Möglichkeit zur Verschlüsselung ohne eine zusätzliche Anwendung an.

QUIC ist ein zuverlässiges Protokoll, dass auch eine schnelle und effiziente Übertragung bietet. Es verschlüsselt außerdem als einziges neben RTMFP seine Daten. Allerdings ist es noch neu und im Aufbau, sodass viele Schwächen noch unbekannt sind.

SCTP ist durch Multistream und -homing etwas effizienter und ausfallsicherer als TCP. Es kann dieselben Sicherheitsfunktionen wie TCP anbieten, diese müssen aber nicht genutzt werden, sodass bei Bedarf Verwaltungsaufwand vermieden wird. Allerdings ist es nicht so weit verbreitet wie TCP.

RTMPF ist nur für spezielle Übertragungen von Multimediadaten geeignet, wenn das Adobe AIR Framework genutzt wird oder Flashplayer genutzt werden, was aber allein aus Ressourcenknappheit bei den wenigsten eingebetteten Systemen der Fall ist.

MPTCP ist etwas anfälliger für Angriffe als TCP und hat sogar noch mehr Verwaltungsaufwand als dieses, bietet aber die Möglichkeit Daten über mehrere Pfade gleichzeitig zu übertragen, was für eine großen Datenmenge oder Übertragung über WLAN und mobiles Netz gleichzeitig relevant sein kann.

Wenn man sich die Protokolle ansieht, gibt es bei jedem Vor- und Nachteile für eingebettete Systeme, sodass man keine pauschale Aussage treffen kann. Je nach Einsatzzweck haben die Anforderungen für die eingebetteten Systeme unterschiedliche Gewichtungen, da für

den einen die Sicherheit an erster Stelle steht und für den anderen die schnelle und effiziente Übertragung. Man muss sich also für seinen Zweck das beste Protokoll aussuchen. Dabei sollte aber auch die Unterstützung des Protokolls beachtet werden, da gerade die neuen Protokolle viele Verbesserungen mit sich bringen, aber noch wenig erprobt sind.

Literaturverzeichnis

[APB09] Allman, M.; Paxson, V.; Blanton, E.: TCP Congestion Control. RFC 5681, RFC Editor, September 2009. http://www.rfc-editor.org/rfc/rfc5681.txt.

[Ba11] Bagnulo, M.: Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6181, RFC Editor, March 2011. https://www.rfc-editor.org/rfc/rfc6181.txt.

[Bi05] Bickhart, Ryan W: Transparent TCP-to-SCTP translation shim layer. Bericht, DELAWARE UNIV NEWARK DEPT OF COMPUTER AND INFORMATION SCIENCES, 2005.

[Fo13] Ford, A.; Raiciu, C.; Handley, M.; Bonaventure, O.: TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, RFC Editor, January 2013. http://www.rfc-editor.org/rfc/rfc6824.txt.

[Ho12] Hogg, Scott: , What About Stream Control Transmission Protocol (SCTP)? Webseite, 2012. http://www.networkworld.com/article/2222277/cisco-subnet/what-about-stream-control-transmission-protocol--sctp--.html, Stand: 27.6.2017.

[IS17] Iyengar, Janardhan; Swett, Ian: QUIC Loss Detection and Congestion Control. Internet-Draft draft-ietf-quic-recovery-03, IETF Secretariat, May 2017. http://www.ietf.org/internet-drafts/draft-ietf-quic-recovery-03.txt.

[IT17] Iyengar, Janardhan; Thomson, Martin: QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-03, IETF Secretariat, May 2017. http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-03.txt.

[LBS15] Lange, Walter; Bogdan, Martin; Schweizer, Thomas: Eingebettete Systeme: Entwurf, Modellierung und Synthese. De Gruyter Oldenbourg, 2015.

[Ma07] Marwedel, Peter: Eingebettete Systeme. Springer-Verlag Berlin Heidelberg, 2007.

[Po80] Postel, J.: User Datagram Protocol. STD 6, RFC Editor, August 1980. http://www.rfc-editor.org/rfc/rfc768.txt.

[Po81] Postel, Jon: Transmission Control Protocol. STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[St07] Stewart, R.: Stream Control Transmission Protocol. RFC 4960, RFC Editor, September 2007. http://www.rfc-editor.org/rfc/rfc4960.txt.

[Ta11] Tanenbaum, Andrew S.: Computer Networks, 5th Edition. Pearson, 2011.

[Th13] Thornburgh, M.: Adobe's Secure Real-Time Media Flow Protocol. RFC 7016, RFC Editor, November 2013. http://www.rfc-editor.org/rfc/rfc7016.txt.

[TS13] Tuexen, M.; Stewart, R.: UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication. RFC 6951, RFC Editor, May 2013. http://www.rfc-editor.org/rfc/rfc6951.txt.

Challenges of Application Layer Protocols in Embedded Communication Systems

Valentina Visintini[1]

Abstract: In many areas of our daily life we are surrounded by objects on the network. Besides common devices such as personal computers and mobile phones, embedded systems in the form of everyday objects are getting more and more common. As these vastly heterogeneous things are being incorporated into the network environment, a unified Internet architecture has become necessary, specifying protocols that fit both machine-to-machine communication and embedded devices using Internet connectivity. This paper gives an overview of the main application protocols suitable for the IoT – CoAP, MQTT and XMPP – and of their challenges compared to traditional Internet application protocols, and illustrates their features and purposes by a few examples.

Keywords: Internet of Things (IoT); Embedded Systems; Application Layer Protocols; CoAP; MQTT; XMPP

1 Introduction

In many areas of our daily life we are surrounded by objects on the network. Besides self-explanatory devices such as personal computers and mobile phones that are specifically used to access the internet, embedded systems in the form of everyday objects – be it cars and traffic lights building a smart traffic infrastructure for sustainable transportation, remotely controllable heaters and window blinds in smart homes, or fitness tracking devices in an augmented healthcare system that allow for analyzing a patient's physical condition based on the recorded data – are getting more and more common.

These "things" have been changing the network environment over the past years, forming the term *Internet of Things* (IoT), which encapsulates the communication between such devices in a local network, but also integrating them globally into the classic Internet infrastructure.

As a consequence, it is necessary to provide a common architecture for embedded devices – both those communicating directly and those depending on Internet connectivity – and to review established Internet protocols regarding their applicability to the IoT.

Since resources such as memory, computing power and energy are usually limited in such devices, and the diversity of things keeps growing, new challenges are issued to protocol requirements.

[1] LMU Munich, Institute of Informatics, Oettingenstr. 67, 80538 Munich, Germany v.visintini@campus.lmu.de

So far, there has not been a common agreement on which standard to use for handling, processing and storing sensor data in the IoT. This paper gives an overview of the main application protocols suitable for the IoT and their challenges compared to the classic TCP/IP protocols, and illustrates their features and purposes by a few examples.

The rest of the paper is organized as follows: Section 2 provides a short summary of the requirements for application protocols and explains possible communication patterns. In section 3, the main protocols CoAP, MQTT and XMPP are introduced and compared based on IoT-critical features. Possible implementations are explained in section 4. Section 5 provides an overview of related work and, finally, Section 6 concludes.

2 Traditional Application Layer Protocols

The following section sums up the most crucial requirements towards application protocols for resource-constrained devices and provides an introduction to typical communication patterns.

2.1 Properties

In a classic TCP/IP context, the application layer deals with representation and provides standard Internet services to applications requiring network communication such as web browsers or email clients. These applications request connections to remote hosts which are passed to the transport layer to send and receive the user data [TW10].

Protocols at this level have provided the means for cross-platform operations: the most common one is the HyperText Transfer Protocol (HTTP), a stateless protocol that transfers text messages from a web server to a client. Running on top of the Transmission Control Protocol (TCP), it enables processes to communicate across the boundaries of diverse networks [TW10].

Further examples of traditional application protocols are the Simple Mail Transfer Protocol (SMTP) that handles email exchange or the File Transfer Protocol (FTP) that allows for file sharing between a client and a server.

An IoT device communicates with other embedded devices through a local network, but it may also need access to the Internet to transfer sensor generated data to a computing facility, a data center, or a cloud [Ka15]. The connection can be established via a gateway or by the device itself, if it has an integrated Ethernet controller, as many microcontrollers now have. Nonetheless, there are several criteria that need to be considered when choosing a suitable protocol for managing and configuring embedded systems. They are discussed in section 3.1.

2.2 Architecture

There are two main communication patterns that are explained in detail below.

Request/Response Request/Response is the a widely used architecture model based on the client-server interaction paradigm, where a client sends a request to a server and the server replies to it, as shown in Figure 1. This can be realized both as a connection-oriented or as a connectionless service [TW10].

Using TCP, in order to send a request to the server, the client must first connect to it. Once the connection is established and the request is sent, the server starts processing it, while the client waits. As soon as it is ready, the server sends a response over the same connection.

Typically, it is implemented in a synchronous way, meaning the connection is kept open until a response is sent back or the timeout period expires, allowing for a communication without delays. However, a response can also be returned asynchronously at a later point, as it is done in CoAP [LS16], see section 3.2.

In contrast, connectionless protocols such as UDP are more lightweight since requests can be sent without establishing the connection first. This way, less messages are sent over the network, reducing overhead significantly.

An example for a connectionless application might be the Voice over Internet Protocol (VoIP) used for voice messaging and multimedia sessions over the Internet.

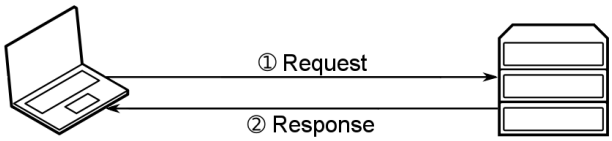


Fig. 1: Request/Response: a client sends a request to a server and the server sends a response back to the client.

Publish/Subscribe A different approach is the Publish/Subscribe pattern, often abbreviated as pub/sub. It enables distributed, asynchronous communication between message publishers and message subscribers.

A publisher sends the sensor data it has produced to the middleware, which forwards the messages to all consumers that have subscribed to the specific event notifications, so every

subscriber can monitor the sensor data related to the topics they have marked interesting [HW16].

This way subscribers get notified about changes in relevant sensor data, however, they cannot directly communicate with the publisher. Every participant only interacts with the broker, as illustrated in Figure 2. Thus, subscribers cannot control actuators.

What makes this communication pattern interesting for IoT applications is the fact that it decouples publishers and subscribers thanks to asynchronous, non-blocking messaging: devices need not be connected at the same time to be able to receive notifications. Furthermore, they are loosely coupled in space, as they do not have to know the addresses of other participants [HW16].

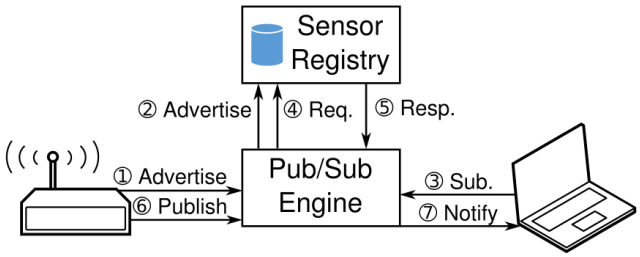


Fig. 2: Publish/Subscribe: sensors advertise themselves to a registry which is queried by the pub/sub engine to which all potential clients subscribe instead of directly searching for sensors. Subscribers are notified everytime a relevant sensor stream publishes some information [HW16].

3 IoT-specific Protocols

This section sums up the most crucial requirements towards application protocols for resource-constrained devices, introduces the most important ones (CoAP, MQTT and XMPP) and describes why they meet the conditions.

3.1 Requirements

As many devices act together on the same network, they compete for resources such as bandwidth, which might be additionally constrained in a wireless network, leading to a low operational duty cycle and unreliable communication. Flooding a network with requests would thus be counterproductive, but not only due to bandwidth restrictions: many devices are battery operated and have therefore limited power supplies. The more overhead communication produces, the shorter the lifetime of a device [Ka15].

Another important point is reliability. Services that never lose any data are considered reliable. This can be achieved by acknowledging the receipt of each message so a sender is sure that it arrived, which is fundamental in an application such as file transfer.

On the other hand, sending additional acknowledging messages introduces overhead and delays, which are particularly undesired in the context of real-time applications and, as stated above, shorten the lifespan of the devices concerned.

The ability to connect, manage and control a device from anywhere at any time and the consequential amount of data that needs to be handled, processed and stored calls for appropriate authentication and authorization to maintain privacy and integrity [Ka15].

In the next chapter, the three principal protocol standards are presented and analyzed with regard to the requirements addressed in this unit.

3.2 Protocols

CoAP The Constrained Application Protocol (CoAP) [SHB14] is a stateless application layer protocol similar to HTTP, designed by the Internet Engineering Task Force (IETF) for constrained device networks.

It is based on the request/response model, but is typically implemented in such a way that participants act both as a client and a server to provide machine-to-machine (M2M) communication.

Similar to HTTP, a CoAP client can access a resource on the server by querying the respective Uniform Resource Identifier (URI), using one of these four methods from the HTTP command compendium: `GET`, `POST`, `PUT` and `DELETE` [LB16]. Being a subset of HTTP it can be easily integrated in the existing web.

Unlike in HTTP, CoAP messages are exchanged asynchronously over UDP, making it a lightweight protocol with minimal bandwidth requirements and reduced overhead compared to TCP. It also supports both unicast and multicast [Ka15].

To provide reliability, four types of messages are defined determining the Quality of Service (QoS) level:

1. *Confirmable* (`CON`): A request message that requires an acknowledgement. If no packet is lost, each `CON` message returns exactly one response of type `ACK` or type `RST`, synchronously or asynchronously. Otherwise it is retransmitted within the default timeout until the recipient sends an `ACK` with the same Message ID.
2. *Non-Confirmable* (`NON`): A message that does not require an acknowledgement, such as readings from a sensor that are repeated regularly. Duplicates and message loss are possible.

3. *Acknowledgement* (`ACK`): A message that confirms the reception of a `CON` message, without any information of success or failure of encapsulated requests.
4. *Reset* (`RST`): A message that confirms that a `CON` message was received but could not be processed properly due to missing context.

The message type is specified by two bits in the header of each packet which is four-byte fixed-length binary and enables small messages through its compact encoding of options.

CoAP comes without any built-in security features, but since HTTP is protected by the Transport Layer Security (TLS) over TCP, it stands to reason to use the Datagram Transport Layer Security (DTLS) running on top of UDP analogously. It is considered a complete security protocol being able to perform authentication, key exchange and protecting application data [KKT14].

MQTT The Message Queue Telemetry Transport (MQTT) by IBM is a lightweight, asynchronous publish/subscribe protocol running on top of TCP. It was designed to provide one-to-many message distribution between devices, reduced network traffic and decoupling of applications, making it suitable for environments with low bandwidth or expensive networks.

Messages in MQTT are published on topics. The protocol is based on the publish/subscribe paradigm and has many clients connect to a middleware (the broker) and publish messages to topics on the one hand and subscribe to topics that they are interested in on the other hand.

Clients can choose between subscribing to an explicit topic, in which case only messages to that topic will be received, or making a subscription that include wildcards covering multiple topics.

As opposed to a message queue a topic is very lightweight as the clients need not create a topic before publishing or subscribing to it. They are accepted by the broker without prior initialization.

Reliability can be classified by three types of messages determining the Quality of Service (QoS) level (see [Lo10]):

1. *At most once*: A message that is sent once and does not require an acknowledgement. Duplicates and message loss are possible. Typically used for messages such as sensor data readings where losing one is not critical as the next one is published shortly afterwards.
2. *Delivered at least once*: A message that requires an acknowledgement and is sent at least once, so reception is ensured but duplicates may occur.

3. *Delivered exactly once*: A four-way handshake mechanism ensures exactly one delivery of a message, when duplicates or message loss are critical, e.g. with billing systems.

Although MQTT runs over TCP, its two-byte fixed-length header reduces transport overhead significantly which means that only little network bandwidth is needed, leading to less data transfer and therefore lower battery consumption. These properties make it a convenient protocol for smartphone applications such as the Facebook Messenger, that is based on this protocol [Zh11].

To ensure security, the protocol itself provides a client identifier and username/password credentials, which are handled by TLS/SSL (Secure Sockets Layer) and can also be used to authenticate devices on the application level. Authorization must be implemented in the broker. Relying on TLS/SSL, MQTT has the same security standards as HTTP.

XMPP The eXtensible Messaging and Presence Protocol (XMPP) is an open XML technology for real-time communication, standardized by the IETF and mainly used for instant messaging and collaboration.

As its name implies, it is designed to be extensible and adapt to changes. However, general features of XMPP are the message transport over TCP and an architecture that provides both asynchronous publish/subscribe and synchronous request/response communication, be it client-to-server or server-to-server communication.

XMPP uses messages in the form of the eXtensible Markup Language (XML) which need additional computational resources to be parsed, leading to shorter lifetimes of devices, and which include tags that produce increased overhead.

In order for servers to know if an entity is present (online, offline or busy), it can advertise its network availability to other systems. This is not indispensable for communication with others but it simplifies real-time interaction because the communication can be started when the intended recipient is available, avoiding unnecessary requests that increase the network load.

Regarding reliability the IETF makes the following point in RFC 6120 [SA11]:

> The use of long-lived TCP connections in XMPP implies that the sending of XML stanzas over XML streams can be unreliable, since the parties to a long-lived TCP connection might not discover a connectivity disruption in a timely manner. At the XMPP application layer, long connectivity disruptions can result in undelivered stanzas. Although the core XMPP technology defined in this specification does not contain features to overcome this lack of reliability, there exist XMPP extensions for doing so [...].

Security is guaranteed by the built-in TLS/SSL, so message streams can be protected against tampering and eavesdropping.

3.3 Comparison

As we can see, even though most embedded devices have very similar if not the same resource constraints, i. e. shared network bandwidth, shorter lifetime or connectivity interruptions due to battery supply and limited computational resources, different standardized protocols take different approaches regarding communication paradigms, transport protocols, Quality of Service levels and security mechanisms. An overview is given in Table 1.

Protocol	Transport	QoS Options	Architecture	Security
CoAP	UDP	YES	Request/Response	DTLS
MQTT	TCP	YES	Publish/Subscribe	TLS/SSL
XMPP	TCP	NO	Request/Response Publish/Subscribe	TLS/SSL

Tab. 1: Differences between application layer protocols [Ka15]

Although communication over TCP seems more common and provides more reliability, there are still protocols that run on top of UDP and ensure reliability through different QoS levels. The only one mentioned in this paper is CoAP, the most lightweight one, which allows for communication via request/response pattern and which, being similar to the widely used HTTP, translates easily to it and can be integrated very efficiently into Internet applications. On the other hand, it has no built-in security features, making it necessary to introduce an additional protocol.

If device lifetime is more important than integrability, we have found that the protocol of choice should keep overhead caused by data transfer very low consuming less battery, such as MQTT. Furthermore, the publish/subscribe pattern is useful when many updates of the same value are required.

Finally, the TCP-based instant messaging protocol XMPP is convenient for real-time applications thanks to its presenter feature and can be easily extended, but the fact that it defines no QoS levels makes it impractical for M2M communication. Additionally, it requires parsing of XML messages, whose tags produce overhead, consuming additional computational resources which shortens the lifetimes of devices.

4 Application Areas

To give the reader an idea of areas where and how the examined protocols are applied, two examples have been chosen to illustrate common use cases: a smoke detector in a smart home and a sensor implementation to monitor a patient's physical levels in real time.

Smart Home Bringing these theoretical paradigms to the application fields of the IoT, a typical (smart) home device immediately suggests itself: the smoke detector. In order to know when there is a fire in a house it would hardly make sense to implement it the request/response way and continually have to ask for it. Not querying often enough could make you miss an actual alarm, while querying too often might lead to useless, duplicate information.

By subscribing to the topic instead, the relevant information is only sent and received when data is published and the overhead caused by useless queries is avoided. This could be elegantly achieved by using the MQTT protocol.

The smoke detector sends a message containing a topic (e.g. home_automation/smart_guys _home/kitchen/smoke_alarm) and the sensor data to the broker. Every MQTT client interested in that specific topic can subscribe to it via the broker that notifies them about new messages as soon as they are published. The avoidance of direct requests from clients to the server enables very efficient communication.

Health Care A different approach to sensor data transmission is shown by Khattak et al. through a practical implementation of patient monitoring, which enables doctors to access the data in real time using a CoAP client [KRDS14].

Several sensors are attached to the patient's body – a pulse oximeter to analyze their heart rate and blood oxygen saturation or an EKG monitor to measure cardiac activity – which send their information to a web server, where the raw data is converted. The client can send a `GET` request and receives a response message from the server.

Thanks to the CoAP protocol this data can be easily monitored via traditional Internet browsers and be transmitted and stored to a web server where medical personnel can access and process it.

5 Related Work

Besides the implementation of an IoT application in health care, further research with various focuses on this topic has been studied.

The starting point for this paper was Karagiannis' et al. [Ka15] investigation on existing IoT application layer protocols and protocols used to connect the things but also end-user applications to the Internet. Besides the protocols mentioned above, they also examined RESTful Services, AMQP and HTML 5's Websocket for their suitability for the IoT.

Ezdiani et. al [EAA15] investigated on the QoS requirements for the Internet integration of Wireless Sensor Networks (WSN) by distinguishing the QoS of the Internet from those

for WSN based on applications that involve traffic with different levels of importance, i. e. real-time traffic and delay-tolerant traffic.

In his thesis Lucio [LS16] evaluated the CoAP protocol for constrained nodes with regard to its key features and functionalities and compared it to further existing web protocols that are applied in similar conditions.

Finally, Fysarakis et. al [Fy16] examined the heterogeneity in hardware, network and overlaying technologies of resource-constrained platforms integrated into smart environments. They evaluated the standardized protocols DPWS, CoAP and MQTT in the context of designing and implementing applications for IoT devices.

6 Conclusion

In this paper we have given an overview of the main application protocols suitable for the IoT and their requirements compared to the classic TCP/IP protocols. For this purpose a recapitulation of basic architecture models handling communication was made.

It was shown that even though most embedded devices have very similar if not the same resource constraints – i. e. shared network bandwidth, shorter lifetime or connectivity interruptions due to battery supply and limited computational resources – CoAP, MQTT and XMPP, all being standardized protocols, take different approaches in terms of communication paradigms, transport protocols, Quality of Service levels and security mechanisms (see Figure 1 in section 3.3.

In the end choosing the most suitable protocol for an IoT application highly depends on the constraints of a given embedded device as well as on the requirements towards the intended application.

References

[EAA15] Ezdiani, Syarifah; Al-Anbuky, Adnan: Modelling the Integrated QoS for Wireless Sensor Networks with Heterogeneous Data Traffic. Open Journal of Internet Of Things (OJIOT), 1(1):1–15, 2015.

[Fy16] Fysarakis, Konstantinos; Askoxylakis, Ioannis; Manifavas, Charalampos; Soultatos, Othonas; Papaefstathiou, Ioannis; Katos, Vasilis: Which IoT protocol? Comparing standardized approaches over a common M2M application. 2016.

[HW16] Happ, Daniel; Wolisz, Adam: Limitations of the Pub/Sub Pattern for Cloud Based IoT and Their Implications. 2016 Cloudification of the Internet of Things (CIoT), pp. 1–6, 2016.

[Ka15] Karagiannis, Vasileios; Chatzimisios, Periklis; Vazquez-Gallego, Francisco; Alonso-Zarate, Jesus: A survey on application layer protocols for the internet of things. Transaction on IoT and Cloud Computing, 3(1):11–17, 2015.

[KKT14] Keoh, Sye Loong; Kumar, Sandeep S; Tschofenig, Hannes: Securing the internet of things: A standardization perspective. IEEE Internet of Things Journal, 1(3):265–275, 2014.

[KRDS14] Khattak, Hasan Ali; Ruta, Michele; Di Sciascio, Eugenio: CoAP-based healthcare sensor networks: A survey. In: Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on. IEEE, pp. 499–503, 2014.

[LB16] Lin, Huichen; Bergmann, Neil W: IoT Privacy and Security Challenges for Smart Home Environments. Information, 7(3):44, 2016.

[Lo10] MQ Telemetry Transport (MQTT) V3.1 Protocol Specification. https://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html, accessed 05-July-2017.

[LS16] Lucio Silva, Ludmilla: , Internet of Things: Pros and cons of CoAP protocol solution for small devices, 2016.

[SA11] Saint-Andre, Peter: Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120, 2011.

[SHB14] Shelby, Zach; Hartke, Klaus; Bormann, Carsten: Constrained Application Protocol (CoAP). RFC 7252, 2014.

[TW10] Tanenbaum, Andrew S.; Wetherall, David J.: Computer Networks. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2010.

[Zh11] Building Facebook Messenger. https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/, accessed 05-July-2017.

Ad-Hoc Netze

Ad Hoc Netze - Anwendungen und Herausforderungen

Felix Desiderato[1], Sebastian Zielinski[2]

Abstract: Ein Ad-hoc Netzwerk ist ein Zusammenschluss von mobilen Endgeräten, die über eine drathlose Verbindung miteinander kommunizieren, zu einem Netzwerk. Diese Netzwerke benötigen keinerlei existierende Infrastruktur und sind in der Lage sich selbst zu verwalten. Jeder mobile Knoten in diesem Netzwerk kann die Funktion eines Routers übernehmen. Desweitern können mobile Knoten sich unabhängig voneinander frei umherbewegen, wodurch sich die Topologie des Netzwerks dynamisch verändern kann. Aufgrund der Flexibilität dieser Netzwerke entstehen zahlreiche Anwendungsmöglichkeite und Herausforderungen. In dieser Arbeit stellen wir mögliche Anwendungen von Ad-hoc Netzwerken dar und erläutern einige Herausforderungen sowie mögliche Lösungsansätze. Der Anwendungsteil erläutert die Verwendung von ANETs in der Katastrophenhilfe und der Fahrzeugtechnik. Der Herausforderungsteil behandelt die Problematik der Energieversorgung, des Routings sowie der Sicherheit in ANETs. Das Ziel dieser Arbeit ist, einen ersten Überblick über Anwendungsmöglichkeiten von sowie Herausforderungen bei Ad-hoc-Netzwerken zu vermitteln.

Keywords: Ad-hoc-Netze; MANET; VANET; Sicherheit; Routing

1 Einleitung

Ad-hoc-Netze (*ANETs*) stellen seit circa 40 Jahren einen Forschungsgegenstand in der Informatik dar und bezeichnen selbstorganisierende Netze, die keinerlei feste Infrastruktur oder eine zentrale Verwaltung benötigen [FL01]. Der Begriff findet auch im mobilen Kontext Verwendung und meint mobile Ad-hoc-Netzwerke, auch *MANET* genannt, bei welchem die Knoten aus mobilen Endgeräten, auch *mobile Knoten* genannt, bestehen.

Die Kommunikation erfolgt entgegen herkömmlichen Netzwerken direkt zwischen Knoten. Die Knoten können sich dabei entweder direkt oder indirekt verbinden, wobei bei indirekten Verbindungen die Knoten selbst als Router fungieren [Ho04]. Da jederzeit neue Knoten dem Netzwerk beitreten bzw. bereits vorhandene Knoten austreten können ändert sich die Topologie des Netzwerks dynamisch. Daher müssen Routen zwischen Kommunikatonspartnern gegebenenfalls neu ermittelt werden. Die Ermittlung einer Route zwischen Kommunikationspartnern wird durch zahlreiche Routingstrategien realisiert, welche auch eine der wesentlichen Herausforderungen in Ad-hoc-Netzwerken darstellen. Ad-hoc-Netze können durchaus mit anderen Netzwerken, auch dem Internet, verbunden sein und müssen nicht isoliert betrieben werden.

[1] TU München, Department of Informations Systems, Boltzmannstraße 15, DE, felix.desiderato@tum.de
[2] LMU München, Department of Computer Science, Oettingerstraße 67, DE, sebastian.zielinski@campus.lmu.de

Ziel dieser Arbeit ist eine zusammenfassende Darstellung der Anwendungsmöglichkeiten und Herausforderung eines Ad-hoc-Netzwerks. Dazu wird in Kapitel zwei auf Anwendungsmöglichkeiten in der Fahrzeugtechnik und der Katastrophenhilfe eingegangen. Kapitel drei erläutert die Herausforderungen der Energieversorgung, des Routings sowie der Sicherheit in ANETs. Zuletzt werden die Erkenntnisse dieser Arbeit in Kapitel vier zusammengefasst.

2 Anwendungsfälle

In diesem Kapitel sollen verschiedene Anwendungsbereiche für Ad-hoc-Netzwerke vorgestellt und erläutert werden. Die Einsatzmöglichkeiten von Ad-hoc-Netzwerken sind vielfältig. Sie variieren von Heim- und Firmennetzen, über die Militärkommunikation bis zu der Verwendung in Bildungseinrichtungen[Ho04]. Im Folgenden sollen allerdings mit der Fahrzeugtechnik und Katastophenhilfe die in der Einleitung bereits genannten wieder aufgegriffen werden.

2.1 Fahrzeug-Ad-hoc-Netzwerke

Wenngleich Fahrzeug-Ad-hoc-Netze keine Neuheit darstellen, so nehmen die Forschungsbemühungen in diesem Gebiet weiter zu. Grund dafür sind neben dem Sicherheitsaspekt auch die Fortschritte in IEEE802.11p (Erweiterung des IEEE802.11-Standards auf Kraftfahrzeuge) [Ha09]. Studien zeigen, dass mehr als 60 Prozent der Unfälle, die auf den Straßen passieren, verhindert werden könnten, wenn die Fahrer durch eine Warnung auf die drohende Gefahr aufmerksam gemacht worden wären [EZL14]. Fahrzeug-Ad-Hoc-Netzwerke, engl. Vehicular Ad Hoc Networks, auch *VANETs* genannt, sind eine Form von MANETs, deren mobile Knoten aus Fahrzeugen bestehen. Auch wenn VANETs durchaus Ähnlichkeiten zu MANETs im Bezug auf Selbstverwaltung und –organisation, eine beschränkte Bandbreite und Übertragungsreichweite aufweisen, gibt es doch signifikante Unterschiede. So haben VANETs keine Einschränkungen bezüglich Stromverbrauch und Speicher, da die Knoten Fahrzeuge sind [LW07]. Der nächste Absatz soll die von Schoch et al. in ihrer Arbeit [SKW08] angeführten Unterschiede im Vergleich zu MANETs vorstellen.

Für VANETs ist eine sehr hohe Geschwindigkeit einzelner Knoten charakteristisch und damit verbunden auch eine schnelle Veränderung der Topologie des Netzes, da die Kommunikation der Fahrzeuge auch bei Fahrten auf Autobahnen und Schnellstraßen gewährleistet werden muss. Hingegen zu MANETs ist das Bewegungsmuster der einzelnen Knoten nicht zufällig, sondern kann zu einem großen Maß vorhergesagt werden, sofern sich das Fahrzeug auf verzeichneten Straßen bewegt. Dies hilft auch eine Häufigkeitsverteilung der Knoten zu bestimmen und ermöglicht ein Kategorisieren des Auftretens in drei grundsätzliche Bereiche: ländlich, städtisch und Schnellstraßen. Ein weiterer Unterschied zu MANETs ist die Heteorgenität der Knoten. Durch die Verschiedenheit der unterschiedlichen Fahrzeugtypen und -herstellern kann nicht gewährleistet werden, dass jeder Knoten auf dieselbe Weise

Nachrichten verarbeitet und weiterleitet. Ältere Fahrzeuge beispielsweise sind gar nicht in der Lage Daten zu empfangen. Außerdem ist noch die Dichte von Fahrzeugen in einem bestimmten Gebiet zu nennen. Diese kann von null bis zu mehreren hundert Fahrzeugen reichen und führt zu zwei Extrema in diesem Kontinuum. Entweder es sind kaum Fahrzeuge in Übertragungsreichweite, so dass eine normale Übertragung nicht möglich ist. Dann muss die zu übertragende Information gespeichert werden und kann erst dann übertragen werden, wenn ein Fahrzeug in Reichweite ist. Das kann dazu führen, dass dieselbe Nachricht mehrmals von demselben Fahrzeug übertragen werden muss. Bei einer hohen Fahrzeugdichte ist es wichtig, dass jede Nachricht möglichst nur von ausgewählten Knoten auf der Übertragungsroute gesendet wird, um das Netz nicht zu überlasten.

Im Folgenden soll nun noch die Architektur eines VANETs beleuchtet werden. Bei VANETs kann neben der rein infrastrukturlosen Kommunikation zwischen Fahrzeugen, vehicle-to-vehicle (*V2V*), auch noch der Bedarf nach sog. *permanent network nodes*, also festen Knoten am Fahrbahnrand, die für eine vehicle-to-infrastructure (*V2I*) communication nötig sind, bestehen. Diese werden auch road side units, *RSU*, genannt. Der Aufbau eines VANETs ist somit entweder rein WLAN- bzw. Mobilfunk-basiert, rein ad hoc basiert oder ein Hybrid aus beiden Technolgoien, siehe Abb. 1 [SAI14].

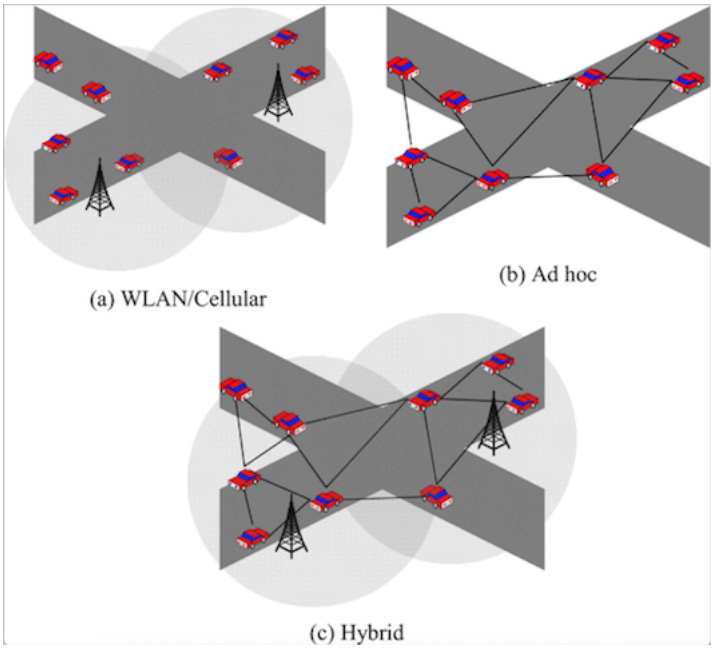


Abb. 1: Drei verschiedene VANET Architekturen[LW07]

In (a) in der Abbildung 1 erkennt man eine reine V2I – Kommunikation. Die Fahrzeuge erhalten durch RSUs Internetzugang und Verkehrsinformationen. Es findet keine Kommunikation

zwischen den Fahrzeugen statt. (b) zeigt eine reine V2V – Kommunikation. Die Fahrzeuge verhalten sich wie mobile Endgeräte und senden sich über ihre Onboard Units (*OBUs*) Nachrichten [SAI14][NAG04]. (c) In Abbildung 1 zeigt die Hybrid-Kommunikation aus V2I und V2V. Durch Ersteres wird die Kommunikation zu RSUs möglich und damit auch die Verbindung mit Standards wie 3G, LTE oder IEEE 802.11. Durch die Kommunikation zwischen den Fahrzeugen ist es auch Fahrzeugen außerhalb der Reichweite eines RSU möglich, mit diesen zu kommunizieren. Durch das Nutzen von beiden Technologien ist das Netz nicht abhängig von RSUs, maximiert aber deren Nutzen, wenn diese vorhanden sind [JN15].

2.2 Ad-hoc-Netzwerke als Katastrophenhife

In Katastrophenszenarien kann kaum oder gar nicht mehr auf feste Kommunikationsinfrastruktur zurückgegriffen werden [Go15]. Dies macht Ad-hoc-Netze im Kontext der Katastrophenkommunikation sehr attraktiv. Anschließend soll nun dieses Einsatzgebiet genauer betrachtet werden.

Ad-hoc-Netzwerke können auf verschiedene Arten als Notfallhilfe dienen. So kann es den Betroffenen entweder möglich sein, sich mit anderen Betroffenen in Verbindung zu setzen oder Helfer zu kontaktieren. Andersrum ist es Helfern leichter möglich Betroffene zu orten und Anweisungen zu geben. Es existieren verschiedene Ansätze, inwiefern MANETs zur Katastrophenhilfe eingesetzt werden können.
Ein Konzept, das die Autoren Di Felice et al. vorstellen, ist das Bilden eines MANETs mit Drohnen, um so nicht nur einen großen Bereich abdecken zu können, sondern auch die mobilen Knoten flexibel, je nach Situation, verlegen zu können. Hierbei werden Drohnen mit einer passenden IEEE 802.11 Technologie ausgestattet und erlauben es so den Betroffenen sich mit dem von ihnen aufgespannten Netzes zu verbinden [Di14].

Es soll im Folgenden auf MANETS, VANETs sowie WSNs (*Wireless Sensor Networks*) und deren Eignung für Katastrophenhilfe eingegangen werden.
Es gibt auch Standards in der Kommunikation im Katastrophenfall, beispielsweise TETRA. TETRA (*TErrestial Trunked RAdio*) ist eine infrastrukturlose Kommunikationstechnologie, die hauptsächlich von Polizei und Feuerwehr verwendet wird [Re14]. Allerdings ist gerade die Kommunikation zwischen Katastrophenopfern und Helfern im Ernstfall essentiell und macht damit TETRA weitestgehend nutzlos. Denn TETRA erlaubt nur eine Kommunikation unter TETRA Endgeräten. Es kann somit keine Verbindung zu Nicht-TETRA Geräten hergestellt werden. Daher ist eine Technologie, die gängige und weitverbreitete Standards nutzt, vorzuziehen.

Die Evaluation über den Nutzen eines MANETs besteht hauptsächlich im Vergleich verschiedener Routingprotokolle im Katastrophenfall [Re14]. Mehrere Arbeiten haben in ihren Studien das Ad-hoc-demand Distance Vector (*AODV*) Protokoll im Falle eines Desasters favorisiert. AODV ist ein reaktives Routingprotokoll, das Routingtabellen mit

einem Eintrag pro Ziel speichert [Re11]. D. G. Reina et al. haben drei Routingprotokolle - AODV, DSR und AOMDV - in drei verschiedenen Szenarios, deren MANETs aus 150-200 Knoten bestanden, untersucht. Hier konnte AODV durch bessere Mobilität und geringeres Delay gegenüber den anderen Protokollen überzeugen [Re11]. Die Autoren Raffelsberger et al. betrachten als Szenario die Explosion einer Chemiefabrik mit 25 mobile Knoten in diesem Areal. In der Packet Delivery Ratio *PDR*), einer der wichtigsten Eigenschaften eines Routingprotokolls, schnitt auch hier das AODV am besten ab. PDR gibt den Anteil der angekommenen Pakete zu den gesendeten an [RH12]. Eine Alternative zu traditionellen Routingprotokollen bietet das Nutzen von *opportunistic contacts*. Bei diesem Verfahren können Knoten miteinander kommunizieren ohne das eine Route zwischen ihnen besteht. Außerdem haben die mobilen Knoten, anders als bei MANETs keinerlei Kenntnis über die Topologie des Netzes [PPC06].

Wie bei MANETs stellen auch bei VANETs die Auswahl des richtigen Routingprotokolls eine Herausforderung dar. Jedoch gibt es auch einige signifikante Unterschiede [Re14]. Die deutlich höhere Geschwindigkeit der Knoten verlangt nach einer Möglichkeit auch eine Verbindung entsprechend schnell herstellen zu können. Außerdem existiert neben der reinen Fahrzeugkommunikation (*V2V*) auch noch die Kommunikation zu RSUs (*V2I*), wenngleich letztere im Katastrophenfall vernachlässigt werden darf, da diese oftmals durch die Katastrophe zerstört wurden [Re14]. Der aktuelle Forschungsgegenstand von VANETs im Katastrophenfall besteht, ähnlich zu MANETs, in der Protokollverbesserung beim Senden von Notfallnachrichten [Re14].

Abschließend sollen noch Wireless Sensor Networks (*WSN*) betrachtet werden. Ein WSN ist ein zentriertes Cluster-Netzwerk. Die Knoten des Netzwerkes sammeln bestimmte Daten und senden diese dann an den zentralen Knoten [Re14]. Durch seine Architektur ist ein WSN besonders als Frühwarnsystem und in der Katastrophenprävention geeignet. Einsatzgebiete sind demnach neben erdbeben- und tsunamigefährdeten Regionen auch die Prävention von Waldbränden [Re14]. Aktive Forschungsthemen, die WSNs in Katastrophenszenarien als Gegenstand haben, behandeln weitestgehend den Einsatz als Frühwarnsystem bzw. als Post-Katastrophen Hilfe.

Dieses Kapitel hat dem Leser eine Vorstellung über zwei wichtige Anwendungsgebiete der Ad-hoc Technologie gegeben. Auf die Herausforderungen, die bereits in diesem Abschnitt angeschnitten wurden, soll im Folgenden noch deutlicher eingegangen werden.

3 Herausforderungen

In MANETs fungiert jeder mobile Knoten als unabhängiger Router in einem drahtlosen Netzwerk, dessen Topologie sich häufig und auf unverhersagbare Weise ändern kann. Daraus ergibt sich der Vorteil, nicht von einer bereits vorhandenen Infrastruktur bzw. zentralen Administration abhängig zu sein. Die gewonnene Flexibilität bringt jedoch einige Herausforderungen und Probleme mit sich [CCL03].

Im Folgenden werden die Herausforderungen der Energieversorgung, des Routings sowie der Netzwerksicherheit weiter ausgeführt.

3.1 Energieversorgung

Die Energieversorgung von mobilen Endgeräten ist abhängig von Batterien und Akkus, deren Kapazität endlich ist. Ein wichtiges Ziel ist daher, Technologien zu entwickeln, die möglichst wenig Energie verbrauchen, so dass die Lebensdauer einer Batterie bzw. eines Akkus und damit auch die Lebensdauer eines mobilen Knoten maximiert wird. So gibt es bereits Displays oder CPUs die weniger Energie verbrauchen [SWR98]. Untersuchungen zeigen, dass der Energieverbrauch eines Netzwerkinterfaces, gerade bei kleineren mobilen Endgeräten, signifikant sein kann [FN01]. Daher ist ein wichtiges Ziel die Entwicklung von energieschonenden Routingverfahren. Auch bei der Entwicklung von Sicherheitslösungen müssen die zur Verfügung stehenden Energiekapazitäten mit berücksichtigt werden.

3.2 Routing in MANETs

Effizientes und dynamisches Routing is eine der Schlüsselherausforderungen in MANETs [MD05]. Die Kombination aus verschiedenen Eigenschaften von MANETs machen das Routing zu einer Herausforderung. Mobile Knoten können sich frei umherbewegen, was dazu führt, dass sich die Netzwerktopologie schnell und häufig verändern kann. Im Gegensatz zu kabelgebundenen Netzwerken ist die Bandbreite bei MANETs aufgrund des drahtlosen Mediums (Luft) deutlich geringer und variabler. Außerdem wird das Kommunikationsmedium von mehreren Nutzern gleichzeitig benötigt, was die verfügbare Bandbreite zusätzlich reduziert [MD05]. Wie in Abschnitt 3.1 bereits erwähnt, spielt auch der Energieverbrauch von Routingprotokollen eine Rolle, da durch einen reduzierten Energieverbrauch die Lebensdauer eines mobilen Knoten verlängert werden kann.
Nach Pathan und Hong [PH09] können Routingverfahren für MANETs grob in zwei Kategorien eingeteilt werden: *proaktive Routingverfahren* und *reaktive Routingverfahren*. Diese werden im Folgenden kurz dargestellt.

Bei **proaktiven Routingverfahren** werden kontinuierlich Informationen über die Netzwerktopologie zwischen den verschiedenen mobilen Knoten ausgetauscht. Das hat zur Folge, dass eine Route zu einem Ziel sofort verfügbar ist, wenn sie benötigt wird. Ändert sich die Netzwerktopologie jedoch zu häufig, so sind die Kosten, um stets alle Routen im Netzwerk zu aktualisieren unter Umständen sehr hoch. Anderenfalls, falls die Netzwerkaktivität gering ist, werden viele Informationen über die Netzwerktopologie möglicherweise nicht benötigt. In diesem Fall wird der Aufwand, um stets Routen zu allen mobilen Knoten im Netzwerk bereitzuhalten, umsonst betrieben und Energie verschwendet. Dies kann dazu führen, dass das Netzwerk eine kürzere Lebensdauer hat, als erwartet wird.

Bei **reaktiven Routingverfahren** wird erst dann nach einer Route zwischen Kommunikationspartnern gesucht, wenn sie auch tatsächlich benötigt wird. Das bedeutet, dass beim Einsatz von reaktiven Routingprotokollen, im Gegensatz zu proaktiven Routingprotokollen, beim Versenden von Nachrichten mit einer gewissen Verzögerungszeit zu rechnen ist, da erst eine Route zum Kommunikationspartner ermittelt werden muss. Allerdings findet kein periodisches Versenden von Informationen der Netzwerktopologie statt, weshalb diese Art von Routingverfahren als ressourcenschonend bezeichnet werden.

3.3 Sicherheit in MANETs

Eines der vorrangigsten Anliegen ist die Verbesserung der Sicherheit in MANETs. Durch das Benutzen von drahtlosen Verbindungen zwischen den verschiedenen mobilen Knoten wird das Netzwerk für Angriffe, die von passivem Belauschen bishin zu aktivem Stören reichen, empfänglich [ZL05], denn das drahtlose Medium ist sowohl legitimen Netzwerkbenutzern als auch schädlichen Angreifern zugänglich [Ya04]. Im Gegensatz zu kabelgebundenen Netzwerken, bei denen ein Angreifer physischen Zugang zu den Netwerkkabeln erlangen bzw. mehrere Sicherheitshürden, wie z. B. Firewalls, überwinden muss, können Angriffe auf ein drahtloses Netzwerk aus allen Richtungen kommen und jeden beliebigen mobilen Knoten betreffen [ZL05].
Desweiteren stellt jeder mobile Knoten eine eigenständige Einheit dar, der es möglich ist sich unabhängig von anderen mobilen Knoten umherzubewegen. Das bedeutet jedoch, dass mobile Knoten, die nicht adäquat phyisch geschützt werden, Gefahr laufen gestohlen oder kompromittiert zu werden [ZL05]. Ein weiteres Sicherheitsproblem stellt die begrenzte Rechenkapazität eines mobilen Knoten dar. So sind Geräte wie zum Beispiel PDAs kaum in der Lage rechenintensive Aufgaben wie zum Beispiel asymmetrische Kryptographie durchzuführen [Ya04].

3.3.1 Sicherheitsanforderungen

Die Sicherheitsanforderungen an MANETs sind den Anforderungen an andere Netzwerke sehr ähnlich. Nach Djenouri et al. [DKB05] sollten effektive Schutzmaßnahmen fünf wesentliche Anforderungen sicherstellen: *Verfügbarkeit*, *Authentizität*, *Vertraulichkeit der Daten*, *Integrität* und *Nichtabstreitbarkeit*.
Verfügbarkeit bedeutet, dass Netzwerkdienste dann zur Verfügung stehen müssen, wenn sie benötigt werden. Sicherheitsmaßnahmen, die die Verfügbarkeit garantieren sollen, versuchen Denial of Service Angriffe bzw. Angriffe, die darauf abzielen die Energiereserven von mobilen Knoten aufzubrauchen, abzuwehren.
Authentizität soll sicherstellen, dass sich kein infizierter bzw. bösartiger mobiler Knoten als ein vertrauenswürdiger mobiler Knoten des Netzwerks ausgeben kann.
Vertraulichkeit ist eines der wichtigsten Sicherheitsanforderungen an Ad-hoc-Netze. Diese Anforderung soll sicherstellen, dass eine gesendete Nachricht lediglich vom beabsichtigten

Empfänger gelesen werden kann. Dies wird typischerweise durch Verschlüsselung ermöglicht.
Integrität soll sicherstellen, dass die Daten, die ein mobiler Knoten von einem anderen mobilen Knoten erhält, authentisch sind, also auf dem Transportweg nicht von einem infizierten bzw. bösartigen mobilen Knoten modizifiert wurden.
Nichtabstreitbarkeit stellt sicher, dass eine Nachricht einen legitimen Ursprung hat. Es soll möglich sein, dass bei Erhalt einer falschen Nachricht der Versender dieser Nachricht nicht abstreiten kann, dass die Nachricht von ihm gekommen ist.

3.3.2 Angriffe auf MANETs

Angriffe auf MANETs können nach Zhang und Lee [ZL05] grob in zwei Kategorien eingeteilt werden: *externe Angriffe* und *interne Angriffe*. Bei externen Angriffen ist ein Angreifer kein Bestandteil des Netzwerks, befindet sich aber im Umfeld des Netzwerks. Bei internen Angriffen hingegen ist ein Angreifer ein Teilnehmer des Netzwerks. Es existieren für viele Schichten des OSI-Modells spezielle Angriffe. Dies wird in Tabelle 1 dargestellt. Die verschiedenen Angriffe auf MANETs in Tabelle 1 sind [Wu07] entommen.

Application Layer	Viren, Würmer
Transport Layer	SYN Flooding(DoS), Sessionhacking
Network Layer	Verschiedene Routingangriffe (z.B. Flooding Angriff, Black Hole Angriff)
Data Link Layer	Caputre-Effekt Schwachstelle, absichtlich unkooperatives Verhalten
Physical Layer	Lauschangriffe, Stören der Übertragung

Tab. 1: Verschiedene Angriffe auf MANETs auf den verschiedenen OSI-Schichten.

Im Folgenden wird nun exemplarisch ein konkreter Angriff auf MANETs vorgestellt.

3.3.3 Flooding Angriff auf MANETs

Der Flooding Angriff auf der Vermittlungsschicht wurde erstmals von Yi et al. [Yi05] vorgeschlagen und stellt einen effektiven Denial of Service Angriff gegen alle gegenwärtigen reaktiven Routingprotokolle dar. Die folgenden Erklärungen und Grundlagen die für diesen Angriff benötigt werden sind vollständig [Yi05] entnommen.
Die Routenfindung im AODV-Protokoll, einem reaktiven Routingprotokoll, funktioniert dabei grundsätzlich wie folgt: Will ein mobiler Knoten eine Verbindung zu einem anderen mobilen Knoten aufbauen, zu dem er keine Route kennt, so versendet er ein RREQ-Paket (Route-Request-Paket) an alle Nachbarn. Bei Erhalt eines solchen Pakets, überprüft der mobile Knoten, der diese Nachricht empfangen hat, ob er eine Route zu dem gewünschten

Ziel kennt. Falls nicht, so merkt sich der mobile Knoten, von welchen seiner Nachbarn er das RREQ-Paket empfangen hat, und versendet das RREQ-Paket weiter an alle seine Nachbarn. Kennt der mobile Knoten einen Route zum gewünschten Ziel, so wird ein RREP-Paket (Route-Reply-Paket) an denjenigen Nachbarn zurückgeschickt, von dem er das RREQ-Paket ursprünglich erhalten hat.
Beim Flooding Angriff sendet nur ein bösartiger mobiler Knoten sehr viele verschiedene RREQ-Pakete in sehr kurzer Zeit, mit dem Wunsch Verbindungen zu mobilen Knoten aufzubauen, die im Netzwerk nicht existieren.
Dies hat nun mehrere negative Auswirkungen auf das gesamte Netzwerk. Zum einen merkt sicht jeder mobile Knoten den Nachbarn, von dem er ein bestimmtes RREQ-Paket empfangen hat. Da kein mobiler Knoten ein RREP-Paket verschicken kann, werden die Einträge bezüglich der Nachbarn von dem ein RREQ-Paket empfangen wurde länger behalten. Da auch der Speicherplatz für die Routingtabelle begrenzt ist, führt dies bei Erhalt von zahlreichen RREQ-Paketen dazu, dass der Speicher voll läuft, wodurch ein mobiler Knoten keine weiteren RREQ-Pakete mehr empfangen kann. Als unmittelbare Folge können diese mobilen Knoten nun keine neuen Verbindungen zu anderen Knoten aufbauen um Daten zu verschicken. Desweiteren wird die zur Verfügung stehende Bandbreite stark durch das Übertragen der zahlreichen RREQ-Pakete beeinträchtigt.

3.3.4 Verteidigungsmaßnahmen gegen Angriffe auf MANETs

Gegen die in Abschnitt 3.3.2 genannten Angriffe gibt es verschiedenste Verteidigungsmöglichkeiten. Diese werden in Tabelle 2 dargestellt. Die jeweiligen Verteidigungsmöglichkeiten sind [Wu07] entnommen.

Application Layer	Intrusion Detection System
Transport Layer	Firewalls, eingeschränkt auch Verwendung von TLS/SSL Protokollen
Network Layer	Verschiedene Sicherheitsmetriken, Sichere (Routing)Protokolle
Data Link Layer	Nachbarn Beobachten das Verhalten eines Mobile Nodes
Physical Layer	Frequency Hopping, Direct Sequence

Tab. 2: Verschiedene Verteidigungsmaßnahmen gegen Angriffe auf MANETs auf den verschiedenen OSI-Schichten.

3.3.5 Verteidigung gegen den Flooding Angriff auf MANETs

Gegen den in Abschnitt 3.3.3 dargestellten Flooding Angriff auf MANETs geben die vorschlagenden Autoren Li et al. [Yi05] auch eine mögliche Verteidigungsstrategie, die im

Folgenden kurz dargestellt wird.
Als wichtige Beobachtung wird festgehalten, dass sämtliche RREQ-Pakete eines bösen mobilen Knoten zuerst an seine unmittelbaren Nachbarn im Netzwerk gesendet werden, die diese Pakete dann weiterleiten. Um nun einen Flooding Angriff zu verhinden, legt man eine maximale Anzahl an RREQ-Paketen fest, die innerhalb einer bestimmten Zeitspanne gesendet werden dürfen. Werden mehr als diese maximale Anzahl an RREQ-Paketen in der festgelegten Zeitspanne von einem bestimmten Knoten versendet, so verweigern die Nachbarn das weiterleiten sämtlicher Pakete dieses mobilen Knotens. Dadurch ist der böse mobile Knoten dann zwar noch Teil des Netzwerks, jedoch komplett isoliert, da er nicht mehr mit anderen mobilen Knoten kommunizieren kann.

3.3.6 Fazit zur Verteidigung gegen Angriffe auf MANETS

Sicherheit in MANETs zu gewährleisten stellt eine große Herausforderung dar. Aufgrund der zahlreichen Angriffspunkte auf verschiedenen Schichten des OSI-Modells gibt es keine universelle bzw. einfache Verteidigungsstrategie. Eine gute Verteidigungsstrategie muss die jeweiligen Gegenbenheiten in einem MANET (z.b. Rechen- und Speicherkapazität, Energieversorgung, Protokollauswahl etc.) berücksichtigen und Sicherheitsmechanismen für verschiedene Schichten bereitstellen [Ya04].

4 Zusammenfassung

In dieser Arbeit wurde zunächst auf zwei Anwendungsbereiche eines Ad-hoc-Netzes eingegangen. Zuerst wurden Fahrzeug-Ad-hoc-Netze (VANETs) vorgestellt. Hierbei wurden Charakteristiken im Aufbau und Architektur sowie Unterschiede zu MANETs beleuchtet. Als zweiter Anwendungsbereich wurde die Katastrophenhilfe vorgestellt. Es wurde aufgezeigt, welche Ad-hoc-Technologie geeignet sind. Zudem wurde bei VANETs und MANETs eine Auswahl an Szenarien gezeigt und auf die Wahl der richtigen Protokolle eingegangen. Abschließend wurden mit WSNs noch eine Technologie zur Katastrophenprävention behandelt. Desweiteren wurden die Herausforderungen der Energieversorgung, des Routings sowie der Sicherheit in MANETs genauer dargestellt. Zuerst wurde dargestellt, dass die effiziente Nutzung der Energie, die jedem mobilen Knoten in einem MANET zur Verfügung steht, einen wichtigen Beitrag zur Verlängerung der Lebensdauer eines mobilen ad-hoc Netzes darstellt. Danach wurden Routingverfahren grob in proaktive und reaktive Verfahren unterteilt. Während proaktive Verfahren sofort Routen zwischen Kommunikationspartnern bereitstellen können, ist bei reaktiven Verfahren mit einer gewissen Verzögerung zu rechen. Reaktive Verfahren erweisen sich dafür als ressourcenschonender als poraktive Verfahren. Abschließend wurden Sicherheitsaspekte in MANETs betrachtet. Nach einer kurzen Darstellung der wüschens- bzw. schützenswerten Eigenschaften wurde dargelegt, dass es für viele Schichten des OSI-Modells verschiedene Angriffe gibt weshalb es keine universelle Verteidiungsstrategie gibt und eine gute Verteidigungsstrategie die individuellen

Gegebenheiten in einem MANET(z.B. Rechen- und Speicherkapazität, Protokollauswahl etc.) berücksichtigen muss. Nachdem in dieser Arbeit ein Überblick über die verschiedenen Herausfordungen gegeben wurde, betreffen zukünftige Forschungsvorhaben die Entwicklung von neuen Sicherheitskonzepten und Routingstrategien, die die zur Verfügung stehenden Ressourcen bestmöglich nutzen und gleichzeitig möglichst effizient arbeiten.

Literaturverzeichnis

[CCL03] Chlamtac, Imrich; Conti, Marco; Liu, Jennifer J-N: Mobile ad hoc networking: imperatives and challenges. Ad hoc networks, 1(1):13–64, 2003.

[Di14] Di Felice, Marco; Trotta, Angelo; Bedogni, Luca; Chowdhury, Kaushik Roy; Bononi, Luciano: Self-organizing aerial mesh networks for emergency communication. In: Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on. IEEE, S. 1631–1636, 2014.

[DKB05] Djenouri, Djamel; Khelladi, Lyes; Badache, Nadjib: A survey of security issues in mobile ad hoc networks. IEEE communications surveys, 7(4):2–28, 2005.

[EZL14] Eze, Elias C; Zhang, Sijing; Liu, Enjie: Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward. In: Automation and Computing (ICAC), 2014 20th International Conference on. IEEE, S. 176–181, 2014.

[FL01] Freebersyser, James A; Leiner, Barry: A DoD perspective on mobile ad hoc networks. In: Ad hoc networking. Addison-Wesley Longman Publishing Co., Inc., S. 29–51, 2001.

[FN01] Feeney, Laura Marie; Nilsson, Martin: Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In: INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Jgg. 3. IEEE, S. 1548–1557, 2001.

[Go15] Gorbil, Gokce: No way out: Emergency evacuation with no internet access. In: Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on. IEEE, S. 505–511, 2015.

[Ha09] Hafeez, Khalid Abdel; Zhao, Lian; Liao, Zaiyi; Ma, Bobby Ngok-Wah: The optimal radio propagation model in VANET. In: Systems and Networks Communications, 2009. ICSNC'09. Fourth International Conference on. IEEE, S. 6–11, 2009.

[Ho04] Hoebeke, Jeroen; Moerman, Ingrid; Dhoedt, Bart; Demeester, Piet: An overview of mobile ad hoc networks: Applications and challenges. Journal-Communications Network, 3(3):60–66, 2004.

[JN15] Jeremiah, Chukwu; Nneka, Agwu Joy: Issues and possibilities in Vehicular Ad-hoc Networks (VANETs). In: Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), 2015 International Conference on. IEEE, S. 254–259, 2015.

[LW07] Li, Fan; Wang, Yu: Routing in vehicular ad hoc networks: A survey. IEEE Vehicular technology magazine, 2(2), 2007.

[MD05] Marina, Mahesh K; Das, Samir R: Routing in mobile ad hoc networks. In: Ad Hoc Networks, S. 63–90. Springer, 2005.

[NAG04] Namboodiri, Vinod; Agarwal, Manish; Gao, Lixin: A study on the feasibility of mobile gateways for vehicular ad-hoc networks. In: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks. ACM, S. 66–75, 2004.

[PH09] Pathan, Al-Sakib Khan; Hong, Choong Seon: Routing in mobile ad hoc networks. In: Guide to Wireless Ad Hoc Networks, S. 59–96. Springer, 2009.

[PPC06] Pelusi, Luciana; Passarella, Andrea; Conti, Marco: Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. IEEE Communications Magazine, 44(11), 2006.

[Re11] Reina, DG; Toral, Sergio L; Barrero, Federico; Bessis, Nik; Asimakopoulou, Eleana: Evaluation of ad hoc networks in disaster scenarios. In: Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on. IEEE, S. 759–764, 2011.

[Re14] Reina, DG; Coca, J M León; Askalani, M; Toral, SL; Barrero, F; Asimakopoulou, E; Sotiriadis, S; Bessis, N: A survey on ad hoc networks for disaster scenarios. In: Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on. IEEE, S. 433–438, 2014.

[RH12] Raffelsberger, Christian; Hellwagner, Hermann: Evaluation of MANET routing protocols in a realistic emergency response scenario. In: Intelligent Solutions in Embedded Systems (WISES), 2012 Proceedings of the Tenth Workshop on. IEEE, S. 88–92, 2012.

[SAI14] Sharef, Baraa T; Alsaqour, Raed A; Ismail, Mahamod: Vehicular communication ad hoc routing protocols: A survey. Journal of network and computer applications, 40:363–396, 2014.

[SKW08] Schoch, Elmar; Kargl, Frank; Weber, Michael: Communication patterns in VANETs. IEEE Communications Magazine, 46(11), 2008.

[SWR98] Singh, Suresh; Woo, Mike; Raghavendra, Cauligi S: Power-aware routing in mobile ad hoc networks. In: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking. ACM, S. 181–190, 1998.

[Wu07] Wu, Bing; Chen, Jianmin; Wu, Jie; Cardei, Mihaela: A survey of attacks and countermeasures in mobile ad hoc networks. In: Wireless network security, S. 103–135. Springer, 2007.

[Ya04] Yang, Hao; Luo, Haiyun; Ye, Fan; Lu, Songwu; Zhang, Lixia: Security in mobile ad hoc networks: challenges and solutions. IEEE wireless communications, 11(1):38–47, 2004.

[Yi05] Yi, Ping; Dai, Zhoulin; Zhang, Shiyong; Zhong, Yiping et al.: A new routing attack in mobile ad hoc networks. International Journal of Information Technology, 11(2):83–94, 2005.

[ZL05] Zhang, Yongguang; Lee, Wenke: Security in Mobile Ad-hoc networks. In: Ad hoc networks, S. 249–268. Springer, 2005.

Vermittlung in Ad hoc Netzen

Michael Freiwald[1]

Abstract: Eine kosteneffiziente Vermittlung von Paketen in Netzwerken spielt eine zentrale Rolle bei der Datenübertragung. Vor allem bei der dezentralen Organisation in Ad hoc Netzen muss jeder Teilnehmer wissen, zu welchem Nachbarn das Paket als nächstes gesendet werden sollte, damit es möglichst schnell beim Ziel ankommt. In dieser Arbeit wird ein Überblick über die Kategorisierung von Routing-Protokollen für Ad hoc Netze gegeben. Anschließend werden das *Ad hoc On-Demand Distance Vector (AODV) Routing* und das *IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)* vorgestellt. Vor allem RPL ist für den Einsatz im Bereich Internet of Things entwickelt worden.

Keywords: Ad hoc; AODV; RPL; Routing; Protokolle; Netzwerk

1 Einleitung

Die Vermittlung in Netzwerken spielt eine zentrale Rolle in der Datenübertragung. Ohne sinnvollem Routing könnte man heutzutage nicht mit High-Speed im Internet surfen oder unsere gesamten Geräte miteinander vernetzen. Vor allem im Bereich autonomes Fahren, wo die Autos per Funkübertragung miteinander kommunizieren sollen, ist ein schnelles und zuverlässiges Routing wichtig [KKK11]. Damit ein Netzwerkteilnehmer weiß, zu welchem Knotenpunkt ein Paket als nächstes übertragen werden muss, gibt es Routing-Protokolle. In diesen wird definiert, was welcher Netzwerkknoten zu tun hat, um den besten Weg durch das Netz zu finden.

In Ad Hoc Netzen ist dies eine noch größere Herausforderung als in zentral gesteuerten Netzwerken. Bei einem Ad Hoc Netz kennen sich die einzelnen Teilnehmer nicht, jeder kann sich einem Verbund von mehreren Knotenpunkten anschließen und auch wieder verlassen. Hierfür ist keine existierende Infrastruktur notwendig. Gerade in Mobilen Ad Hoc Netzen (MANETs) stellt die dynamische Zusammenstellung des Netzwerks ein Problem da, da sich durch die Beweglichkeit der einzelnen Teilnehmer das Netz beliebig oft verändert [AWD04]. Durch diese dezentrale Struktur muss jedes Gerät sich selbst verwalten können und die Möglichkeit haben, etwas über das Netz zu erfahren. Routing-Protokolle helfen dabei, energiesparend die optimalen Pfade von der Quelle bis zum Ziel zu finden [DP11, S. 12].

[1] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland, m.freiwald@campus.lmu.de

2 Routing

Wie bereits erwähnt kommunizieren die Teilnehmer in einem Ad hoc Netz ohne zentrale Station, wie einem WLAN-Access-Point, direkt miteinander. Liegen Sender und Empfänger einer Nachricht weiter auseinander, als die Sendeleistung des Absenders, so müssen die einzelnen Pakete einer Nachricht über näherliegende Netzwerkteilnehmer weitergeleitet werden. Für eine erfolgreiche Weiterleitung der einzelnen Pakete muss jeder Knoten im Netzwerk wissen, an wen das Paket als nächstes gesendet werden muss. Dieses Vorgehen wird durch ein Routing-Protokoll bestimmt. [DP11, S. 163]

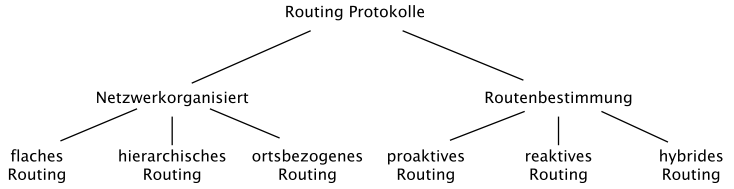


Abb. 1: Kategorische Einteilung von Routing-Protokollen. [DP11, S. 164, Fig. 7.2]

Routing-Protokolle können in verschiedene Kategorien eingeteilt werden. Zwei Möglichkeiten werden von Dargie und Poellabauer in [DP11] erläutert. Die Aufteilung ist in Abbildung 1 dargestellt. Die genauere Erläuterung der einzelnen Kategorien erfolgt im nachfolgenden Absatz. Somit ist eine Möglichkeit der Klassifizierung, Protokolle anhand ihrer Netzorganisation, wie flaches, hierarchisches oder ortsbezogenes Routing, zu definieren. Eine andere Aufteilung entsteht bei der Betrachtung ihrer Routenbestimmung. Je nach dem ob sie reaktiv, proaktiv oder in Kombination ihren Weg durch das Netzwerk finden. [DP11, S. 164-165]

2.1 Netzwerkorganisiertes Routing

Ein Ad hoc Netz kann unterschiedlich strukturiert aufgebaut sein. Die meisten Routingprotokolle passen in eine von drei Klassen, wie in Abbildung 1 auf der linken Seite zu sehen ist. Beim **flachen Routing** kommunizieren alle Knoten eines Netzes direkt miteinander [GSV11]. So geben Dargie und Poellabauer in [DP11, S. 164] an, dass beim flachen Routing alle Teilnehmer die gleiche Funktionalität besitzen. Somit darf jeder Pakete zu Stationen senden, die sich in der unmittelbaren Nähe befinden. Dies hat allerdings den Nachteil, dass so ein Netz in der Größe schlecht skalierbar ist. Tritt die Situation auf, dass zwei vorhandene Netzwerke sich miteinander verbinden, so müssen bei allen Netzwerkknoten die Routingtabellen angepasst werden. Dies führt zu einem enormen Aufwand und kann das Netz stark belasten. [GSV11].

Anders sieht es bei Protokollen mit **hierarchischer Struktur** aus. Laut [DP11, S. 164] haben hier die einzelnen Knoten verschiedene Aufgaben inne. So werden mehrere Knoten

zu einer Gruppe zusammengefasst und es bildet sich eine Baumstruktur. Innerhalb dieser Zusammensetzung kommuniziert jeder Teilnehmer nur mit dem Elternknoten, wie es in Abbildung 2 dargestellt ist. Der Elternknoten kennt alle unter sich liegenden Netzwerkteilnehmer und weiß somit, wohin er ein Paket senden muss, damit es beim richtigen Empfänger an kommt. Liegt der Empfänger nicht unterhalb des Knoten, wird das Paket weiter nach oben gesendet [GSV11]. Dies bringt den Vorteil mit sich, dass bei einer Netzzusammenführung, ein vorhandener Netzwerkbaum an einen Knoten in einem anderen Baum angehängt werden kann. Eine Aktualisierung der Routingtabellen aller Netzwerkteilnehmer erfolgt nur über die in der Baumstruktur gegebenen Pfade. Ein Nachteil an dieser Struktur ist allerdings, dass festgelegt werden muss, wo sich die einzelnen Knoten innerhalb der Baumstruktur befinden. Durch die selbstorganisierende Eigenschaft von Ad hoc Netzen müssen die einzelnen Netzwerkteilnehmer dafür in der Lage sein, untereinander auszuhandeln, wer Eltern- und wer Kindknoten ist.

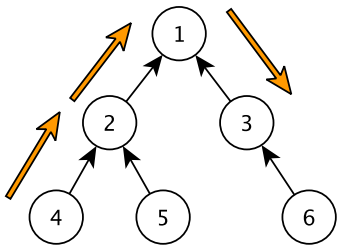


Abb. 2: Hierarchisches Routing. Knoten 4 möchte zu Knoten 6 senden.

Beim **ortsbezogenen Routing** haben die Netzteilnehmer die Möglichkeit ihren eigenen Standort zu bestimmen. Anhand dieser Information können sie für die Weiterleitung von Paketen bessere Entscheidungen treffen. Durch sogenanntes Geocasting besteht die Möglichkeit, Pakete in eine bestimmte Region zusenden. Das Paket muss sich nicht mehr im ganzen Netz ausbreiten, um den richtigen Weg zum Empfänger zu finden. Abbildung 3 zeigt ein zweidimensonales Koordinatenssystem. Knoten A kennt den Standort vom Empfänger (Knoten B) und sendet deshalb das Paket nur in dessen Richtung. Dadurch verringert sich die benötigte Bandbreite und der Energieverbrauch. Ist das Paket am Zielbereich angelangt, kann es durch Multicast an alle Netzwerkknoten verteilt oder durch Anycast an einen bestimmten Teilnehmer gesendet werden. Ein Nachteil beim ortsbezogenen Routing kann sein, dass die Netzteilnehmer die Möglichkeit haben müssen, ihren aktuellen Standort zu einem zentralen Bezugspunkt bestimmen zu können. Dies wäre zum Beispiel durch GPS (Global Positioning System) möglich. Diese Technologien können allerdings den Energieverbrauch erhöhen, obgleich eingebetteten System oft nur geringe Energiespeicher haben [DP11, S. 183].

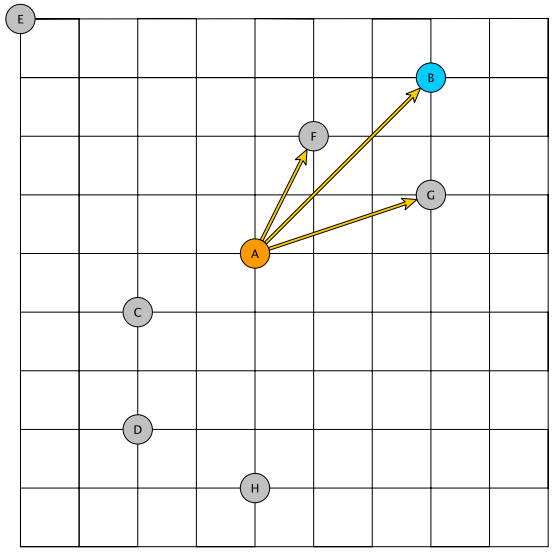


Abb. 3: Ortsbezogenes Routing. Knoten A möchte zu Knoten B senden und kennt seinen ungefähren Standort.

2.2 Routenbestimmung

Bei der Unterteilung von Protokollen anhand ihrer Routenbestimmung unterscheidet man nach proaktiven und reaktiven Routingprotokollen. Das heißt: wie werden die Wege zu anderen Netzwerkteilnehmer gefunden. Die Unterscheidung, welche nachfolgend beschrieben wird, wurde von Abolhasan et al. in [AWD04] erläutert. Bei **proaktiven Routingprotokollen** werden die Pfade zu allen Knoten im Netzwerk in der Startphase ermittelt. Durch einen periodischen Updateprozess werden die Routen aktuell gehalten. Hierbei sendet jeder Teilnehmer in festgelegten Abständen Informationen an seine direkten Nachbarn. Die Routinginformationen werden dabei von den Knoten in Tabellen gespeichert. Soll nun eine Nachricht versendet werden, so kann jeder Teilnehmer in seiner Tabelle den günstigsten Weg zum Empfänger finden. Bei großen Netzen kann dieses Prinzip allerdings eine zu große Überlast bedeuten, da durch das Aktuell halten der Routingtabellen zu viel von der vorhandenen Bandbreite einnimmt. Der Energiebedarf ist ebenfalls sehr hoch, da die Knoten permanent mit dem Updateprozess beschäftigt sind, auch wenn keine Daten durch das Netzwerk zu transportieren sind. Bei **reaktiven Routingprotokollen** werden aktive Routen nur bei Bedarf ermittelt. Dies verringert den Energiebedarf, da keine periodischen Statusnachrichten notwendig sind. Um den Weg zum Ziel zu finden, wird von der Quelle aus ein *Flooding Route Request* gesendet, womit nach dem Zielknoten gesucht wird. Sobald die Anfrage beim finalen Empfänger angekommen ist, sendet er eine Antwort direkt zurück zum Absender. Der Nachteil an diesem Vorgehen besteht darin, dass im schlimmsten Fall

ein Route Request durch das ganze Netz gesendet werden muss, bis das Ziel gefunden wurde. Sobald der Quellknoten die Antwort erhält, wird die zu übertragende Nachricht versendet. Dies kann auf zwei Arten erfolgen. Beim Source Routing kennt der Absender alle benötigten Knoten. Das Paket wird über einen vom Absender festgelegten Pfad durch das Netz geleitet. Jeder Knoten bekommt genau gesagt, wohin weitergeleitet wird. Dies hat einen großen Nachteil in mobilen Netzen, bei denen sich die einzelnen Netzwerkteilnehmer in Bewegung befinden und sich die Routen zueinander oft ändern. Fällt ein Knoten aus, so wird über die festgelegte Route nichts mehr übertragen und das Senden der Nachricht wird eingestellt bzw. eine neue Route muss erst wieder gefunden werden. Dies wird beim Hop-by-Hop Routing besser gehandhabt. Erhält ein Knoten ein weiterzuleitendes Paket, so prüft er seine eigene Routingtabelle nach dem aktuell besten Weg zum Ziel. Dieser kann sich schließlich zwischenzeitlich verändert haben. Ist kein Eintrag vorhanden, so startet dieser auch wieder ein Route Request um den optimalen Pfad zu finden. Allerdings ist durch den vom Sender vorausgegangen Flooding Route Request ein Eintrag in den meisten Fällen bereits vorhanden. Um die Vorteile der beiden Kategorien zu kombinieren, gibt es Protokolle, die beide Verfahren verwenden. Diese gehören zu den **hybriden Routingprotokollen**. Hier wird das Netz oft in Zonen bzw. Cluster eingeteilt. Durch diese Aufteilung können innerhalb von Zonen stromsparende, reaktive Protokolle verwendet werden. Für die Kommunikation zwischen verschiedenen Zonen zueinander kommen dann proaktive Formen zum Einsatz. Schwierig hierbei ist es, zu bestimmen, welche Teilnehmer die Vermittlung zwischen den unterschiedlichen Protokollen vornehmen.

3 Routingprotokolle

Mit der Zeit wurde eine große Auswahl an Routingprotokollen erforscht. Im Folgenden sollen AODV (Ad hoc On-Demand Distance Vector) und RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) genauer erklärt werden. AODV wurde ausgewählt, da es in Sensornetzen sehr weit verbreitet ist. Bei RPL handelt es sich um ein von der IETF 2012 standardisiertes IPv6 Routing Protokoll, welches vor allem in Netzen mit geringer Energieversorgung und schlechten Übertragungen zum Einsatz kommen soll. Spezifiziert ist es in der RFC 6550. Somit wird mit dieser Arbeit ein etabliertes Routingprotokoll und ein relativ neues Protokoll vorgestellt.

3.1 Ad hoc On-Demand Distance Vector (AODV) Routing

AODV ist ein reaktives Routingprotokoll. Dadurch werden keine Routen zu anderen Zielknoten aktuell gehalten, sofern sie nicht für eine aktive Kommunikation benötigt werden [PBRD03]. Das Protokoll verwendet Broadcast-Nachrichten, um dynamisch auf allen erreichbaren Knoten einen Eintrag in der Routingtabelle anzulegen. Dadurch wird den direkten Nachbarn mitgeteilt, dass man selbst erreichbar ist. Dieser Prozess wird immer dann von einem Netzwerkknoten angestoßen, wenn er ein Paket an ein bestimmtes Ziel

senden soll, selbst aber keine Informationen über eine Route diesem Punkt hat. Solch ein *Route Request (RREQ)* ist in Abbildung 4 a) dargestellt. Die Quelle möchte Daten zum Ziel übertragen, kennt aber keinen Pfad durch das Netz. Aus diesem Grund sendet die Quelle ein RREQ an seine direkten Nachbarn. Da diese zu Beginn selbst keine Informationen haben, leiten diese den RREQ an deren Nachbarn weiter. Dies geschieht solange, bis der Zielknoten erreicht wurde. Ein Route Request enthält bei AODV folgende Informationen:

Adresse der Quelle: Eindeutige Adresse von der Quelle im Netz.

Adresse vom Ziel: Eindeutige Adresse von dem Ziel im Netz.

Hop-Count: Der Hop-Count gibt an, wie oft der Route Request weitergeleitet werden darf. Bei jedem Knoten wird der Wert um eins verringert.

Broadcast-ID: Besteht aus der Quell-Adresse und einem Zählwert, der von der Quelle bei jedem neuen RREQ hochgezählt wird. So ist jeder Route Request im Netz eindeutig identifizierbar.

Sequenznummer der Quelle: Fortlaufende Sequenz, wie der aktuelle Zeitstempel, um die Aktualität einer Anfrage bestimmten zu können.

Sequenznummer vom Ziel: Letzte bekannte Sequenz vom Ziel. Diese kann auch nicht vorhanden sein.

Empfängt ein Knoten ein Route Request, prüft dieser in seiner Routingtabelle, ob die Zieladresse vorhanden ist und wenn ja, ob die darin gespeicherte Ziel-Sequenznummer größer ist, als die vom RREQ. Ist dies der Fall, dann ist der Eintrag in der Tabelle aktueller und es kann ein *Route Reply (RREP)* zurückgesendet werden. Die Antwort enthält dabei die Adresse von Quelle und Ziel, die Sequenznummer des Ziels und einen Hop-Count. Der RREP wird wieder im Netzwerk verteilt und ein Knoten leitet ihn nur weiter, wenn in dessen Routingtabelle kein aktuellerer Eintrag eines RREP vorhanden ist. Das heißt, die Sequenznummer im RREP muss größer als die Sequenznummer des Eintrags sein. Handelt es sich um die gleiche Sequenznummer, so wird nur weitergeleitet, wenn der Hop-Count kleiner ist, als beim gespeicherten Route Reply. Somit wird sichergestellt, dass der RREP die kürzeste Route zurück zur Quelle nimmt. Dies ist in Abbildung 4 b) veranschaulicht. Hat ein Knoten ein RREP erhalten, so wird die Route für eine bestimmte Zeit als aktiv deklariert. Danach müsste wieder ein neuer Route Request initiiert werden. Um die Verfügbarkeit seinen Nachbarn mitzuteilen, senden die Netzwerkteilnehmer in periodischen Abständen HELLO Pakete aus. Wird auf einer aktiven Route die Verbindung unterbrochen, so werden Route Error (RERR) Pakete zur Quelle gesendet um ihr mitzuteilen, dass der aktuelle Pfad nicht mehr Verfügbar ist. Daraufhin kann vom Quell-Knoten ein neuer RREQ initiiert werden. [DP11, S. 178-179]

Problematisch wird es bei AODV, wenn sich das Netz drastisch vergrößert. Perkins und Royer zeigen dies in einer Simulation in [PR99]. So nimmt bei starkem Teilnehmeranstieg

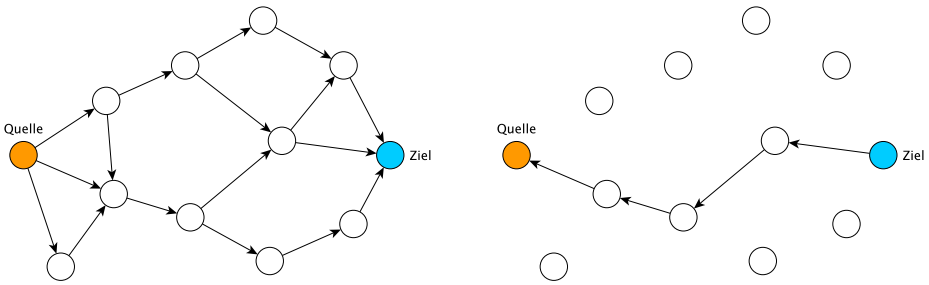


Abb. 4: RREQ und RREP in AODV. [DP11, S. 179, Fig. 7.10]

vor allem der Verlust durch Kollisionen beim Übertragen von Paketen rapide zu. Dadurch erhöht sich die Wartezeit bis auf den Übertragungskanal zugegriffen werden kann, wodurch eine Verzögerung beim Senden von RREQs und RREPs entsteht. Es gibt viele Ansätze, wie AODV noch erweitert und verbessert werden kann. So beschreiben Bianchi et al. in [BPV15], wie die einzelnen Knoten in einem NACK-basierendem AODV (N-AODV) Protokoll das Netzwerk besser wahrnehmen. Dadurch lässt sich, vor allem bei langen Betriebszeiten der einzelnen Knoten, die Effizienz und Wirksamkeit steigern, da seltener eine neue Route erst gefunden werden muss. Abschließend ist noch zu erwähnen, dass AODV ein symmetrisches Netzwerk voraussetzt, da RREP den inversen Pfad von RREQ nimmt. Das heißt, die Erreichbarkeit von zwei Knoten zueinander muss gegeben sein. Es darf nicht vorkommen, dass ein Knoten zu einem Anderen etwas senden kann, die Leistung in die entgegengesetzte Richtung aber nicht für eine Übertragung von Paketen ausreicht. [DP11, S. 179]

Was in dieser Arbeit nicht betrachtet wurde, in der heutigen Zeit aber ein wichtiges Kriterium beim Einsatz sein sollte, ist der Aspekt Sicherheit. In AODV wäre es zum Beispiel möglich, sich bei einem Route Request mit dem drauf folgenden Route Reply als Ziel auszugeben. Dies wäre ein so genannter Man-in-the-Middle Angriff. Aber auch eine Überlastung durch einen Denial-of-Service Angriff ist möglich, in dem vom Angreifer permanent RERR Pakete ausgesendet werden [DP11]. Aus diesem Grund sollten die einzelnen Nachrichtenpakete auch geschützt sein.

3.2 IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)

RPL ist ein von der IETF in RFC 6550 [Wi12] standardisiertes Routing Protokoll. Es wird von der *Routing Over Low power and Lossy networks (ROLL) Working Group* entwickelt und ist vor allem für den Einsatz von IoT-(Internet of Things) Geräten interessant. In

[Wi12] wird von *Low power and Lossy networks (LLNs)* geschrieben. Hierbei handelt es sich um Netze, in dem die einzelnen Knoten wenig Energie zur Verfügung haben und die Übertragungen oft Unterbrechungen aufweisen können. In solchen Netzen ist es wichtig, einen energiesparenden und ausfallsicheren Algorithmus zu verwenden. Für die Entwicklung von RPL wurden vier anwendungsspezifische Routing-Anforderungen verfasst. Diese sind:

- RFC 5548: Routing Requirements for Urban Low-Power and Lossy Networks
- RFC 5673: Industrial Routing Requirements in Low-Power and Lossy Networks
- RFC 5826: Home Automation Routing Requirements in Low-Power and Lossy Networks
- RFC 5867: Building Automation Routing Requirements in Low-Power and Lossy Networks

Alle vier Spezifikationen zeigen auf, dass der Einsatz von RPL vor allem im IoT anzusiedeln ist. Sei es in der Heim- und Gebäudeautomatisierung (RFC5826, RFC2867), Verbindung von Industriemaschinen (RFC 5673) oder Überwachung und Übertragung von verschiedensten Sensordaten (RFC 5548). Dies bedeutet allerdings nicht, dass RPL nicht auch für andere Anwendungszwecke und stabilere Netze verwendet werden kann. [Wi12]

RPL ist ein Distance-Vector Routing Protokoll, wobei das Routing auf gerichtete, azyklische Graphen (engl. DAG - Directed Acyclic Graph) bzw. zielorientierte, gerichtete, azyklische Graphen (engl. DODAG - Destination Oriented Directed Acyclic Graph) basiert [Wi12]. Bevor das Prinzip von RPL aufgezeigt und die Erklärung von DODAG erfolgt, werden die im Zusammenhang zum Protokoll bestehenden, Begriffe erläutert: [Wi12, S. 11-13]

DAG: Gerichteter, azyklischer Graph. Ein gerichteter Graph, bei dem alle Knoten so miteinander verbunden sind, dass keine Schleifen entstehen. Dies wird dadurch erreicht, dass alle Kanten nur in die Richtung (nach oben) eines oder mehrerer *Root Nodes* zeigen.

DAG Root: Ein DAG Root hat keine ausgehenden Kanten. Da der Graph azyklisch ist, müssen alle DAGs mindestens einen DAG Root haben, bei dem alle Wege enden.

Destination-Oriented DAG (DODAG): Hierbei handelt es sich um ein DAG, welcher allerdings nur einen DAG Root besitzt. Dieser wird dann DODAG Root genannt.

DODAG Root: Der oberste Knoten in einem DODAG, ohne ausgehende Kanten. Der DODAG Root kennt alle Kanten im Graphen und kann als Verbindungsglied zu anderen Routingprotokollen dienen. So können z.B. mehrere DODAGs miteinander über das Internet verbunden sein.

Rank: Der Rang eines Knoten gibt dessen relative Position zu anderen Knoten innerhalb des Graphen an. Durch die Objective Function (OF) eines DAG wird der Rang eines jeden

Knoten bestimmt. Die relative Position bezieht sich in diesem Fall immer auf die Wurzel des Graphen.

RPL Instance: Eine RPL Instanz wird durch die RPLInstanceID eindeutig identifiziert und kann mehrere DODAGs enthalten. Ein Knoten kann allerdings nur zu einem DODAG innerhalb einer Instanz gehören.

DODAGID: Die DODAGID gehört immer zu einem DODAG Root und muss in einer RPL Instanz einzigartig sein. Dies ist meist die IPv6-Adresse des Knoten.

DODAG Version: Die DODAG Version gehört immer zu einem DODAG mit fester DODAGID und einen bestimmten Aufbau des Graphen. Verändern sich die Verbindungen der Knoten zueinander, so erhöht sich die Versionnummer des DODAG.

DIO: DODAG Information Object. Durch die DIO werden Informationen über den DODAG ausgetauscht. So erfährt jeder Knoten die Parameter, welche für die RPL Instanz anzuwenden sind und kann sich so seine Elternknoten aussuchen.

DAO: Destination Advertisement Object. Durch DAOs werden Zielinformationen in den Graphen nach oben weiterverbreitet. Nach Erhalten eines DAO senden die Elternknoten gegebenenfalls DAO Acknowledgements (DAO-ACK) an den Absender zurück.

DIS: DODAG Information Solicitation. Durch DIS Nachrichten haben Knoten im DODAG die Möglichkeit, von anderen Teilnehmern ein DIO anzufordern.

Durch Abbildung 5 soll die Verbreitung der Routinginfomationen durch RPL genauer erklärt werden.

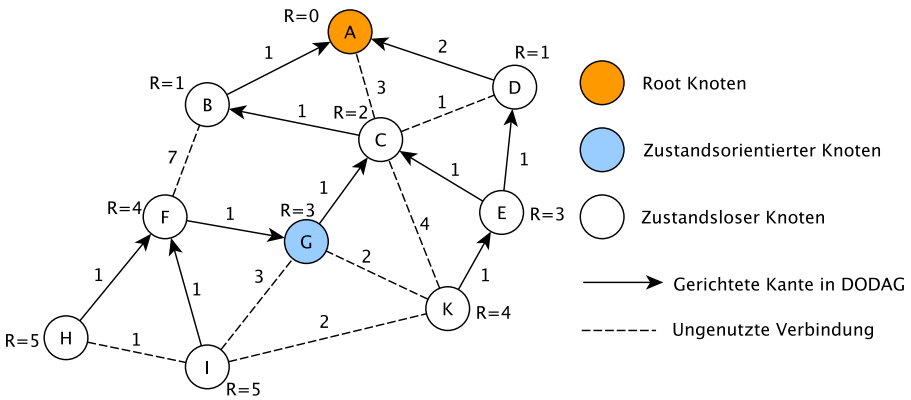


Abb. 5: Routing im RPL. [Ku10]

Alle Knoten senden periodisch DIO Nachrichten an ihre direkten Nachbarn, damit andere Geräte dem DODAG beitreten können und Informationen darüber erhalten. Knotenpunkte

können explizit ein DIO durch eine DIS Nachricht anfordern. Anhand der Informationen wählt ein Knoten seine Elternknoten aus, um einen möglichst kurzen Weg zum DODAG Root zu haben. Hierfür dient die Objective Function. Dadurch erhält man immer Routen, die nach oben zum DODAG Root gerichtet sind. So nimmt in Abbildung 5 Knoten F als Elternknoten G (4 Einheiten zum Root) und nicht Knoten B (8 Einheiten zum Root). E wählt C und D als Elternknoten, da bei beiden 3 Einheiten reichen, um zum DODAG Root zu kommen. [Wi12][Ku10]

Knoten innerhalb eines DODAG kennen nur ihre Elternknoten, nicht aber ihre Kinderknoten. Damit Routen zu ihnen bekannt werden, informieren die Knoten ihre Eltern über ihre Erreichbarkeit durch DAO Nachrichten. Ist ein Teilnehmer im Stande Routinginformationen zu speichern, so kann dieser die Routen zu seinen Kindern zusammenfassen. Ist jemand auf Grund von zu wenig Ressourcen nicht dazu im Stande, so fügt er sich als Next-Hop in der DAO Nachricht an und leitet diese an seine Eltern weiter. Es wird wieder Abbildung 5 als Beispiel verwendet. Knoten G hat die Möglichkeit alle empfangene Routeninformationen zusammenzufügen und zu speichern. Alle anderen Knoten, außer dem DODAG Root, sind Zustandslos. Nun sendet H eine DAO Nachricht an F, dieser fügt sich als Next-Hop-Adresse an die DAO Nachricht an und leitet sie an G weiter. I sendet ebenfalls eine DAO Nachricht an Knoten F, welcher sich wieder als Next-Hop-Adresse an die DAO Nachricht anfügt und sie an G weiterleitet. Nun sendet F eine DAO Nachricht an G, um dessen Verfügbarkeit mitzuteilen. Knoten G kann nun alle Routen zusammenfassen und sie in einer eigenen DAO Nachricht an C senden. C und B sind wieder Zustandslos, wodurch die aggregierte DAO von G einfach an A weitergeleitet wird, nachdem beide sich als Next-Hop-Adresse angefügt haben. A kennt so am Ende alle Routen zu den einzelnen Knoten im Graphen. [Wi12][Ku10]

Wie in [Pa16] erwähnt, haben Radoi et al. in [RSA12] die Performance von proaktiven (RPL) und reaktiven (AODV) Protokollen untersucht. Dabei konnten sie feststellen, dass RPL, bei einer guten Positionierung des Root Node, die niedrigste Latenz hat. Zudem hat RPL im Vergleich zu den anderen Protokollen den besten Energieverbrauch pro Knoten. Wie bei allen hierarchisches Protokollen hat RPL den Nachteil, dass der Root Node als sogenannter „single point of failure“ betrachtet werden kann. Fällt dieser aus, können die Knoten die Pakete nicht mehr richtig durch das gesamte Netz leiten. In der RFC Spezifikation sind Möglichkeiten genannt um das Problem zu lösen, diese anzuwenden ist lauter [Pa16] allerdings nicht immer möglich. Ein weiterer Kritikpunkt ist die Anforderung, welche an ein Root Node gestellt wird. Damit dieser erfolgreich in RPL eingesetzt werden kann, muss er über ausreichend Speicherplatz und Energie verfügen, um die Routen zu den Knoten speichern und zusammenführen zu können. Diese Anforderung ist bei einer dezentralen Struktur mit unbekannten Knoten nicht immer vorhanden [Pa16, S. 51].

Durch RPL wird eine gute Grundlage geschaffen, um IPv6 im Bereich von eingebetteten Kommunikationssystemen weiter zu verbreiten. Vor allem wenn man das Umfeld um IoT betrachtet, wird ihm eine große Rolle zugespielt. [ZL14]

4 Fazit

In dieser Arbeit wurde ein kurzer Überblick über die Vermittlung von Paketen in Ad hoc Netzen gegeben. Dabei wurde auf die unterschiedliche Kategorisierung der Routing-Protokolle eingegangen. So können Protokolle in flaches, hierarchisches oder ortsbezogenes Routing eingeteilt werden. Je nach dem wie es die Netzwerkstruktur vorsieht. Eine andere Einteilung erfolgt in proaktives und reaktives Routing. Dies sagt aus, wie der Weg für ein Paket durch das Netz bestimmt wird. Entweder werden die einzelnen Routen in periodischen Abständen aktuell gehalten oder der Weg wird erst bei Bedarf ermittelt. In großen Netzen wird oft auch eine hybride Lösung bevorzugt. Der Hauptteil der Arbeit konzentriert sich auf die detaillierte Beschreibung von zwei, von der IETF spezifizierten, Protokolle AODV (RFC 3561) und RPL (RFC 6550). AODV stellt ein einfaches Routingverfahren da, bei dem durch Route Requests und darauf folgende Route Responses der kürzeste Weg von Sender zu Empfänger bestimmt wird. Ein Nachteil bei diesem reaktiven Routingprotokoll ist allerdings, dass sich die Übertragungszeiten bei immer größer werdenden Netzen verlängern. Dies ist durch mehr Kollisionen bei der Übertragung und Auslastung der Übertragungskanäle durch zu viele Knoten geschuldet, wodurch sich die Findung des optimalen Pfades durch das Netz verlängert. Bei RPL handelt es sich um ein hierarchisches bzw. proaktives Routingprotokoll. Dessen Eigenschaften weisen sich durch sehr gute Werte beim Energieverbrauch und durch geringe Verzögerungszeiten aus. Durch einen einzigen DODAG Root besteht allerdings die Gefahr, dass im Falles dessen Ausfalls, keine richtige Weiterleitung von Paketen mehr statt findet.

Für weitere Forschungszwecke sollten auf jeden Fall noch Aspekte der Sicherheit bei den einzelnen Routingprotokollen in Betracht gezogen werden. Dies würde über den Umfang dieser Arbeit hinausgehen. Zudem tritt das Thema Internet of Things durch die Medien immer mehr in den Vordergrund. Hierfür gibt es viele weitere Forschungsarbeiten die auch an RPL anknüpfen, wie zum Beispiel die Paketzustellung in *Low Power Wireless Personal Area Networks (6LoWPANs)*. Dies wird von der IETF im RFC 6282 beschrieben. Abschließend sollte noch erwähnt werden, dass es eine Vielzahl an unterschiedlichen Routing-Protokollen gibt und diese Arbeit lediglich den Einstieg in die Thematik erleichtern soll.

Literaturverzeichnis

[AWD04] Abolhasan, Mehran; Wysocki, Tadeusz; Dutkiewicz, Eryk: A review of routing protocols for mobile ad hoc networks. Ad Hoc Networks, 2(1):1 – 22, 2004.

[BPV15] Bianchi, Alessandro; Pizzutilo, Sebastiano; Vessio, Gennaro: Comparing AODV and N-AODV Routing Protocols for Mobile Ad-hoc Networks. In: Proceedings of the 13th International Conference on Advances in Mobile Computing and Multimedia. MoMM 2015, ACM, New York, NY, USA, S. 159–168, 2015.

[DP11] Dargie, Waltenegus; Poellabauer, Christian: Fundamentals of Wireless Sensor Networks: Theory and Practice (Wireless Communications and Mobile Computing). Wiley, 2011.

[GSV11] Gupta, Anuj K.; Sadawarti, Harsh; Verma, Anil K.: A Review of Routing Protocols for Mobile Ad Hoc Networks. WTOC, 10(11):331–340, November 2011.

[KKK11] Kumar, Yugal; Kumar, Pradeep; Kadian, Akash: A survey on routing mechanism and techniques in vehicle to vehicle communication (VANET). International Journal of Computer Science & Engineering Survey (IJCSES), 2(1):135–143, 2011.

[Ku10] Kuryla, Siarhei: , RPL: IPv6 Routing Protocol for Low power and Lossy Networks. Networks and Distributed Systems - Cource at Jacobs University Bremen, März 2010. Url: http://cnds.eecs.jacobs-university.de/courses/nds-2010/kuryla-rpl.pdf. Letzter Zugriff: 09.06.2017.

[Pa16] Parasuram, Aishwarya: An Analysis of the RPL Routing Standard for Low Power and Lossy Networks. Masterarbeit, University of California, Berkeley, 2016.

[PBRD03] Perkins, C.; Belding-Royer, E.; Das, S.: , Ad Hoc On-Demand Distance Vector (AODV) Routing, 2003.

[PR99] Perkins, C. E.; Royer, E. M.: Ad-hoc on-demand distance vector routing. In: Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on. S. 90–100, Feb 1999.

[RSA12] Radoi, Ion Emilian; Shenoy, Aditi; Arvind, D K: Evaluation of Routing Protocols for Internet-Enabled Wireless Sensor Networks. In (Krstic, Dragana ; Borcoci, Eugen, Hrsg.): Proceedings of ICWMC 2012 : The Eighth International Conference on Wireless and Mobile Communications. IARIA, S. 56–61, 6 2012.

[Wi12] Winter, T.; Thubert, A. B. P.; Clausen, T.; Hui, J.; Kelsey, R.; Levis, P.; Pister, K.; Struik, R.; Vasseur, J.: RPL: IPv6 Routing Protocol for Low Power and Lossy Networks. IETF ROLL WG, Tech. Rep, 2012.

[ZL14] Zhang, Tao; Li, Xianfeng: Evaluating and Analyzing the Performance of RPL in Contiki. In: Proceedings of the First International Workshop on Mobile Sensing, Computing and Communication. MSCC '14, ACM, New York, NY, USA, S. 19–24, 2014.

Sicherheit

Sicherheitskonzepte der Hardware für eingebettete Systeme

Daniel Haas[1]

Abstract: Eingebettete Systeme sind wichtige Bausteine des heutigen technologischen Zeitalters. Durch immer Platz sparendere und gleichzeitig leistungsstärkere Hardware steigt die Performance dieser Systeme rasch an, die Sicherheit hingegen entwickelt sich im Vergleich eher langsam. Hackerangriffe auf Spielautomaten in Kasinos beispielsweise verdeutlichen, dass die fehlende Sicherheit ein finanzielles Risiko bedeutet. Da die Wichtigkeit des Themas Sicherheit immer prävalenter wird, ist die treibende Kraft für sichere Systeme möglichst viele Wissenschaftler auf diesem Gebiet zu beschäftigen. Dieses Paper gibt einen umfassenden Überblick über das Thema Sicherheit in eingebetteten Systemen, vom Motiv des Angreifers, zu den verschiedenen Attacken (z.B Hardware Trojaner, Side Channel Analysis) bis hin zu den aktuellsten Gegenmaßnahmen (z.B. Zufallszahlengeneratoren, Physical Unclonable Functions). Dieser wissenschaftliche Text soll zum Einstieg für Interessierte aus dem Bereich Sicherheit oder Embedded in das Themengebiet dienen und einen Anhaltspunkt geben, in welchen Bereichen eventuell weitergeforscht werden kann.

Keywords: IT-Sicherheit; Eingebettete Systeme; Mikrokontroller; Hardware; Überblick; Zufallszahlengenerator

1 Einleitung

Eingebettete Systeme werden immer wichtiger und sind immer häufiger im Einsatz. Das „Internet of Things", indem die Welt zunehmend von intelligenten Gegenständen ergänzt wird, ist die aktuellste Strömung, die dies verdeutlicht. Jedoch auch im Automobilbereich wird eine rapide steigende Anzahl an System on a Chip (SoCs) verbaut. Hieran kann erkannt werden, dass eine Fehlfunktion ein großes Sicherheitsrisiko bedeuten kann, wenn beispielsweise ein automatischer Bremsvorgang, aufgrund eines kompromittierten Steuergeräts, versagt. Auch Ampeln werden von SoCs gesteuert und wurden bereits manipuliert. Da Hersteller von Mikrochips, auch integrierte Schaltkreise (IC) genannt, lange nicht mehr das gesamte Produkt in ihrem eigenen Unternehmen produzieren, sondern Teilschritte an Dritte auslagern, wird der Herstellungsprozess schwieriger zu überwachen. Um die einwandfreie Sicherheit eines ICs zu garantieren, muss garantiert werden, dass dieser auch so verbaut wird, wie es im Design vorgesehen wurde. Dies ist jedoch, aufgrund von potentiellen Angreifern, nicht sicher vorausgesetzt. Ein Angreifer kann, sofern er in die Produktionskette eingreifen kann, ICs fälschen. Falls der Angreifer ein Glied der Produktionskette ist, kann er die ICs

[1] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538 München, Deutschland daniel.haas@campus.lmu.de

illegal weiter produzieren. Sofern der Angreifer ein konkurrierender Hersteller ist, kann das Intellectual Property (Know How des Chips) gestohlen werden, um möglicherweise sogar einen verbesserten Chip zu entwickeln. Es gilt also sowohl das Eigentum der Hersteller, als auch den einwandfreien Zustand der ICs zu schützen. Sind diese nämlich zum Beispiel gefälscht, kann keine Vorhersage über deren Verhalten getroffen werden. Da die Technologie sich sehr rasch weiterentwickelt und die Hersteller Sicherheit teilweise vernachlässigt haben, ist das Feld der Sicherheit im eingebetteten Bereich wichtig und schnell wachsend. Es gibt zwar hochwertige Papers zu einzelnen Maßnahmen, die gegen oben genannte Gefahren helfen sollen, jedoch keinen umfassenden Überblick. Um den Einstieg für Interessierte zu erleichtern und schnelle Ergebnisse zu ermöglichen, wäre es wichtig einen guten Überblick über die gesamte Thematik zu geben. Dies soll Gegenstand dieses Papers sein, wobei sich auf Sicherheitskonzepte in Hardware fokussiert wird. In Kapitel 2 sollen die wichtigsten Angriffe auf eingebettete Systeme erläutert werden, um zu verstehen, vor welchen Gefahren sich im speziellen geschützt werden muss. In Kapitel 3 sollen präventive und in Kapitel 4 detektive Maßnahmen aufgeführt werden, die den Angriffen entgegenwirken. Schließlich werden in Abschnitt 5 die Ergebnisse zusammengefasst und ein kurzer Ausblick gegeben.

2 Angriffe auf Eingebettete Systeme

Im Folgenden werden die bedeutendsten Angriffe erläutert, wobei zwei Unterscheidungsmerkmale berücksichtigen werden sollten. Zum einen können Angriffe invasiv oder nichtinvasiv sein und zum anderen passiv oder aktiv[AATS17].

2.1 Invasive Angriffe

Bei invasiven Angriffen wird auf den IC physikalisch eingewirkt und somit die Integrität verloren. Invasive Attacken sind auch immer aktiv, da der IC immer strukturell verändert werden muss, um das Angriffsziel zu erreichen.
Das Reverse Engineering ist das bekannteste Beispiel für eine invasive Attacke. Hierbei wird der Chip aus seiner Umgebung komplett herausgelöst, zerlegt und schließlich auf seine Funktionen genauestens studiert. Somit kann Industriespionage betrieben werden und beispielsweise ein IC des Konkurrenten kopiert oder gar verbessert und auf den Markt gebracht werden. Dieser Angriff ist sehr kostenintensiv, da die Hardware zwangsläufig defekt wird beim zerlegen und somit eine Menge Hardware, sowie extrem gutes technisches Wissen auf diesem Gebiet notwendig sind[Kr10].
Ein weiters Beispiel für invasive Angriffe ist der Hardware Trojaner. Hierbei handelt es sich laut [RKK14] um ein absichtliches Verändern, Kontrollieren, Behindern oder Überwachen des Inhalts und der Kommunikation eines IC.
Technisch umgesetzt wird dies durch Hinzufügen, Auslöschen oder Modifizieren von Transistoren. Transistoren sind elektronische Halbleiter-Bauelemente und kleinste Bestandteile von logischen Schaltungen in Hardware. Sogar das Ändern der Dotierungspolarität von

existierenden Transistoren wurde schon durchgeführt[Ji15]. Diese Art der Manipulation ist extrem schwer zu detektieren, da alle Transistorschaltungen in ihrer Schaltungslogik gleich bleiben. Nur das Material der Transistoren wird modifiziert. Potentielle Angreifer sind dabei alle Parteien, die im Herstellungsprozess involviert sind.

2.2 Nichtinvasive Angriffe

Bei nichtinvasiven Angriffen wird der IC in seiner Struktur nicht geändert. Ziel ist hier generell sensible Daten aus dem IC auszulesen. Zum Beispiel der Encryption Key bei einem symmetrischen Verschlüsselungsverfahren kann ausgelesen werden. Somit kann die kryptographische Sicherheitsbarriere eines Chips, dank des gewonnenen Geheimnisses, umgangen werden. Es gibt zwei verschiedene Arten von Attacken: Das Fault Injection und Side-Channel-Attacken[Kr10].
Das Fault Injection wird auch semi-invasiv genannt, weil es temporär aktiv in den IC eingreift, diesen aber nicht zerstört oder dauerhaft Transistoren löscht oder hinzufügt. Im Gegensatz zu Side-Channel Attacken kann tatsächlich aktiv in den Chip eingegriffen werden. Die Strategie ist es ein Fehlverhalten beim Verschlüsselungsprozess auszulösen und dieses zu nutzen, um an ansonsten nicht zugängliche Informationen zu gelangen[Kr10]. Ein Beispiel hierfür ist das schon genannte Manipulieren von Spielautomaten. Hier kann durch ein abgestrahltes Signal von beispielsweise einer Spule der SoC des Automaten so beeinflusst werden, dass dieser den Programmablauf für Auszahlung ausführt. Wird das Signal unterbrochen, so befindet sich der Automat wieder in seinem ursprünglichen funktionierenden Zustand. Dieses Fehlverhalten kann durch 4 verschiedene Methoden ausgelöst werden[AATS17]:

- Variation der Versorgungsspannung: Ein Fehlverhalten im IC kann hier durch eine Unter- bzw. Überversorgung oder durch Spannungsspitzen provoziert werden.
- Variation der externen Taktzeiten: Hier kann die Einheit schneller oder langsamer getaktet werden oder aber ein Glitch (sehr kurzer Puls) kann verursacht werden.
- Temperaturvariation
- Elektromagnetische Pulse: Hierbei wird ein elektromagnetisches Feld in der Nähe des IC erzeugt, was zu einem Fehlverhalten führen kann.

Um solche marginalen Schwachstellen gezielt auszunutzen, braucht es teures Spezialequipment und großes Fachwissen[Kr10].

Side-Channel-Angriffe nutzen, ähnlich wie der vorhergehende Angriff, physikalische Eigenschaften des IC, um an kryptographische Informationen zu gelangen. Unterschiedlich ist jedoch, dass bei Side-Channel Attacken keinerlei Eingriff ins laufende System passiert und somit die Funktion des IC nicht beeinflusst wird. Folglich wird dieser Angriff als passiv

kategorisiert. Hierbei unterscheidet man vier verschiedene Ansätze, die Zugang zu den geheimen Daten verschaffen können[AATS17].

- Timing Attacken: Es wird die Zeit gemessen, die verschiedene kryptographische Operationen brauchen, um mit diesem Wissen den kryptographischen Schlüssel zu erlangen.
- Power Analyse: Hierbei werden Stromverbrauch-Traces des IC gelesen und anhand des verbrauchten Stroms einer kryptographischen Operation Aussagen über deren Inhalt geschlussfolgert.
- Elektromagnetische Attacken: Es werden hier elektromagnetische Wellen nahe des Geräts aufgefangen und analysiert.
- Akustische Attacken: Bei kryptographischen Operationen entstandene akustische Geräusche werden genauestens analysiert.

Diese Art von Angriff ist um einiges leichter zu realisieren als die vorher genannten Angriffstypen und benötigt um ein Vielfaches weniger an Wissen und Hardware[Kr10].

3 Prävention

Nun da ein Verständnis für die Seite der Gefahren und Risiken geschaffen wurde, wird in den nächsten zwei Kapiteln der essentielle Teil dieser Arbeit erläutert: die Maßnahmen gegen Angriffe auf eingebettete Systeme. Diese werden in, die in der Informationssicherheit geläufigen Kategorien, Prävention und Detektion eingeteilt. Laut Konvention fehlt noch die Kategorie Reaktion, wobei im eingebetteten Bereich dies quasi entfällt. Da ICs oft sehr simple Systeme sind, muss versucht werden schon in der Produktion jegliche zukünftige Angriffe unmöglich oder möglichst schwer zu gestalten. Dies entspricht der Prävention.

3.1 Schutz auf Transistorebene

Anpassungen auf Transistorebene können das ungewollte Preisgeben von kritischen Informationen an Angreifer möglichst gering halten und stellen damit einen interessanten Sicherheitsansatz dar. Transistoren repräsentieren die kleinste Einheit im IC und können somit ohne Änderungen an der Software vorzunehmen, sondern lediglich durch intelligente Nutzung von Hardwarekomponenten, einen großen Schritt zu einem sicheren System bedeuten.

3.1.1 Masking und Hiding

Die folgenden beiden Methoden eignen sich, um Side-Channel Attacken und Fault Injection zu verhindern.
Beim Masking ist die Idee, dass der Stromverbrauch einer kryptographischen Operation nicht mit den tatsächlichen unverschlüsselten Daten, die verarbeitet werden, korrelieren sollen. Dies kann auf Gate-Ebene erzielt werden, indem Schaltkreise erzeugt werden, bei denen keine Verbindung ein Ergebnis transportiert, das tatsächlich mit einem Zwischenergebnis eines Verschlüsselungsalgorithmus korreliert. Es wird also eine sichere Masking-Vorlage für einen bestimmten Schaltkreis eines Algorithmus entwickelt und später automatisiert von einem Computerprogramm zu einem Gebilde von maskierten Gates konvertiert[FG05]. Folglich können abgefangene Informationen der Gates keine Aussage über geheime Schlüssel oder andere Daten preisgeben.
Bei der Methode des Hiding ist die Idee, dass verschiedene Operationen immer denselben Stromverbrauch generieren und somit die Differential Power Analysis, also das Analysieren des ICs aufgrund der Variation im Stromverbrauch, keine sinnvollen Ergebnisse erzielt. Die effektivste Hiding-Variante nutzt ein Dual-Rail Precharge Logic Gate (DPA). Diese berechnen immer zusätzlich zum Output auch dessen Komplement, was dazu führt, dass jeden Taktzyklus derselbe Stromverbrauch zu messen ist.[AATS17].

3.1.2 Camouflaging und Polymorphe Gates

Beim Camouflaging geht es darum, die Gates zu „tarnen“. Es soll Reverse Engineering und folglich Intellectual Property-Piraterie erschwert werden. Bei der IP-Piraterie werden neben dem Reverse Engineerung auch aus hochauflösenden Fotos des Chips die verschiedenen Gatetypen erkannt und können somit nachgebaut werden. Dies soll verhindert werden, indem die Gates, durch verschiedene Mittel, im Aussehen gleich gemacht werden. Ein NAND- und ein NOR-Gate sehen beispielsweise im Normalfall unterschiedlich aus und deshalb können auch ihre Funktionen, falls identifiziert, extrahiert werden. Eine Möglichkeit dafür ist es, programmierbare Standardgates zu benutzen und sie erst nachdem sie verbaut sind zu konfigurieren. Es können auch sogenannte „Dummy-Kontakte“verwendet werden, die eine Lücke in der Verbindung haben und somit eine Verbindung nur antäuschen[RKK14].
Ein spezieller Graphin-Transistor, ein sogenannter „Graphene SymFET“, bietet aufgrund seiner besonderen elektrischen Eigenschaften eine weitere Methode der Tarnung. Diese Transistoren bieten die Möglichkeit sogenannte „polymorphe Gates“ zu implementieren. Ein solches Gate kann seine logische Eigenschaft, aufgrund äußerer Zustände, ändern und somit einen anderen logischen Baustein technisch realisieren. Ein Beispiel wäre hier ein Gate, das bei 3.3 Volt Versorgungsspannung ein AND-Gate repräsentiert, jedoch bei einer reduzierten Spannung von 1.5 Volt die Funktion eines OR-Gates annimmt. Dies ist vom äußeren Erscheinungsbild des Gates nicht abzuleiten und bietet somit guten Schutz vor IP-Piraterie.[Bi16].

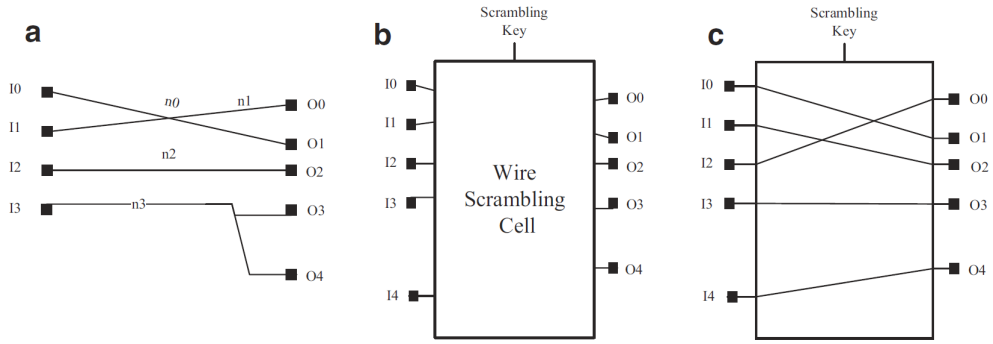


Abb. 1: (a) Ursprüngliche Verbindungen (b) Symbolische Sicht auf WS-Zelle (c) Falsche Topologie wegen falschem Schlüssel [ZJ16]

3.1.3 Verschleierung

Das Ziel bei der Verschleierung ist es, die Funktionalität sowie die Implementierung eines Designs zu verbergen und somit Reverse Engineering zu verhindern. Dafür können beispielsweise zusätzlich spezielle Gates (XOR/XNOR) oder Speicherelemente zum bereits funktionierenden Design hinzugefügt werden. Ein solches verschleiertes Design funktioniert dann nur wie gewünscht, wenn die richtigen Werte an diese Gates und Speicherelemente weitergegeben werden.
Ein weiterer Ansatz sieht vor, den Endlichen Automaten (EA) eines Designs zu manipulieren. Die Verschleierung kann dabei durch Hinzufügen von Zuständen erfolgen. Diese können z.B. andere Zustände replizieren. Auch Übergänge können hinzugefügt werden. Es gibt dabei die Möglichkeit ungültige Übergänge zwischen Zuständen oder Übergängen hinzuzufügen, die in keinen neuen Zustand führen, sogenannte „Black Hole States“. Alle diese Designmöglichkeiten haben gemeinsam, dass die Logik nur funktioniert, wenn ein richtiger Schlüssel genutzt wird. Wird ein falscher Schlüssel benutzt, steckt der IC in Black Hole States oder in invaliden Zuständen oder Übergängen fest [RKK14].
Zusätzlich kann noch eine andere Art der Verschleierung genutzt werden, das sogenannte „Wire Scrambling“. Hierbei werden speziell standardisierte Wire Scrambling-Zellen (WS-Zellen) entwickelt und in das Design des IC eingefügt. Die Eingänge dieser Zellen werden auf der physikalischen Ebene „durcheinandergewürfelt“, je nachdem welcher WS-Schlüssel auf diese Zelle angewandt wird. Das heißt im konkreten Fall, dass alle Eingänge theoretisch auf alle Ausgänge der WS-Zelle physikalisch verdrahtet werden können, dies aber außerhalb der Zelle nicht erkennbar ist, wie auf Abbildung 1 veranschaulicht wird. Nur die Kenntnis des WS-Schlüssels kann ein derartiges Design aufdecken. Technisch realisiert werden können diese WS-Zellen durch Multiplexer oder Durchlasstransistoren. Experimente zeigen, dass durch diese Technik Reverse Engineering stark erschwert wird, wobei die Kosten an zusätzlicher Fläche, Stromverbrauch und Gesamtdrahtverbrauch zu vernachlässigen sind[ZJ16].

3.1.4 Split Manufacturing

Überproduktion von ICs, IP-Piraterie und Fälschungen können den Hersteller eines ICs finanziell extrem schwächen. Diese Probleme ergeben sich oft aus der mangelnden Übersicht über die Produktionskette. Oft wird das gesamte Design an eine dritte Partei zur Produktion weitergegeben, in der Hoffnung, dass diese rechtmäßig damit umgeht. Um sich nicht mehr auf den guten Glauben verlassen zu müssen, wurde der Ansatz des „Split Manufacturing“ entwickelt. Hierbei handelt es sich um eine Maßnahme auf organisatorischer Ebene. Die Idee dabei ist, dass nicht mehr das gesamte Design bei einer fremden Partei produziert wird, sondern ein Teil, das „Front End of the Line (FEOL)“, das technologisch sehr aufwendig zu produzieren ist, in einer schlecht überwachten Umgebung hergestellt wird. Der zweite Teil, das „Back End of the Line (BEOL)“, wird dann in den eigenen Produktionsstätten gefertigt. Durch diese Teilung ist es für diese Parteien nicht mehr möglich das vollständige IP zu extrahieren[Ji15].

3.2 Kryptographischer Schutz durch Hardware

Um ein eingebettetes System grundsätzlich sicher gegen triviale Angriffe zu gestalten, müssen starke kryptografische Verschlüsselungsalgorithmen die verwendeten Daten verschlüsseln. Dies gestaltet sich speziell bei ICs jedoch, aufgrund der beschränkten Rechen- und Speicherkapazität, besonders anspruchsvoll . Trotzdem wurden starke Maßnahmen gefunden, einen ausreichenden kryptographischen Schutz zu erzielen. In diesem Kapitel sollen diese Maßnahmen und ihre Prinzipien analysiert werden.

3.2.1 Zufallszahlengenerator

Bei der Implementierung von kryptographischen Algorithmen ist das wichtigste Element der Zufall. Ein Zufallszahlengenerator (Random Number Generator RNG) hat die Aufgabe Sequenzen von zufälligen Zahlenwerten zu erzeugen. Diese Sequenzen sollen im besten Falle unvorhersagbar sein. Folglich hängt die Sicherheit eines kryptographisches Systems stark von der Effektivität des RNGs ab. Es lassen sich grundsätzlich zwei Arten von RNGs unterscheiden [AATS17]:

- Wahrer RNG (True RNG: TRNG); beruht auf stochastischen physikalischen Prozessen
- Pseudo RNG (PRNG); deterministischer endlicher Automat, der innerhalb einer Periode so scheint, als ob er wahre Zufallswerte generiert

Beide Konzepte sind wichtige Bausteine einer sicheren Kryptographie und werden deshalb im Folgenden erläutert werden.
Ein PRNG kann als Device angesehen werden, das für einen unendlichen Zeitraum eine

endliche Zufallssequenz ausgibt. Diese Zufallssequenz ist in seinem Speicher gespeichert oder durch verschiedene Berechnungen erzeugt[AATS17].
Ein Beispiel für einen PRNG zeigt [Tk02], mit einem PRNG, der in Hardware realisiert wurde. Dieser besteht im Prinzip aus zwei unterschiedlich strukturierten Automaten und jeweils einem frei schwingenden Oszillator. Die beiden Automaten sind dabei ein linear rückgekoppeltes Schieberegister (LFSR) und ein zelluläres Automaten-Schieberegister (CASR). Dies sind zwei relativ simple Bausteine, die einzeln sehr regelmäßige, also schlechte Ergebnisse erzielen. Abbildung 2 zeigt jedoch, dass gemeinsam ein relativ artefaktfreies Rauschen an Zuständen erzielt werden kann. Zusätzlich muss noch sogenantes „Site- und Time Spacing“ verwendet werden, um die Korrelation der Bits, der beiden Automaten, zu verhindern. Die ausgegebenen Werte bestehen dann sogar Marsaglias DIEHARD-Tests, eine oft referenzierte Sammlung an statistischen Tests, die die Qualität eines RNG messen[Tk02].

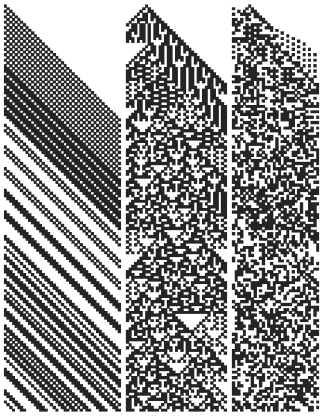


Abb. 2: Zustand-Zeit-Diagramm des LFSR, CASR und Kombination aus beiden [Tk02]

Um hingegen einen TRNG zu entwerfen, gibt es drei Quellen laut [AATS17]:

- Chaotische Schaltkreise
- hochgradig flimmernde Oszillatoren(High-jitter oscillators)
- Schaltkreise, die andere stochastische physikalische Prozesse messen

Chaotische Schaltkreise sind Schaltkreise, in denen sich Ströme und Spannungen über die Zeit hinweg ändern. Diese Änderungen entsprechen einer deterministischen Evolutionsregel, die spezielle mathematische Voraussetzungen erfüllt. Die Zeitevolution der Ströme und Spannungen wird dabei als Zustandsevolution eines nichtlinearen dynamischen Systems, das chaotisches Verhalten vorweist, beschrieben. Bekannte Beispiele für chaotische Systeme sind das Lorenzsystem, die logistische Gleichung, die Henon-Abbildung und das Doppelpendel. Diese Systeme können in einer Schaltung implementiert und dann gemessen werden[AATS17].

Bei hochgradig flimmernden Oszillatoren wird das Abweichen der Ausgabe eines Oszillators von seiner eigentlichen Periodizität, das sogenannte „Flimmern“ ausgenutzt. Dieses Flimmern verursacht Ungewissheit bei den Übergangszeiten von high-low und low-high des Oszillators. Ein TRNG nutzt typischerweise ein Verfahren, das auf mehreren freilaufenden, miteinander interagierenden Oszillatoren, basiert. Außerdem sollten diese eine möglichst hohe relative Differenz zwischen ihren Nominalfrequenzen haben. Besonders an diesem Ansatz ist, dass einige Implementierungen in vollkommen digitalen Prozessen genutzt werden können, im Gegensatz zu anderen Ansätzen[AATS17].
Der dritte Ansatz nutzt zufällige physikalische Phänomene, wie radioaktiven Zerfall, Detektion von Photonen, sowie verschiedene Quellen von elektronischem Rauschen in Halbleitergeräten, wie z.B. thermisches Rauschen, Diffusion oder Stoßentladung[AATS17].

3.2.2 Post-processing

Auch wenn die Quelle eines TRNG vollkommen zufällig ist, werden die ausgegebenen Werte negativ von statistischer Verzerrung und vom Fehlverhalten des Speichers beeinflusst. Fehler können beispielsweise bei der A/D-Wandlung passieren. Um diese Fehler zu glätten, wird das Post-Processing eingesetzt, das vollkommen digital abläuft. Dabei können laut [AATS17] zwei Methoden unterschieden werden.
Bei der Kompression ist die Idee, das Datenwort des TRNG auf weniger Bits zu reduzieren und somit die Entropie zu vergrößern. Hierbei kann nochmal zwischen verlustfreien und verlustreichen Kompressoren unterschieden werden. Da verlustreiche Kompressoren weniger komplex in Hardware abzubilden sind, werden diese meist bevorzugt[AATS17].
Bei der Diffusion oder Konfusion sollen alle verbleibenden statistischen Defekte, durch sorgfältiges Vermischen und Verschlüsseln der generierten Sequenz des TRNG, maskiert werden. Die einfachste Lösung hierfür ist es, eine Bit-für-Bit XOR-Operation anzuwenden. Dabei werden die komprimierte Sequenz und eine, von einem PRNG generierte, Sequenz als Input gewählt[AATS17].

4 Detektion

Aufgrund der Schwierigkeit, die gesamte Fertigungskette ausreichend zu überwachen, ist es zusätzlich wichtig, manipulierte ICs zu finden und dementsprechend den Produktionsprozess zusätzlich zu überprüfen. Diese können bspw. einen Hardware Trojaner enthalten.

4.1 Physical Unclonable Functions

Das Konzept von Physical Unclonable Funktions (PUF) nutzt die Varianz zwischen verschiedenen Exemplaren des gleichen Chiptyps aus. Beispiele für diese Unterschiede sind interne Verzögerungen und Timing-Varianzen. Diese entstehen zwangsläufig bei der Produktion

und sind praktisch nicht nachimplementierbar[Ji15].
Eine PUF ordnet einer Gruppe von Challenges eine Gruppe von Antworten zu, die auf diesen minimalen Produktionsvarianzen basieren. Diese Paare sollen möglichst zufällig im Hinblick auf den Input, aber reproduzierbar sein. Außerdem sollen die Eigenheiten des Chips einzigartig sein und die PUF sollte mit zusätzlichen Sicherheitsmaßnahmen geschützt werden[Ji15]. Die Evaluation kann auch nur mit dem eigenen Chip erfolgen und kann zur Gerätauthentifikation verwendet werden. Dazu speichert derjenige, der das Device überprüfen will, eine Reihe an Challenge-Response-Paaren in einer Datenbank ab. Will er in Zukunft das Gerät prüfen, nimmt er eines dieser Paare, führt die Challenge erneut durch und vergleicht, ob die selben Werte generiert wurden. Falls ja, ist das Gerät authentifiziert. Um vor Man-In-the-middle Angriffen zu schützen, darf jedes Paar nur einmal verwendet werden[G.07].
Dieser Prozess kann dazu verwendet werden, Überproduktion oder Hardwaretrojaner aufzudecken. Bei der Überproduktion würde die Authentifikation fehlschlagen, da diese Chips nicht offiziell registriert sind und somit keine Challenge-Response-Paare vorhanden wären. Bei Hardwaretrojanern würde das einzigartige Profil verändert und somit würden die Challenge-Response-Paare nicht mehr übereinstimmen.
Eine interessante Möglichkeit, die PUF zusätzlich vor physischen Attacken (Reverse Engineering) zu schützen, ist es, diese in einer manipulationsüberwachten Umgebung zu platzieren, wie es beim IBM 4758 Prozessor der Fall ist. Dies ist jedoch relativ teuer und muss kontinuierlich mit Strom versorgt werden [G.07].
Ein weiterer Ansatz, um die Sicherheit der PUF zu erhöhen, ist es, statt den standardmäßigen CMOS-Transistoren einen hybriden PUF-Schaltkreis aus CMOS- und sogenannten „Memristoren (Memory Resistor)“ aufzubauen. Durch die elektrischen Eigenschaften der Memristoren, die das Austreten von Side-Channel-Informationen verhindern, wird eine der bedeutendsten Attacken erschwert. Dieser Angriff ist das Modellbilden eines PUFs mittels maschinellem Lernen[Ma15]. Ein solcher Angriff kann dem Reverse Engineering zugeordnet werden. Anstatt den Chip in seine Einzelteile zu zerlegen, wird hier ein genaues Verhaltensmodell gebildet. Dazu werden Side-Channel-Informationen gespeichert und mit Hilfe von maschinellem Lernen, auf Basis einer großen erzeugten Datenmenge, ein PUF-Profil gebildet. Schließlich kann die PUF, anhand diesen Modells, mit anderen Transistoren nachgebildet werden.

4.2 Watermarking und Fingerprinting

Watermarking und Fingerprinting sollen vor IP-Piraterie und Überproduktion von Chips schützen. Es sollen kompromittierte Exemplare detektiert werden können, nachdem vorher spezielle Modifikationen am Chip vorgenommen wurden, die diesen identifizieren.
Beim Watermarking werden dem IC aktiv Elemente hinzugefügt, die unverkennbar als Wasserzeichen wiederzuerkennen sind. Hierfür können beispielsweise einem endlichen Automaten sogenannte „Black-Hole Zustände“ hinzugefügt werden. Ein weiteres Beispiel wäre es, gewisse Zwangsbedingungen bei der Graphpartitionierung als Wasserzeichen zu

definieren. Es sollte auf jeden Fall extrem schwer zu entfernen, eindeutig und für alle vorgesehen Designs anwendbar sein[RKK14].
Fingerprinting wird in Verbindung mit Watermarking genutzt. Um Piraterie aufzudecken, wird die Signatur des IC-Käufers und das Wasserzeichen des Designers in den Chip integriert. Will man nun den Status des IC prüfen, werden diese beiden Merkmale abgeprüft. Das Wasserzeichen stellt klar, wer der rechtmäßige Designer des Chips ist und die Signatur, wer der rechtmäßige Besitzer ist. Für diesen Vorgang können physikalische Fingerabdrücke von Strom-, Timing und Thermischen Merkmalen als Eingabevektor für eine Challenge genutzt werden[RKK14]. Auch PUFs können für Fingerprinting verwendet werden (siehe Kapitel 4.1).

4.3 Hardware Überwachung

Diese Art von Maßnahmen soll vor aktiven Angriffen auf das System schützen (Fault Injection, Hardwaretrojaner, Reverse Engineering). Bei der Hardware Überwachung kann man grundsätzlich, laut [RKK14], zwischen passivem und aktivem Überwachen unterscheiden.

Die Idee bei der *aktiven* Überwachung ist es, das System permanent zu überwachen, sowohl interne als auch externe Zustände. Diese werden mit gespeicherten Referenzwerten, die einen Normalzustand attestierten, verglichen. Ein Controller überwacht diese Zustände und erkennt so, wenn es zu abnormalem Verhalten kommt. Ist dieses detektiert, so muss das System agil handeln. Eine mögliche Reaktion wäre es, von einem normalen Operationsmodus in einen abgesicherten Modus überzugehen. Hier können einige vertrauliche Daten automatisch gelöscht werden und einige Funktionen blockiert werden (z.B. externe Kommunikation). Oft ist es sicherer Daten zu verlieren als diese an Angreifer preiszugeben[Kr10].
Im Kapitel Watermarking und Fingerprinting wurden bereits zwei Varianten der *passiven* Überwachung besprochen. Zusätzlich kann laut [RKK14] eine weitere Maßnahme zur Überwachung getroffen werden. Durch den Verbau von Alterungssensoren im IC, können Fälschungen und der Verkauf von gebrauchten Chips verhindert werden. Gewisse Umstände sind dafür zuständig, die Lebensspanne eines IC festzulegen. Darunter sind zu nennen negative Reaktionen auf Temperatur und elektromagnetische Migration. Werden diese Faktoren nun gemessen, so kann eine Aussage darüber getroffen werden, wie alt in etwa ein Chip ist und Differenzen zum angegebenen Alter aufgedeckt werden.

5 Schlussfolgerungen

In diesem Paper wurde ein Überblick über den aktuellen Stand der Informationssicherheit im eingebetteten Bereich gegeben, mit besonderem Fokus auf Hardwarekonzepte. Es sollte für den Leser möglich sein, ein gutes erstes Verständnis für die Thematik zu erlangen.
Jedes System sollte sowohl präventiv als auch detektiv geschützt werden. Bei der Prävention sollte mindestens eine Art des Schutzes auf Transistorebene implementiert werden. Hier sind

aus Kosten- und Effektivitätsgründen materialspezifische Maßnahmen, wie Polymorphe Gates und Wirescrambling, besonders interessant. Split Manufacturing ist zusätzlich eine sehr sinnvolle Maßnahme, um die Produktionskette besser überwachen und Fälschungen erschweren zu können. Kryptographischer Schutz ist der zweite essentielle Schutzfaktor auf präventiver Ebenen. Hier sollte im besten Falle ein TRNG verwendet werden, wie jedoch gezeigt, erzielen auch spezielle PRNG sehr respektable Ergebnisse.
Auf detektiver Ebene sind PUFs zu empfehlen, da diese kostengünstig und leicht zu implementieren sind.
Mit einer wachsenden Gruppe an Experten für diesen Bereich wird es bald weniger Sicherheitslücken geben. Somit werden auch fairere Bedingungen für ehrliche Hersteller von ICs, sowie bedenkenlose Sicherheit für den fachfremden Endnutzer, ermöglicht.

Literaturverzeichnis

[AATS17] Acosta, Antonio J.; Addabbo, Tommaso; Tena-Sanchez, Erica: Embedded electronic circuits for cryptography, hardware security and true random number generation: An overview. International Journal of Circuit Theory and Applications, 45(2):145–169, 2017.

[Bi16] Bi, Yu; Shamsi, Kaveh; Yuan, Jiann-Shiun; Gaillardon, Pierre-Emmanuel; Micheli, Giovanni De; Yin, Xunzhao; Hu, X. Sharon; Niemier, Michael; Jin, Yier: Emerging Technology-Based Design of Primitives for Hardware Security. ACM Journal on Emerging Technologies in Computing Systems, 13(1):8–11, 2016.

[FG05] Fischer, Wieland; Gammel, Berndt M.: Masking at Gate Level in the Presence of Glitches: Masking. In (Rao, Josyula Ramachandra, Hrsg.): 7th international workshop Edinburgh UK August 29 - September 1 2005 ; proceedings: Literaturangaben, Jgg. 3659 in Lecture Notes in Computer Science, S. 3. Springer, Berlin, 2005.

[G.07] G. Edward Suh, Srinivas Devadas: Physical Unclonable Functions for Device Authentication and Secret Key Generation. ACM, New York, NY, 2007.

[Ji15] Jin, Yier: Introduction to Hardware Security. Electronics, 4(4):763–784, 2015.

[Kr10] Krief, Francine: Communicating embedded systems: 5. Hardware Security in Embedded Systems. ISTE, London, 1. publ. Auflage, 2010.

[Ma15] Mathew, Jimson; Chakraborty, Rajat Subhra; Sahoo, Durga Prasad; Yang, Yuanfan; Pradhan, Dhiraj K.: A Novel Memristor-Based Hardware Security Primitive. ACM Transactions on Embedded Computing Systems, 14(3):1–20, 2015.

[RKK14] Rostami, Masoud; Koushanfar, Farinaz; Karri, Ramesh: A Primer on Hardware Security: Models, Methods, and Metrics. Proceedings of the IEEE, 102(8):1283–1295, 2014.

[Tk02] Tkacik, Thomas E.: A Hardware Random Number Generator. In (Kaliski, Burton S., Hrsg.): 4th international workshop Redwood Shores CA USA August 13 - 15 2002 ; revised papers: Includes bibliographical references and index, Jgg. 2523 in Lecture Notes in Computer Science, S. 450–453. Springer, Berlin u.a., 2002.

[ZJ16] Zamanzadeh, S.; Jahanian, A.: Higher security of ASIC fabrication process against reverse engineering attack using automatic netlist encryption methodology. Microprocessors and Microsystems, 42:1–9, 2016.

Security concepts on layer 2, 3 and 4 of embedded communication systems in the Internet of Things

Christian Ziegner[1]

Abstract: Extensive real-world presence of IoT networks require a high level of security. As such 6LoWPAN networks only have limited resources, it is necessary to adapt existing security mechanisms for traditional IP traffic to meet the requirements of resource-constrained networks. In this paper we present an overview of established security methods, i.e. link-layer security, IPsec and DTLS, and further provide 6LoWPAN header compression schemes for IPsec and DTLS. We shortly compare IEEE 802.15.4 link-layer security to IPsec and show that using both approaches concurrently is preferable in terms of security. Generally, we illustrate that adjusting and using existing security mechanisms is a feasible option for IoT networks.

Keywords: Internet of Things; 6LoWPAN; CoAP; Security; IPsec; DTLS; IEEE 802.15.4 link-layer security; Header Compression

1 Introduction

In recent years, the Internet of Things (IoT) has become a hot topic that will have significant impact on the evolution of the future Internet [Tan+10]. IoT interconnects so called smart objects, i.e. small embedded devices with context specific sensing and actuating capabilities, with traditional computers and humans [Vas+10]. As the presence, popularity and scale of such wireless IP-based IoT networks grows, also the likelihood and impact of potential network attacks increases. This, in turn, raises the necessity of investigating methods to secure IoT traffic. Since IoT devices communicate over the IP protocol and there already are several widely used methods to protect IP traffic, such as IPsec and TLS, securing the traffic may seem to be a rather simple task. However, IoT networks, also referred to as IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), only have limited resources. 6LoWPAN compression mechanisms are applied to the packets traversing the network in order to reduce their size. The mentioned resource constraints require the development of new compression schemes to be able to integrate heavyweight, complex security protocols into 6LoWPAN communication.

[1] Technical University of Munich, Department of Informatics, Boltzmannstraße, 85748 Garching, Germany
christian.ziegner@tum.de

In this paper, we present and briefly analyze security concepts and their corresponding 6LoWPAN compression mechanisms. We begin with background knowledge about 6LoWPAN and the Constrained Application Protocol (CoAP). Section 3 provides an overview of security techniques for different layers optimized for IoT, namely IEEE 802.15.4 link-layer security for the link layer, IPsec for the network layer and DTLS for the transport layer. In the next section, we shortly evaluate and compare link-layer security and IPsec. The last section concludes this document.

2 Background

In this chapter, we give background information about the underlying technologies relevant for the understanding of the rest of the paper. We present basics of the concept of 6LoWPAN and the CoAP. More detailed information about those concepts can be found in [She+11] and [Bor+12]. For further, more extensive background information about IPv6, the IoT and IEEE 802.15.4 networks, we recommend having a closer look at [Dee98], [Tan+10] and [IEE16].

2.1 IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN)

The 6LoWPAN standard [Hui+11] specifies the communication and compression of IPv6 packets over IEEE 802.15.4-based networks. Those so called low-rate wireless personal area networks generally comprise devices which are limited in terms of computation power, bit rate and transmission range. A typical application is a wireless sensor network [Mai+11]. 6LoWPAN adopts the layer 1 and 2 specifications of IEEE 802.15.4 [IEE16] and further specifies the network layer standard. Additionally, it introduces an extra intermediate layer between link and network layer which takes care of header compression and fragmentation [Ma+08] [Raz+14].

In 6LoWPAN networks it is necessary to reduce the data to be transmitted as the maximum transmission unit (MTU) is very limited. The maximum payload at the data link layer is about 80 bytes of which 40 bytes would be used for uncompressed IP headers [Mon+07]. This header size can be reduced to 2 bytes in single-hop networks by applying the IP header compression mechanism LOWPAN_IPHC (subsequently referred to as IPHC) defined in the 6LoWPAN standard. The IPHC encoding is shown in figure 1. If the next header field is set to 1, it indicates that the next header following the compressed IPv6 header is also compressed using the next header compression mechanism LOWPAN_NHC (subsequently referred to as NHC) which is defined for IP extension headers and the UDP header in the current standard [Hui+11]. The size of an NHC header is between 1 and multiple octets of which the first bits identify the next header type and the remaining bits encode header information. The general NHC format is illustrated in figure 2. Header compression is only applied inside 6LoWPAN networks which is connected to outside nodes through a so called boarder router.

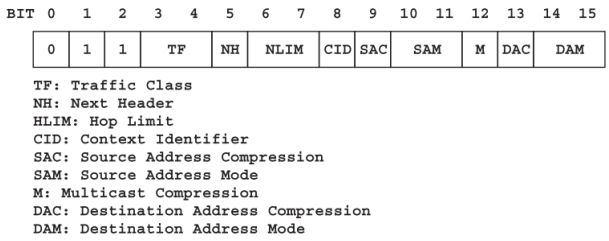


Fig. 1: Basic IPHC encoding [Raz+14].

Fig. 2: General NHC encoding [Raz+14].

2.2 Constrained Application Protocol (CoAP)

As stated in [She+14], CoAP is a web transfer protocol specialized for the communication of constrained nodes in lossy, low-power networks. It uses the unreliable UDP protocol and can be seen as a variation of the extensively used HTTP web protocol optimized for the IoT. It provides a REST interface but is more lightweight and cost-effective compared to HTTP. For securing CoAP traffic Datagram Transport Layer Security (DTLS) is proposed which transforms CoAP into CoAPs, similar to HTTP and HTTPs.

3 Securing Traffic in 6LoWPAN

In the following section, we present security mechanisms that can be applied to secure traffic in 6LoWPAN networks on different layers, namely on layer 2, 3 and 4.

3.1 Data Link Layer

We start with data link layer security, i.e. protecting communication between neighboring nodes. For that, we consider link-layer security as defined in the IEEE 802.15.4 standard [IEE16].

3.1.1 IEEE 802.15.4 link-layer security

According to [Sas+04], IEEE 802.15.4 link-layer security provides access control, message integrity, message confidentiality and replay protection as security principles. Message integrity and access control is achieved by adding a message authentication code (MAC) to

every packet that is transfered. Encrypting the traffic with a pre-shared key and adding a sequence number to the messages ensures confidentiality and replay protection. However, packets that are sent towards or come from nodes outside of the 802.15.4 network cannot be protected by the provided security mechanisms. Also, it does not provide end-to-end (E2E) protection, as traffic is only protected on a hop-by-hop basis. To achieve E2E security, encryption should take place in one of the layers above. More fine-grained information about IEEE 802.15.4 link-layer security can be found in [IEE16].

3.2 Network Layer

In the following subsection we consider the next upper layer, namely the network layer, and have a closer look at corresponding security approaches. We begin with an overview of the standard IPsec protocol suite as it is used in conventional IP networks. Afterwards, we provide a 6LoWPAN IPsec extension that makes IPsec also applicable in IoT networks.

3.2.1 IPsec Overview

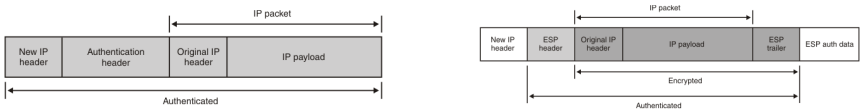
IPsec was designed in conjunction with the new version of the Internet Protocol IPv6 to secure traffic in an end-to-end manner in IP networks. It offers two types of transfer protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP). The IP AH provides integrity and data origin authentication, as well as an optional replay protection feature. It protects the whole IP packet by calculating a message integrity code (MIC) of IP header, AH and IP payload. ESP ensures the same security goals as AH and, in addition to it, offers confidentiality. In other words, ESP also makes use of encryption, but compared to the AH protocol, it operates solely on the IP payload. Further, both protocols offer access control by managing the distribution of cryptographic keys [Ken+05].

The protocols can operate in two different modes, either in transport or in tunnel mode. Transport mode retains the original IP header as is and is most often used in a host-to-host scenario. When using IPsec in tunnel mode, however, the original IP header is also encapsulated by AH or ESP and replaced by a new IP header. The new header contains the source and destination addresses of the security endpoints which might differ from the source and destination addresses of the actual data. The structure of the two protocols in transport and tunnel mode is illustrated in figure 3 and 4 respectively. Since there is notable overhead resulting from the extra header, tunnel mode is not recommended in 6LoWPAN networks. For a more detailed view on the AH and ESP protocols, we recommend having a closer look at the corresponding RFCs [Ken05a] and [Ken05b].

(a) AH securing basic IP packet.

(b) ESP securing basic IP packet.

Fig. 3: AH and ESP protocol in transport mode [Cen].



(a) AH securing basic IP packet.

(b) ESP securing basic IP packet.

Fig. 4: AH and ESP protocol in tunnel mode [Cen].

3.2.2 6LoWPAN/IPsec Extension

In order to make use of IPsec in 6LoWPAN networks, we present appropriate extensions for AH and ESP that are elaborated in [Raz+14]. Therefor, we make use of the next header compression format for IP extension headers (NHC_EH) [Hui+11] mentioned in 2.1 to link the self-defined NHC encodings for AH (NHC_AH) and ESP (NHC_ESP). Figure 5 describes the general NHC_EH. Since 101 and 110 are the only two values left for the extension header ID (EID) that are not defined otherwise, we have to use one of them for our purposes. We choose 101 as the EID indicating that either NHC_AH or NHC_ESP is following. Further, the NH bit has to be set to 1 to denote that the following header is also NHC encoded.

The NHC encoding for AH is shown in figure 6a. The fields of the NHC_AH are defined as follows:

- The first 4 bits represent the NHC ID which is defined as 1101 for AH.
- If $PL = 0$, the payload length field is omitted in AH and can be derived from the SPI. Otherwise, it is carried in-line.
- If $SPI = 0$, the SPI field is elided in AH and the default SPI is used. Otherwise, it is carried in-line.
- If $SH = 0$, only the last 16 bits of the sequence number are displayed in AH. Otherwise, all 32 bits of the sequence number are carried in-line.
- If $NH = 1$, the next header is NHC encoded and the next header field in AH is omitted. Otherwise, it is carried in-line and used to specify the following header.

Figure 7a shows an AH secured IPv6/UDP packet that is maximally compressed, i.e. next

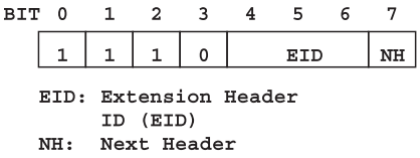
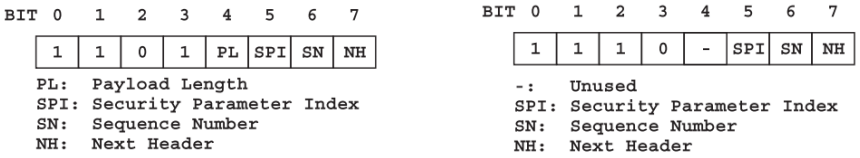


Fig. 5: General NHC_EH encoding [Raz+14].



(a) NHC_AH encoding.

(b) NHC_ESP encoding.

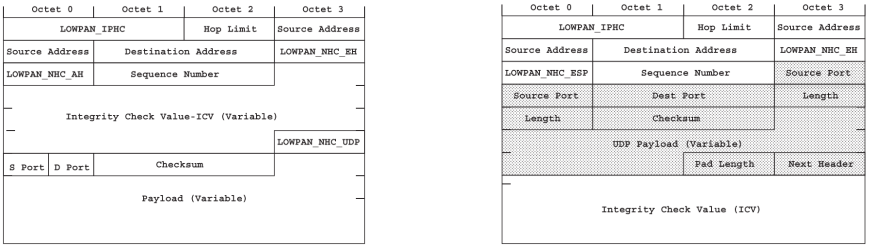
Fig. 6: NHC encodings for AH and ESP [Raz+14].

header field, payload length field and the first 16 bits of the sequence number are omitted. A comparison with the uncompressed AH [Ken05b] clearly illustrates the effect of the compression.

The NHC encoding for ESP is shown in figure 6b. The fields of the NHC_ESP are defined as follows:

- The first 4 bits represent the NHC ID which is defined as 1110 for ESP.
- The next bit is unused since ESP does not have a payload length field. It is still present for consistency reasons.
- If $SPI = 0$, the SPI field is elided in ESP and the default SPI is used. Otherwise, it is carried in-line.
- If $SH = 0$, only the last 16 bits of the sequence number are displayed in ESP. Otherwise, all 32 bits of the sequence number are carried in-line.
- If $NH = 1$, the next header is NHC encoded and the next header field in ESP is omitted. Otherwise, it is carried in-line and used to specify the following header.

Figure 7b shows a compressed and ESP secured IPv6/UDP packet. Note, that the UDP header in the payload is encrypted when using ESP and, therefore, cannot be compressed as in the case of AH.



(a) Compressed and AH protected IP packet using NHC_AH header compression.

(b) Compressed and ESP protected IP packet using NHC_ESP header compression.

Fig. 7: Compressed IP packets using IPsec header compression [Raz+14].

3.3 Transport Layer

The last layer that we focus on in this paper is the transport layer. As described in 2.2, CoAP is the web transfer protocol that is primarily used in IoT networks. Like TLS securing HTTP traffic, DTLS can be used to secure the UDP payload on the transport layer in CoAP traffic. In the following, we shortly describe the basics of DTLS. The overview is kept rather abstract, as the DTLS protocol is very complex and a detailed description would exceed the scope of this document. For a more detailed view on TLS and DTLS, we recommend inspecting [Die+08] and [Res+12]. To be able to use such a heavyweight security protocol as DTLS in a resource constrained IoT network, it is necessary to also apply 6LoWPAN compression mechanisms to the protocol. Therefore, we outline a DTLS-6LoWPAN extension as presented in [Raz+12] and [Raz+13] at the end of this section.

3.3.1 DTLS Overview

DTLS is used to secure UDP traffic and provides E2E security. It is composed of two layers and comprises multiple protocols. The lower layer uses the Record protocol that encapsulates the second layer which in turn contains either application data or one of the three protocols Handshake, Alert and ChangeCipherSpec. The Handshake and ChangeCipherSpec protocols are used by communicating hosts to negotiate several security parameters, like cipher suits and keys, whereas the Alert protocol is used to communicate error messages between DTLS peers. The main task of the Record protocol is to protect the upper layer protocols or, once the handshake process is completed, the application data. The structure of a DTLS secured packet is illustrated in figure 8.

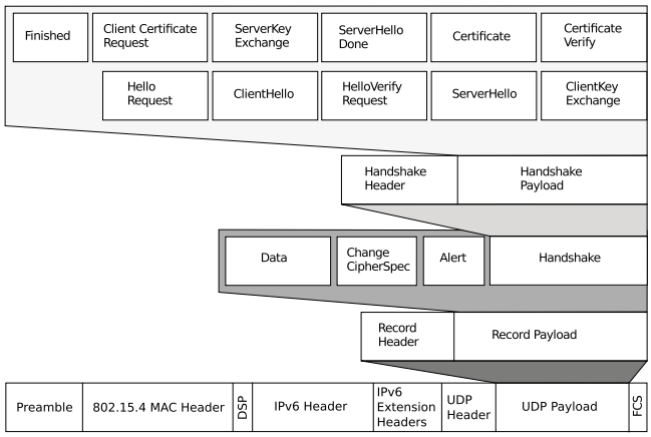


Fig. 8: DTLS secured IP packet [Raz+13].

3.3.2 DTLS-6LoWPAN Extension

As stated before, applying DTLS results in significant overhead which we want to counteract with 6LoWPAN compression. Therefor, we introduce a new NHC encoding for UDP with 11011 as ID bits as it is defined in [Raz+13]. [Raz+12] proposes a similar approach using a different ID. In contrast to the NHC_UDP defined in the 6LoWPAN standard [Hui+11], the new encoding indicates that also a part of the UDP payload is compressed. In our case, this is the DTLS header which is carried in the payload.

We consider two cases for which we specify compression mechanisms. The first case is during the handshake process where the Record encapsulates the Handshake protocol. In this situation Record and Handshake headers are both compressed using NHC_RHS encoding. After the handshake process is finished, the Record protocol contains application data only. Then, just the Record header is compressed using NHC_R encoding. Both encoding schemes, NHC_RHS and NHC_R, are shown in figure 9. The encoding bits should be interpreted as follows:

- The first four bits make up the ID field, which is set to 1000 for NHC_RHS and 1001 for NHC_R.
- If $V = 0$, the version is the latest DTLS version 1.2 and the version field is omitted in the header. Otherwise, it is carried in-line.
- If $EC = 0$, the right most 8 bits of the epoch field make up the compressed epoch field. Otherwise, it comprises all 16 bits.
- For the sequence number, we have to differentiate between the two encodings, as we either have one or two SN bits

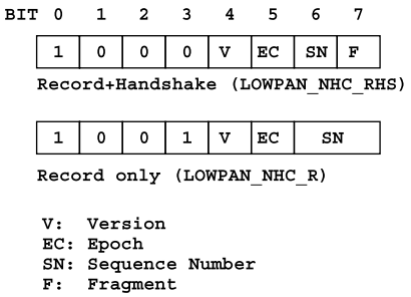


Fig. 9: NHC_RHS and NHC_R encoding [Raz+13].

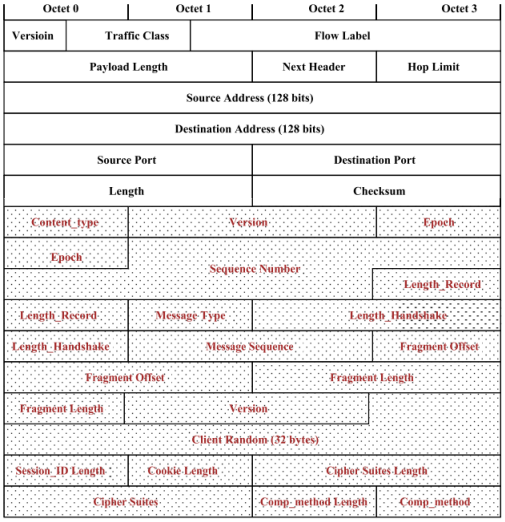
  - In case of NHC_RHS: If $SN = 0$, the right most 16 bits of the 48 bit sequence number are used. Otherwise, the full 48 bit sequence number is used.
  - In case of NHC_R: If $SN = 00$, the right most 16 bits are used. If $SN = 01$, the right most 32 bits are used. If $SN = 10$, the right most 24 bits are used. If $SN = 11$, all 48 bits of the sequence number are used.
- If $F = 0$, the handshake message is not fragmented and the fields fragment_offset and fragment_length are elided in the compressed header. Otherwise, the message is fragmented and the fields are present.

Furthermore, the content_type field of the Record header and the message_type and message_seq fields of the Handshake header are always carried within the according header, whereas the length field of both headers can be omitted.

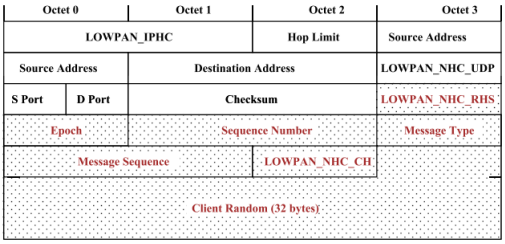
There are also NHC encodings for the ClientHello (NHC_CH) and ServerHello (NHC_SH) messages. Since this would exceed the scope of this paper, we do not further describe them here. However, a detailed description of these encodings can be found in [Raz+12] and [Raz+13]. See figure 10 for a comparison of an uncompressed with a compressed IP/UDP datagram containing a DTLS ClientHello message. As one can see, all fields of the ClientHello message except for the random field can be elided with the proposed NHC_CH compression. The other handshake massages, namely ServerHelloDone, ClientKeyExchange and Finish, cannot be compressed. Additionally, see [Pri+10] for a compression scheme for X.509 certificates that can be used in the optional Certificate and CertificateVerify DTLS messages.

4 Evaluation

After presenting the security and compression mechanisms for the three different layers, we briefly evaluate the proposed techniques in this section according to [Raz+14]. Therefor, we focus on the comparison of link-layer security and IPsec.



(a) Uncompressed DTLS ClientHello message.



(b) Compressed DTLS ClientHello message.

Fig. 10: Comparison of uncompressed and compressed DTLS ClientHello messages [Raz+14].

First of all, IEEE 802.15.4 link-layer security provides hop-by-hop encryption which indicates that every single node in the network has to be trusted. This leads us to the first drawbacks of link-layer security: there is no support for host authentication and key management. Also, the security operations at the link layer have to be performed by every node and, therefore, it is more resource-consuming compared to IPsec. Another important aspect is fragmentation. When a packet is fragmented, link-layer security is applied to every fragment, whereas the E2E encryption of IPsec is applied before fragmentation happens and has to be taken care of only by the end hosts. This means that IPsec is more efficient with increasing number of hops and payload size.

With that in mind, one could argue that using the ESP transfer protocol of IPsec is sufficiently

secure and more efficient than link-layer security. However, when focusing on security it is always advisable to attempt to detect potential attacks as early as possible. According to this principle, link-layer security should be used, as it detects e.g. illegal message modification on layer 2 already on intermediate hosts, whereas in IPsec integrity checks are only done by end nodes.

In conclusion, it is recommended to use link-layer security and IPsec in combination in order to achieve maximum security. Additional to the benefits of IPsec, the concurrent use further provides early threat detection and encryption of IPsec headers. The overhead that comes with the usage of both mechanisms can be counteracted to some degree by e.g. choosing simple cipher algorithms at the link layer when using ESP.

5 Conclusion

As shown in this paper, it is feasible to adapt existing security mechanisms to match the requirements of resource-constrained IoT networks and comply with given standards. We presented different approaches for securing IP traffic in 6LoWPAN networks for layer 2, 3 and 4 and argued on using link-layer security and IPsec concurrently. Concurrent use provides the best protection for IoT traffic. It is important to especially focus on a high level of security, as current developments indicate that the Internet of Things will become a vital part of our daily lives and security should still be ranked first in one's life.

References

[Bor+12] C. Bormann, A. P. Castellani, and Z. Shelby. "Coap: An application protocol for billions of tiny internet nodes". In: *IEEE Internet Computing* 16.2, 2012, pp. 62–67.

[Cen] I. K. Center. *IPsec: Transport mode and tunnel mode*. URL: `https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.halz002/ipsecurity_ipsec_ah_esp_encap_modes.htm` (visited on 06/11/2017).

[Dee98] S. E. Deering. *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification*. 1998.

[Die+08] T. Dierks and E. Rescorla. *RFC 5246: The Transport Layer Security (TLS) protocol version 1.2*. 2008.

[Hui+11] J. Hui and P. Thubert. *RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-based Networks.* 2011.

[IEE16] IEEE. *IEEE Standard for Low-Rate Wireless Networks*. 2016.

[Ken+05] S. Kent and K. Seo. *RFC 4301: Security Architecture for the Internet Protocol*. 2005.

[Ken05a] S. Kent. *RFC 4303: IP Encapsulating Security Payload (ESP)*. 2005.

[Ken05b] S. Kent. *RFC 4302: IP Authentication Header (AH)*. 2005.

[Ma+08] X. Ma and W. Luo. "The analysis of 6LoWPAN technology". In: *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*. Vol. 1. IEEE. 2008, pp. 963–966.

[Mai+11] L. Mainetti, L. Patrono, and A. Vilei. "Evolution of wireless sensor networks towards the internet of things: A survey". In: *Software, Telecommunications and Computer Networks (SoftCOM), 2011 19th International Conference on*. IEEE. 2011, pp. 1–6.

[Mon+07] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *RFC 4944: Transmission of IPv6 packets over IEEE 802.15. 4 networks*. 2007.

[Pri+10] M. Pritikin and D. McGrew. *The Compressed X. 509 Certificate Format - draft-pritikin-comp-x509-00*. 2010.

[Raz+12] S. Raza, D. Trabalza, and T. Voigt. "6LoWPAN compressed DTLS for CoAP". In: *Distributed Computing in Sensor Systems (DCOSS), 2012 IEEE 8th International Conference on*. IEEE. 2012, pp. 287–289.

[Raz+13] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. "Lithe: Lightweight secure CoAP for the internet of things". In: *IEEE Sensors Journal* 13.10, 2013, pp. 3711–3720.

[Raz+14] S. Raza, S. Duquennoy, J. Höglund, U. Roedig, and T. Voigt. "Secure communication for the Internet of Things - a comparison of link-layer security and IPsec for 6LoWPAN". In: *Security and Communication Networks* 7.12, 2014, pp. 2654–2668.

[Res+12] E. Rescorla and N. Modadugu. *RFC 6347: Datagram Transport Layer Security Version 1.2*. 2012.

[Sas+04] N. Sastry and D. Wagner. "Security considerations for IEEE 802.15. 4 networks". In: *Proceedings of the 3rd ACM workshop on Wireless security*. ACM. 2004, pp. 32–42.

[She+11] Z. Shelby and C. Bormann. *6LoWPAN: The wireless embedded Internet*. Vol. 43. John Wiley & Sons, 2011.

[She+14] Z. Shelby, K. Hartke, and C. Bormann. *RFC 7252: The Constrained Application Protocol (CoAP)*. 2014.

[Tan+10] L. Tan and N. Wang. "Future internet: The internet of things". In: *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*. Vol. 5. IEEE. 2010, pp. V5–376.

[Vas+10] J.-P. Vasseur and A. Dunkels. *Interconnecting smart objects with ip: The next internet*. Morgan Kaufmann, 2010.

Concepts for authentication and key exchange in constrained environments

Dominik Bitzer[1]

Abstract: The threat of weak authentication in embedded devices has recently become more obvious than ever. Many of the currently used security algorithms and procedures are not suitable due to special constraints in embedded and Internet-of-Things (IoT) devices, especially the limited resources which can be used for security measures. Cryptographic keys are widely used to ensure security in information systems, which leads to a central role of key management. After introducing the security requirements and special challenges in embedded devices, this paper presents some general and specific technical solutions and current research, such as HIP derivatives, certificate compression and IKEv2 modifications. It becomes clear that further research is needed and no clear standard has established itself yet.

Keywords: Key management; Authentication; Embedded systems; Constrained networks

1 Mirai and the alarming state of security in current IoT products

In September 2016 a botnet of multiple hundred thousand devices surfaced, launching one of the biggest DDoS attacks seen until date. The first prominently known target was the blog of security journalist Brian Krebs - with the scale of the attack so big that even the DDoS mitigation service of one of the world’s largest CDNs Akamai stopped serving his website. In subsequent attacks traffic of up to 1,2 terabit per seconds from this network was measured. The sourcecode of the responsible worm "Mirai" was released shortly after and delivered insights about how this attack was possible. The systems that had been taken over were not desktop computers or servers as known from prior worms, but actually security cameras and other Internet-enabled devices. Interestingly enough, no sophisticated attacks or complicated exploits of security flaws were used, potential targets were simply found in wide-range IP scans of the entire Internet and then infected after login with publicly accessible standard passwords such as "admin", "supervisor" or "default". [HBZ16]

This shows, how weak security in embedded devices is already posing a major threat as of today. And the dangers are quickly growing with the quickly increasing number of network-enabled devices, that come with new applications for embedded computers, such as dishwashers with "smart" functionalities. These dangers are not just heavier attacks of known type but also new types of attacks, for example in adhoc-networks that are not even

[1] Ludwig-Maximilians-Universität München, Institute of Informatics, dominik.bitzer@campus.lmu.de

connected to the Internet. This has been shown in a proof of concept of taking over light bulbs of the Philips Hue series, which could make a worm possible that directly spreads from Hue device to Hue device. Additionally in this example, permanent installation of malicious firmware was possible, signed using previously extracted manufacturer's keys stored in the devices. [O'16]

IoT and embedded products in general will need to secure their communication in order to be viable products for widespread use. This paper first will discuss the security requirements and specific challenges for key management and authentication in embedded systems and then go on to present and compare some possible technical solutions. It ends with a summary of some of the open challenges for strong authentication and secure key management.

2 Traditional IT security and its specific problems in embedded systems

The reason why securing communication in embedded device networks is such a big challenge and needs new technological solutions, is that the requirements are very specific and much different to those of traditional computing systems. While the latter often require similar protocols and underlying technologies, with embedded devices this is often not the case. High performance computers just like home computers use hierarchical network topologies, which are most efficient in comparatively static networks. On the other hand, in many applications of embedded devices (e.g. large scale, flexible sensor networks) research suggests that optimized multicast protocols are the best choice in order to lower complexity and achieve energy savings. [He01] This section will introduce abstract security requirements in information systems and the special challenges in embedded environments.

2.1 Security requirements and the role of key exchange and authentication

Information security requirements are usually approached using abstract desired goals, such as keeping a certain piece of information secret. As a minimal set, usually the CIA triad of confidentiality, integrity and availability are used, but different additional requirements have been defined in literature. In order to fulfill these requirements, widely used technical measures usually require effective handling and distribution of secret or public keys. Below are the seven goals defined in the ISO 27000 standard [Br17, 3ff.]. For each of them, possible attacks on privacy and security in embedded devices are presented, in case the requirements are not fulfilled. Also the role of cryptographic keys and key exchange in solving each of the challenges is shown:

Confidentiality: Only authorized entities may gain access to information. For example, this goal is violated if the live video stream of an IP-camera in a private room is accessed by a hacker. In order to use encryption for the video stream, it is necessary to distribute keys between the camera and authorized entities.

Integrity: Information and devices are secure from unwanted modification. This goal would be violated, if in the update process of an embedded device the update packet is changed during deployment, e.g. in order to include malicious code. Integrity of the packet can be assured using signatures that then are validated using public or shared keys.

Availability: Information or devices are accessible when needed. This goal would be violated if an adversary can disrupt correct functionality or communication of embedded devices through unauthorized access as described in the following bullet point.

Authenticity: Entities can assure through some form of authentication, that the sender of information or commands is actually the communication partner that they are claiming to be. If attackers instead of legitimate employees can access an embedded controller of a power plant, they might be able to disrupt the power supply in a city or overheat an oven in a power plant and cause damage. The embedded system may therefore only grant access after strong authentication has successfully happened beforehand, which is a common use case for PKI.

Accountability: Since not only intruders but also malicious insiders are possible, it is necessary to be able to identify the person that sent information or a command to a device. For example an angry employee might take the role of the attacker in the previous example. In order to track back the origin of a message, each user accessing a resource needs separate keys.

Non-repudiation: An entity can not deny having sent some command or information. For example, the employee from the previous attack should not plausibly be able to deny sending the command once it has been tracked back to them. This is not possible, if other people than the employee have access to the same key. Since this is the case in symmetric key methods, private key methods can be used as a solution.

Reliability: The used procedures work reliably and whenever needed. If during rekeying a jammer blocks a wireless signal and the new keys don't arrive at all devices, communication with them should still be possible afterwards. Therefore a suitable key management solution has to be used, keeping track of old keys and allowing later resending of new keys.

2.2 Specific requirements and challenges for cryptography in embedded systems

Since the underlying approaches for communication in embedded devices differ from full-scale systems, existing solutions such as centralized key distribution centers often are not feasible for use in these specialized scenarios any more. The requirements that lead to some of the most important challenges for cryptography and key management in embedded devices are: [Xi07]

Limited memory, computing power and energy supply: Cryptographic primitives rely on the assumption that an adversary does not have access to sufficient processing power

to decrypt messages without knowledge of the keys in a feasible amount of time. Due to the increasing processing performance of available chips, the amount of time needed for attacks is decreasing every year. A common way of countering this problem is to increase the length of keys and number of iterations in cryptographic methods, therefore increasing arithmetic and memory intensity, which in turn means higher energy consumption. [Xi07] In embedded devices, this development presents a problem because they are not designed for performance increases but other conflicting factors such as decreases in price or size. This means, that these additional constraints have to be considered in the development of cryptographic protocols, especially since transmission already uses the largest part of energy used in embedded sensors and therefore provides the biggest remaining lever for energy consumption reduction. [THM15]

Scalability and flexibility: Many applications of embedded devices include the deployment of large numbers of them. Additionally, a final number may not be known from the beginning but is likely to often change. This could be due to deployment of additional devices or recovery or loss of connection. [Xi07] Therefore, the used protocols need to be able to scale well with large and changing numbers of participating entities.

Longevity: In many non-embedded devices, the hardware is either exchanged on a regular basis (e.g. 2-3 years for smartphones) or easily replaced in case of loss of function. In many application scenarios of embedded devices, some or all of these assumptions do not hold true: e.g. longliving sensors may be deployed in places that are hard to access, such as a canal system. This aspect has to be considered when choosing a cryptographic primitive, that will need to be sufficiently secure for a comparatively long amount of time. [Xi07]

Limited in-field updates and troubleshooting: In classical systems software can easily be upgraded to better or more secure versions or be patched in case of discovered flaws. If problems arise, extensive troubleshooting methods can be employed, since a lot of information can be presented and full remote access is possible over a network connection. For embedded devices bandwidth, memory and energy constraints may make updates impossible. If no display or other indicators are present, the user can't easily be notified about problems, displayed detailed error messages or asked for further procedures, such as it would be done in case of expired certificates in web browsers. [Xi07] Therefore, the used cryptographic primitives and implementations need to be very robust and secure for a long time as stated in the previous bullet point.

Standardization: Full-scale systems, but also for example smartphones, usually support a wide range of cryptographic methods, therefore ensuring compatibility between manufacturers and across versions and allowing gradual introduction of new technologies or cryptographic protocols. In embedded systems, this comfort may often not exist and therefore few, widely used standards are necessary. [Xi07]

2.3 Definitions and general approaches for authentication

The process of authentication is necessary to fulfill the security goal "Authenticity" from chapter 2.1. An authentication protocol generally is an exchange of messages between principals (e.g. actual people, processes or devices) in order to evaluate the validity of further messages. A secret, or information that can only have been generated by using a secret such as an HMAC, can be included in the authentication messages. Some additional information like timestamps, counters or challenges may be included in order to prove, that a message is not replayed. The secrets are usually cryptographic keys, either shared and symmetric or asymmetric public-private pairs. In order to combine the benefits of both, hybrid protocols may be used: symmetric keys can be securely sent with an asymmetric primitive and afterwards be used to secure further communication with a less computationally intensive symmetric key method. [CJ97]

2.4 The central role of efficient key management

Due to the amount of keys needed for the previously described reasons, such as multicast communication as described in section 2 and secure authentication between principals as described in section 2.1, it is obvious that one of the most important issues in securing embedded device communication is providing an efficient but secure management solution that takes care of generation, distribution and installation of keys. This process is not carried out only once at the beginning or only for re-keying when keys have been compromised, but actually on a regular basis as a precautionary measure or if members join or leave the group and backward and forward secrecy of communication are required [An02]. This ensures, that there is a way for the network to revoke access for single members and that no information about all participants is leaked if one node is compromised.

Key management protocols can be separated into the three basic groups of centralized, decentralized or distributed approaches. From the first to the last, this means a growing amount of devices involved in the key management process, ranging from one central entity over subgroup managers to potentially each single device contributing the the generation of keys. [RH03]

3 Technical solutions for authorization and key exchange in embedded devices

In order to fulfill the previously explained requirements, while working within the limits that are set by the constraints, several protocols have been suggested for authentication and key exchange in constrained environments. While some of them have been designed from scratch, others modify established protocols in order to make them usable in embedded devices. The advantage of using such adapted versions of widespread protocols is compatibility with

the existing infrastructure. This chapter will introduce three solutions for different aspects: first it will be shown that X.509 certificates can be compressed in order to make their use more feasible in constrained environments. For key exchange, which is necessary for secure communication, adapted versions of the Internet Key Exchange protocol will be introduced. Last of all HIP and some lightweight derivatives will be introduced. This protocol offers a from of addressing scheme and an integrated solution for authentication and key exchange.

3.1 Compressed X.509 certificates

One of the most widespread solutions for certificates is the X.509 standard, which describes a format of public key certificates that can be used to define a hierarchy of keys that are signed by certificate authorities. Revocation of certificates happens through certificate revocation lists, although in practice approaches like the Online Certificate Status Protocol are used in addition. [Co08] The standard is used in TLS, which again is used for encryption in many application layer protocols such as HTTP or SIP. Even though the X.509 standard only defines a certificate format and hierarchical organization, the certificates can be used as a basis for solutions for all discussed security requirements. This makes consideration for use in embedded devices an obvious option.

Besides several security flaws that have been found in the last couple years, usage of X.509 comes with some challenges for embedded devices. As the encoding of the certificates themselves, the Distinguished Encoding Rules (DER) of the X.690 standard are used. This encoding requires a complex and expensive parser and leads to a size of around 2 kilobyte per certificate, which especially in the case of large numbers of certificates such as described in section 2.4 is a lot in sensor systems. For example the Atmel ATmega328P that is used on the Arduino UNO board offers just 2 kilobyte of memory and 1 kilobyte of storage capabilities. [Co15] Furthermore, the fields that are used to identify the issuer and owner of a certificate were designed while considering few certificate authorities and a moderate number of servers. In the use case of sensor networks with large numbers of embedded devices, this identification scheme is not convenient. Lastly, in decentralized and distributed networks an embedded device should be able to directly provide proof of authorization. Identity certificates do not provide any means for this, instead separate attribute certificates are specified, leading to additional overhead. [SC15]

In order to leverage the advantages of using this widespread standard, it has been suggested to compress the certificates. In order to do so, the large number of optional fields that are usually left empty are omitted. Afterwards, the lossless DEFLATE algorithm, which is mostly known from ZIP compression, can be used to compress the certificates. In order to provide efficiency even when compressing just a single certificate, a pre-defined compression dictionary is used, which is optimized for compression of this specific certificate type by including commonly found strings. Since compression comes with additional computation, storage and memory requirements for the decompression software itself and the compression dictionary, this method is more appropriate if the goals are bandwidth and storage savings.

In case of memory restrictions, the trade-of has to be considered depending on the use case and if the additional expenses are compensated for by storage and bandwidth savings, which are especially big when regularly distributing large numbers of certificates.

In an experiment of compressing 100000 certificates on a certificate transparency monitor, an average saving of 35% was possible by using the deflate algorithm alone. [Ed16] This effect should be even bigger for self-signed certificates, where a lot of fields are included more than once and therefore information is duplicated. This effect es even greater in certificate chains, since additional redundancy is contained. Since usage of self-signed certificate chains is likely in some use cases of embedded devices, the reduction in size should be even greater. If large numbers of certificates are to be deployed, the contents should be kept deliberately sparse, in order to be able to omit more empty fields. [MP10]

3.2 Lightweight IKEv2 and G-IKEv2

The Internet Key Exchange protocol is used in IPsec for the set up of security associations for securing communication on the network layer. It is based on asymmetric cryptography and the Diffie-Hellman key exchange. [Er10] IPsec can be used in conjunction with the 6LoWPAN standard for IP communication in low power wireless devices in order to provide embedded systems a secure approach for communication with good interoperability with the current worldwide network infrastructure. [Ra11] In order to lower the hardware requirements, several approaches exist that address different problems. Compression of IKEv2 can be based on 6LoWPAN header compression, a combination two methods: [RVJ12]

LOWPAN IPHC and next-header compression (NHC): LOWPAN IPHC is used to compress the headers in IP packets and can also compress headers of TCP and UDP. NHC on the other hand allows to leave out the IPv6 field "next header", in case a special next-header field is set. At the same time, having this bit set to one indicates the usage of compression also for the following package, allowing further compression. Additionally the length field can be dropped if the length can be incurred from lower layers.

Elliptic Curve Cryptography (ECC): similarly to the method described in section 3.3, the resource requirements for cryptography can be lowered using ECC. In protocols with asymmetric cryptography, that are based on Diffie-Hellman or RSA, the cryptographic primitive can be replaced in order to lower the computational load and the length of the keys that need to be transmitted.

Since it has been shown that multicast is more efficient than peer-to-peer communication in big groups of devices [He01], key management should also be able to leverage the improved scalability. Otherwise the size of groups would be limited by the key management solution, even though the communication protocols could support many more participants. [RH03]

In order to leverage the possible savings [Fe17], a group key management protocol that uses IKEv2 (G-IKEv2) has been proposed. While staying close to the syntax of IKEv2 messages and it's request-response pairs of messages, message types are defined in a way to provide multicast support. For group members that want to communicate, a key is used that was previously distributed by the "Group Controller/Key Server (GCKS)". This GCKS also distributes policy and key updates by using a newly defined group message exchange. [WNS17]

3.3 D-HIP, DEX and LHIP: HIP modifications for embedded devices

In the traditional model of the Internet the majority of nodes were expected to just rarely change their location. This lead to the fact that IP addressing is more or less tightly coupled with the physical connection and therefore the location of a device. In today's world, a device might quickly change between wireless LAN networks, mobile data and a wired connection. Since this leads to a different IP address every time, applications need to frequently recover their connections and announce their new IP address. An example for this would be wearable devices like smartwatches, that their owner wears on their commute. So it might quickly switch from Wifi to mobile network, and then connect to Wifi once is at work. With the current architecture, frequent updates about the route that a node currently is reachable most easily would have to be sent. In sensor networks, some form of multicast is thinkable, where multiple sensors are addressable by the same identifier even though they are connected to different networks. On the other hand also multihoming is possible, where one device is connected over multiple network connections. [GSW11]

Different approaches have been developed such as the IETF standard Mobile IP [Pe11]. An alternative approach is the Host Identity Protocol (HIP), which is an additional layer between the network and the transport layer that introduces an identifier that communication partners can use. It therefore decouples the identity of a device from it's location, which is called the Identifier / Locator Split Paradigm. In order to achieve this, mobile hosts register their current IP address to a rendezvous server (RVS), which can be contacted by other devices in order to achieve initial reachability. The RVS then forwards any incoming packets to the host. The host contacts and identifies to the initiator of the communication [Mo08] Since this type of addressing is especially suitable for complex and quickly changing topologies such as in sensor networks, HIP is considered a strong candidate for an authentication protocol in embedded devices.

Both during announcing an updated locator IP address at the RVS, as well when identifying to other hosts, a HIP entity needs to authenticate in order to protect the identifier so it may not be spoofed. HIP solves the security requirement "Authenticity" by using signatures and MACs and the derived keys can be used to solve the remaining requirements. The solution is called the HIP Base Exchange (BEX): the initiator (I) contacts a node, which from then-on is called the responder (R), to agree on a shared secret using the Diffie–Hellman (DH) key exchange. Afterwards IPsec ESP can is used in order to secure further communication

of HIP. In order to mitigate a Man-in-the-middle attack, pre-shared keys or a public key infrastructure are needed. A big drawback is the computational load of the cryptographic operations of RSA based cryptography and Diffie-Hellman key exchange. [SO12]

The exact process of a HIP Base Exchange can be seen in figure 1. R answers to the request I1 of I by sending a packet R1 with a puzzle, it's DH public key DH_R, the public host identifier key HI_R and the accompanying signature. After calculating K_{DH} and checking the signature, I sends the solution to puzzle (which is supposed to mitigate DoS attacks on R), their own keys DH_I and HI_I and the accompanying signature. If the signature and the solution to the puzzle are correct, R computes the Diffie-Hellman session key K_{DH} and sends the MAC that it computed with K_{DH} and a signature to I. Computations are needed on both sides for calculating K_{DH}, solving the intentionally expensive puzzle and generating and checking signatures. The signature process usually is negligible, but has to be considered in constrained systems.

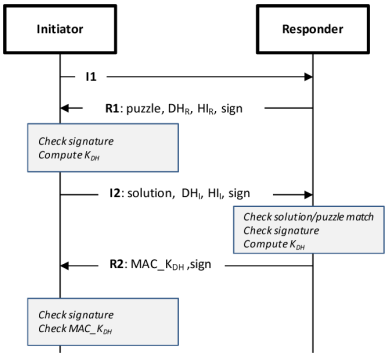


Abb. 1: Host Identity Protocol Base Exchange [SO12]

There are proposed modifications such as HIP Diet Exchange (DEX), which replace the host identifier by a long-term Elliptic Curve Diffie-Hellman public value. Elliptic curve cryptography provides memory, energy and bandwidth savings through smaller key sizes and faster computation. [Gu04] This is solving the problem, that the widespread asymmetric cryptography primitives like RSA and Diffie-Hellman are becoming infeasible for use in embedded devices due to sharply growing key sizes with higher performance and energy demands. DEX only slightly lowers the computational load, since it doesn't change the BEX protocol but only switches to less computationally expensive cryptography. Lightweight HIP (LHIP) keeps the BEX syntax but replaces RSA and DH with hash chains. LHIP compromises on security, since the HIP BEX mechanisms are not used but just hashes over consecutive messages are calculated as a basic security mechanism. D-HIP is a proposed distributed version of BEX, that is most apt for devices with very restricted resources and high security requirements. [SO12]

The process can be seen in figure 2. Maintaining the high level of security while lowering the computational load is achieved through delegating parts of the Diffie-Hellman key

exchange of BEX from the embedded devices to less restricted nodes, called proxies (P_I). After sending I1 and receiving R1, I generates and then splits up the secret exponent a, which it sends together with R1 to P_I. The proxies also require a proof of authorization in order to authenticate to R. These dynamically managed authorization proofs are handled by a trusted computer with higher storage and computation capabilities and access to a steady energy source. Each proxy of P_I then sends their part of the exponent to R, which validates the solution to the puzzle and reassembles the exponent in order to compute K_{DH}. At the same time, each proxy of P_I calculates their shared key's part. The proxies then each receive the MAC that R computed with K_{DH} along with the signature, after which they forward their computed part of K_{DH} to I, which assembles the single parts to the final K_{DH}.

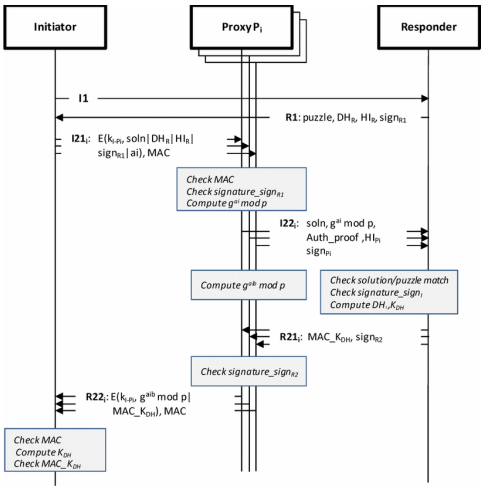


Abb. 2: Outsourcing DH to a proxy in D-HIP [SO12]

Even though there is additional communication overhead that leads to higher energy consumption, overall a 75% energy saving has been achieved on the initiator side in a sample implementation. It has to be considered though, that these savings lead to higher expenses for the proxy nodes. These additional nodes raise the cost and complexity of the overall installation which contradicts the point of using cheap embedded devices. [SO12]

4 Key management and strong authentication as prerequisites for a trustworthy Internet of Things

Embedded devices provide a massive potential for use-cases in the current process of digitalization in many industries, governmental use and also for end-users. As shown, this transition to a more connected but still secure world will only be successful, if security concerns will be taken seriously. This paper has shown the central role of authentication and key exchange in such constrained environments. Many proposed solutions already exist, but no clear state of the art has established itself yet.

Further research will have to happen at the same speed as current cryptographic protocols become obsolete and considered insecure, either since design flaws are found or the advancement of computational power renders old versions insecure. Also solutions have to be found for the upgrading of old devices to newer, more secure methods. Another important aspect is the question of how to handle devices after central keys have been compromised. If re-keying does not happen fast enough, they might be compromised in the meantime and restoration of a secure mode of operation might not be possible without physical access.

In fields with higher security requirements and less restricted budgets, such as the military, case studies for special solutions such as security modules with special coprocessors are currently being researched. [Va16] This shows a possible way of handling security, since it makes separation of security from the functional parts of the device possible and could be exchanged if needed. In other fields, the additional cost that is incurred by approaches of this kind could make deployment infeasible though. This again shows the trade-off between cost and security.

Literaturverzeichnis

[An02] Anderson, Ross: Two remarks on public key cryptology. Bericht, University of Cambridge, Computer Laboratory, 2002.

[Br17] Brenner, M.; Felde, N.; Hommel, W.; Metzger, S.; Reiser, H.; Schaaf, T.: Praxisbuch ISO/IEC 27001: Management der Informationssicherheit und Vorbereitung auf die Zertifizierung. Carl Hanser Verlag GmbH & Company KG, 2017.

[CJ97] Clark, John; Jacob, Jeremy: , A survey of authentication protocol literature: Version 1.0, 1997.

[Co08] Cooper, D.; Santesson, S.; Farrell, S.; Boeyen, S.; Housley, R.; Polk, W.: RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Bericht, Mai 2008.

[Co15] Corporation, Atmel: , ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH DATASHEET. http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf, 2015. (Accessed on 06/07/2017).

[Ed16] Edgecombe, Graham: , Compressing X.509 certificates. https://www.grahamedgecombe.com/blog/2016/12/22/compressing-x509-certificates, 12 2016. (Accessed on 06/07/2017).

[Er10] Eronen, Pasi; Nir, Yoav; Hoffman, Paul E.; Kaufman, Charlie: , Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, September 2010.

[Fe17] gentschen Felde, N.; Guggemos, T.; Heider, T.; Kranzlmüller, D.: Secure Group Key Distribution in Constrained Environments with IKEv2. 2017.

[GSW11] Grayson, Mark; Shatzkamer, Kevin; Wierenga, Klaas: Building the Mobile Internet. 2011. (Accessed on 05/30/2017).

[Gu04] Gura, Nils; Patel, Arun; Wander, Arvinderpal; Eberle, Hans; Shantz, Sheueling Chang: Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In: International Workshop on Cryptographic Hardware and Embedded Systems. Springer, S. 119–132, 2004.

[HBZ16] Herzberg, Ben; Bekerman, Dima; Zeifman, Igal: , Breaking Down Mirai: An IoT DDoS Botnet Analysis |. https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html, 10 2016. (Accessed on 05/05/2017).

[He01] Heidemann, John; Silva, Fabio; Intanagonwiwat, Chalermek; Govindan, Ramesh; Estrin, Deborah; Ganesan, Deepak: Building Efficient Wireless Sensor Networks with Low-level Naming. SIGOPS Oper. Syst. Rev., 35(5):146–159, Oktober 2001.

[Mo08] Moskowitz, et al.: , Host Identity Protocol. https://tools.ietf.org/html/rfc5201, 04 2008. (Accessed on 07/05/2017).

[MP10] McGrew, D.; Pritikin, M.: , The Compressed X.509 Certificate Format. https://tools.ietf.org/html/draft-pritikin-comp-x509-00, 11 2010. (Accessed on 05/20/2017).

[O'16] O'Flynn, Colin: , A LIGHTBULB WORM? - Details of the Philips Hue Smart Lighting Design. http://colinoflynn.com/wp-content/uploads/2016/08/us-16-OFlynn-A-Lightbulb-Worm-wp.pdf, 8 2016. (Accessed on 04/29/2017).

[Pe11] Perkins, et al.: , Mobility Support in IPv6. https://tools.ietf.org/html/rfc6275, 07 2011. (Accessed on 07/05/2017).

[Ra11] Raza, Shahid; Duquennoy, Simon; Chung, Antony Ho Ming; Yazar, Dogan; Voigt, Thiemo; Roedig, Utz: Securing Communication in 6LoWPAN with Compressed IPsec. In: 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS). IEEE Xplore, S. 1–8, 6 2011.

[RH03] Rafaeli, Sandro; Hutchison, David: A Survey of Key Management for Secure Group Communication. ACM Comput. Surv., 35(3):309–329, September 2003.

[RVJ12] Raza, Shahid; Voigt, Thiemo; Jutvik, Vilhelm: Lightweight IKEv2: A Key Management Solution for both the Compressed IPsec and the IEEE 802.15.4 Security. 2012.

[SC15] Schukat, M.; Cortijo, P.: Public key infrastructures and digital certificates for the Internet of things. In: 2015 26th Irish Signals and Systems Conference (ISSC). S. 1–5, June 2015.

[SO12] Saied, Y. B.; Olivereau, A.: D-HIP: A distributed key exchange scheme for HIP-based Internet of Things. In: 2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM). S. 1–7, June 2012.

[THM15] Trappe, Wade; Howard, Richard; Moore, Robert S: Low-energy security: Limits and opportunities in the internet of things. IEEE Security & Privacy, 13(1):14–21, 2015.

[Va16] Vai, Michael; Whelihan, David J; Nahill, Benjamin R; Utin, Daniil M; O'Melia, Sean R; Khazan, Roger I: Secure Embedded Systems. LINCOLN LABORATORY JOURNAL, 22(1), 2016.

[WNS17] Weis, B.; Nir, Y.; Smyslov, V.: Group Key Management using IKEv2. Bericht, March 2017.

[Xi07] Xiao, Yang; Rayi, Venkata Krishna; Sun, Bo; Du, Xiaojiang; Hu, Fei; Galloway, Michael: A survey of key management schemes in wireless sensor networks. Computer Communications, 30(11–12):2314 – 2341, 2007. Special issue on security on wireless ad hoc and sensor networks.

Technischer Datenschutz

Applications of Homomorphic Encryption in a privacy respecting (Cloud-based) Internet-of-Things

Tobias Heider[1] heidert@nm.ifi.lmu.de

Abstract: Internet-of-Things (IoT) devices already make up a big share of today's internet-connected devices. It is predicted that in the future IoT sensor networks will monitor and automate almost all aspects of people's lives. Due to the constrained nature of sensor devices many IoT systems utilize cloud solutions to provide both storage and computational resources. The outsourcing of large amounts of sensor data to potentially untrusted cloud servers presents a great risk for user privacy.
A possible solution is presented by the use of Homomorphic Encryption (HE) which allows arbitrary computations on the encrypted cipher, instead of computing on the plaintext data and thus hides the real value of the performed operation from the computing entity. This can allow the outsourcing of computations on privacy sensitive data without leaking sensitive information. It is shown that homomorphic encryption provides a theoretic foundation to solve various privacy problems arising from the use of constrained devices in conjunction with cloud solutions, but is limited by the performance of existing HE schemes.

Keywords: Homomorphic Encryption; IoT; Cryptography; Cloud; Security; Sensor Networks

1 Introduction

Today, embedded computers in the form of IoT or smart devices can be found in almost all kinds of everyday objects, including cars, household electronics or smart homes. An especially popular kind of IoT networks are wireless sensor networks, which can help monitor, control and automate almost any aspect of life. The main value of sensor networks arises from a large amount of collected data which allow users to benefit from statistical evaluation of said data. The amount of data generated presents a challenge for the constrained storage capabilities of end-devices. An immediate solution to this problem is the utilization of cloud services, that are either self hosted or, more commonly, outsourced to third-party providers with scalable resources. While the adaption of IoT systems has a great potential value for users, it is apparent that an ever surrounding sensor network also poses a great risk for user's privacy. The adoption of cloud solutions adds a new dimension of privacy problems as collected data is stored on eventually untrusted third-party systems, which has raised many concerns and has been subject to various research works in the past [Io11][He16].

[1] LMU Munich, Institut für Informatik, Oettingenstr. 67, 80538 München, Germany heidert@nm.ifi.lmu.de

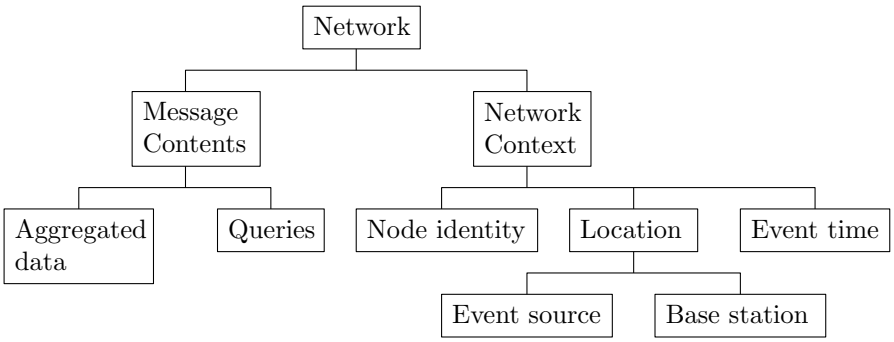


Fig. 1: *Network-centric* Privacy Problems in WSN [Lo17]

This work proposes homomorphic encryption, a technique allowing transparent computations to be performed on the ciphertext instead of a plaintext as a solution to privacy problems arising from cloud-based IoT networks.

The remainder of this paper is organized as follows:
First, Section 2 examines privacy problems arising from a use of IoT networks in conjunction with cloud platforms and formulates a concrete problem statement. Section 3 introduces homomorphic encryption as a possible solution to the stated problem. The fundamental concept of homomorphic encryption schemes and the definitions necessary to understand the mathematical foundation of homomorphic encryption is described in detail, followed by an overview of known homomorphic encryption schemes. The last subsection considers the limitations of existing HE schemes. Section 4 presents several practical real world applications of homomorphic encryption techniques in the cloud-based IoT. Last, Section 5 draws a conclusion about the usefulness of homomorphic encryption in privacy respecting IoT solutions as well as it's practicability in an IoT setting.

2 Privacy in the Cloud-based IoT

Several works have been published dealing with privacy in the specific setting of IoT sensor networks and cloud services [VFS12] [He16]. A useful classification of privacy with a focus on IoT sensor networks is presented in [Lo17]. The privacy definition is split into *user-centric* and *network-centric* privacy. The first describes privacy problems related to sensors detecting human presence or other assets, providing the ability to be used as spy tools. This kind of privacy problem can not be resolved by technology alone, but is commonly solved through legislative regulations and fair information practices. *Network-centric* privacy describes a scenario where the attacker is a hostile actor in the network trying to learn from content and metadata already available in the existing network traffic.

Figure 1 illustrates *network-centric* privacy problems in IoT networks and the privacy critical information provided. Network privacy is divided into *content-privacy* and *context-privacy*. *Context-privacy* treats the privacy of queries as well as the actual aggregated data communicated over the network. *Context-privacy* describes less obvious privacy problems arising from privacy leaks through metadata like location or timing of the communication. The focus of this work to explore the possibility of providing *content-privacy* with the help of cryptographic encryption. The field of *context-privacy* can not be solved by cryptographical means and is thus out of scope of this contribution.

Problem Statement Traditional cryptographic *content-privacy* solutions allow only the secure storage of data on possibly untrusted cloud-storage solutions. In order to preserve confidentiality, data has to be downloaded and encrypted before it can be further processed. The application of cloud computing without leaking privacy critical data is not possible. A practical privacy respecting cloud-based IoT system must provide confidentiality and privacy of data not only in cloud storage applications but also in cloud computing application handling privacy critical sensor data.

A solution is presented by the use of homomorphic encryption (HE) schemes, which could allow cloud computing platforms to compute on the cryptographic ciphertext instead of the plain text data and thus does not leak real data values to untrusted cloud solution providers.

3 Homomorphic Encryption

A homomorphism describes a map from one algebraic structure to another, of the same type (e.G. groups, rings, algebras) preserving the operations of the original structure. Homomorphic encryption schemes utilize this property by making the encryption function a homomorphism from a structure containing the set of plaintext messages and the valid operations on this set to a structure containing the set of resulting cipher texts and the corresponding operations. This allows any operation that can be computed on the plain text data to be performed on the encrypted ciphertext.

A homomorphic public-key encryption scheme HE = (Keygen, Enc, Dec, Eval) typically implements four probabilistic polynomial time (PPT) algorithms:

Key Generation (Keygen): The key generation algorithm $(pk, ek, sk) \leftarrow Keygen()$ outputs a valid key tuple containing a public and secret key as well as the evaluation key ek used for the *Eval* algorithm.

Encryption (Enc): The encryption algorithm has the form of $e \leftarrow Enc(pk, m)$. It takes a public key generated with *Keygen* as well as the plain text message and returns the encrypted cipher-text e.

Decryption (Dec): The decryption algorithm $m \leftarrow Dec(sk, e)$ takes the secret key sk as well as a cipher e and returns the decrypted plaintext message m.

Evaluation (Eval): The evaluation algorithm $f(e) \leftarrow Eval(ek, f, e_1..., e_i)$ takes the evaluation key ek, a function f and a number of input cipher texts e_1, ..., e_i and outputs a homomorphically evaluated result ciphertext $f(e)$

Figure 2 illustrates the application of such a scheme: A function f, that can be computed over the plain text message m, is instead computed over the encrypted cipher-text e. The evaluation $Eval(f)$ yields $f(e)$, which decrypts to the result obtained from applying f to the plaintext message m. This allows to perform operations on data without leaking the real value of the input and output data.

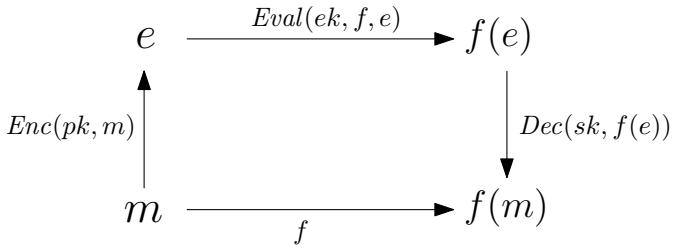


Fig. 2: Evaluation of a function f in a homomorphic encryption scheme

3.1 Definitions and Notions

Homomorphic encryption schemes can be categorized by the form of homomorphism they implement. This section gives a number of mandatory definitions in order to distinguish HE schemes based on their homomorphic properties.

Given an encryption scheme $HE = (Keygen, Enc, Dec, Eval)$, where M denotes the set of plain text messages and E the set of resulting ciphertexts.

Definition 1 (Group Homomorphism) *A map $Enc : M \rightarrow E$ from a group $(M, +_M)$ to a group $(E, +_E)$ is called group homomorphic, if it satisfies:*

$$\forall m_1, m_2 \in M, \qquad Enc(m_1 +_M m_2) = Enc(m_1) +_E Enc(m_2)$$

HE is called a *partially homomorphic* encryption (PHE) scheme. Group homomorphisms can either be additively or multiplicatively homomorphic, depending on the groups operation, not both at the same time.

Ring and algebraic homomorphisms are both additively and multiplicatively homomorphic and are defined as follows:

Definition 2 (Ring/Algebraic Homomorphism) *A map* $Enc : M \rightarrow E$ *from a ring/algebra* $(M, +_M, *_M)$ *to a ring/algebra* $(E, +_E, *_E)$ *is called ring/algebra homomorphic if it satisfies:*

$$\forall m_1, m_2 \in M, \qquad Enc(m_1 *_M m_2) = Enc(m_1) *_E Enc(m_2),$$
$$Enc(m_1 +_M m_2) = Enc(m_1) +_E Enc(m_2)$$

In order to express more complex operations on the ciphertext than simple additions and multiplications, functions on the cipher are commonly treated as arithmetic circuits which use boolean operations (AND, OR, NOT) instead of additions and multiplications. Homomorphic encryption schemes can further be defined by the class of functions that can be homomorphically evaluated.

Definition 3 (Somewhat Homomorphic Encryption (C-Homomorphism)) *Let C be a class of arithmetic circuits, a encryption scheme* HE *is C-homomorphic (or somewhat homomorphic) if it satisfies:*

$$\forall m \in M, \forall f \in C \qquad Dec(Eval(f, Enc(m))) = f(m)$$

In simple words: a somewhat homomorphic encryption scheme allows the evaluation of a limited class of circuits on the cipher.

Definition 4 (Fully Homomorphic Encryption) *An encryption scheme* HE *is called fully homomorphic if it is homomorphic for the class of all arithmetic circuits*

Fully homomorphic encryption allows any circuit to be calculated over the cipher and thus enables arbitrary computation. All known homomorphic encryption schemes additionally allow a further homomorphic evalutation on outputs of *Eval* (multi-hop homomorphism).

A special form of FHE is the leveled fully homomorphism which is defined as follows:

Definition 5 (Leveled Fully Homomorphic Encryption) *A encryption scheme* HE *is called leveled fully homomorphic if it's key generation algorithm* Keygen *takes an additional argument L, and* HE *is homomorphic for the class of all circuits with depth L.*

3.2 Related Work

The idea of utilizing homomorphisms in encryption schemes was originally introduced by Rivest et al. in [RAD78], in an effort to describe a privacy respecting banking scheme using the multiplicative homomorphic properties of the original RSA algorithm [RSA78].

Other early partially homomorphic encryption schemes include the encryption systems of Goldwasser and Micali [GM82], ElGamal [El85] as well as Paillier [Pa99]. The first Somewhat Homomorphic Encryption scheme supporting an unlimited number of additions, but only a single multiplication was proposed by Boneh, Goh and Nissim in [BGN05].

In his breakthrough work [Ge09], Gentry showed, that any SHE scheme able to homomorphically evaluate a slightly augmented version of it's own decryption circuit can be transformed into a Fully Homomorphic Encryption scheme. This technique was consequently called *bootstrapping*. Because of the general rareness of SHE schemes, *bootstrappable* schemes are even harder to come by. Gentry introduced another technique called *squashing* that allows to reduce the complexity of the decryption circuit, by preprocessing the cipher during the encryption phase. It can be used to make non *bootstrappable* SHE scheme *bootstrappable*. The downside of this technique is the requirement for another hardness assumption, in this case based on the Sparse Subset Sum Problem (SSSP). With the help of *bootstrapping* and *squashing* Gentry managed to design the first *leveled* Fully Homomorphic Encryption scheme, based on *ideal lattices*. The scheme could further be leveraged to a full (non-leveled) FHE scheme, by making a circular security assumption based on the security of encrypting the secret key with it's own public key. Gentry's scheme was further optimized in terms of efficiency and key sizes in works of Smart and Vercauteren [SV10] as well as Stehle and Steinfeld [SS10]. Several other FHE schemes have been developed utilizing Gentry's, bootstrapping and squashing techniques, including [Va10] which introduces a new SHE scheme based on integers, as well as [BV11] which relies on the ring learning with errors assumption [LPR10].

In [GH11] Gentry et al. proposed a FHE scheme constructed without squashing and thus not relying on the SSSP assumption, but rather on the Decisional Diffie-Hellman assumption or ideal lattices. Based on this revelation several FHE schemes without the need for squashing. Brakerski and Vaikuntanathan made use of this new techniques in [BV14] and presented a scheme solely based on the learning with errors (LWE) assumption as in [Re09]. The contribution of this work, in the form of a modulus refinement technique, was further simplified and used to construct the first "(Leveled) Fully Homomorphic Encryption without bootstrapping" in [BGV14]. To achieve non-leveled FHE bootstrapping is still necessary.

In a recent contribution [GSW13] Gentry, Sahai and Waters constructed a FHE scheme based on the LWE problem which uses matrices instead of vectors as ciphertexts. This removes the necessity for an expensive relinearization step, required by previous LWE based approaches. Additionally it allows homomorphic evaluation without the need for a special evaluation key and as a result lead to the development of two compilers which allow to transform identity-based and attribute-based LWE encryption schemes into FHE schemes.

3.3 Limitations of homomorphic encryption schemes

While homomorphic encryption offers obvious advantages over traditional non-homomorphic encryption schemes, they are not necessarily a viable choice for every application of traditional cryptography. This section reviews several limitations of known HE schemes.

Performance The main limiter in the adoption of homomorphic encryption in real world systems is the general poor performance of homomorphic crypto systems. The most resource efficient homomorphic encryption schemes are partially homomorphic schemes like the Paillier cryptosystem, which on the downside are severely limited in their usefulness for real world applications. As natural SHE schemes are rare, most SHE schemes are constructed to perform better in additive operations than in multiplicative ones. [NLV11] analyzed the performance of an implementation of the LWE based SHE scheme presented in [BV11] and measured a computing time of 20ms to compute the addition of 100 integers used to derive the statistical *mean* after decryption, while a computation of the sum of square roots and the sum in order to derive the *variance* took a total of 6s (measured on a consumer laptop computer). The additional overhead of Gentry's *bootstrapping* technique, which is used in the construction of all true FHE schemes known today makes the use of FHE in most real world applications unpractical. Nevertheless, Microsoft in 2015 released their *Simple Encrypted Arithmetic Library* (SEAL) [LP16], which in it's current version implements the Fan-Vercauteren [FV12] FHE scheme. The use of similar schemes on microprocessors, as they are used in today's sensor networks, however, still is highly impractical for their expensive encryption procedures.

Security Another consideration when using homomorphic encryption schemes is the limited maximal achievable security. The highest notion of security, namely: *Indistinguishability under adaptive Chosen Ciphertext Attack* (IND-CCA2) can not be achieved by HE schemes, as it requires *non-malleability*. *Malleability* describes a property of encryption schemes that allows to transform a encryption of a message m into an encryption of $f(m)$ for a known function f, which is allowed in HE by design. Most schemes presented in Section 3.2 are designed to provide semantic security against chosen plaintext attacks (IND-CPA). The design of a IND-CCA1 secure HE scheme was an open problem until recently, when Canetti et al. presented a technique to transform any FHE scheme to a IND-CCA1 secure scheme in [Ca17].

4 Applications of Homomorphic Encryption in the IoT

Technical limitations in IoT devices commonly lead to the use of external computing or storage capabilities. Often these requirements are outsourced to third-party untrusted cloud services instead of self-hosted solutions. Homomorphic encryption presents a helpful

tool to protect privacy-sensitive data from third-party providers, without limiting on-data computations, by either user, or cloud computer. Practical applications of homomorphic encryption in conjunction with cloud security and privacy have been widely studied [Ch09] [TEE12] [NLV11]. In the following, three concrete practical applications are presented that use homomorphic encryption schemes to protect privacy critical data in the specific setting of IoT networks, utilizing cloud solutions for both, storage and computing capabilities.

4.1 Private Cloud Storage

An application of IoT systems combined with cloud storage and computing capabilities in the form of a electronic medial records (EMR) system was presented in [NLV11]. It proposes a system that collects users vital health information, for example in the form of blood pressure, blood sugar readings or a heart monitor. The information could be collected using wearable smart monitoring devices streaming them to a cloud server, which can consequently compute statistics over the users data and return warnings, alerts or the plain statistics. In a primitive solution the servers data could be stored encrypted on the cloud server using traditional cryptographic schemes. In order to evaluate the data, the devices needs to first download and decrypt the stored data before it can be used in further computations. An advantage from the use of cloud solutions is limited as the device still needs to provide enough storage for the downloaded encrypted data and further has to perform operations itself in order to preserve the information's confidentiality. A better solution is offered by the utilization of a homomorphic encryption scheme: The patients privacy is protected by encrypting the information homomorphically, which allows the cloud provider to compute statistical functions like the mean, standard deviation and logical regression over the encrypted inputs. Only the result is returned to the users device which can then be decrypted to it's real value. What makes this application practical, is that a SHE scheme might be sufficient as all named statistical functions require mostly additions and only a few multiplications.

To take this example even further, various approaches have been made to build homomorphically encrypted relational databases [GGE15] [Bo13]. The achievement of [Bo13] is the design of a system that enables private database queries: a concept which allows the client to issue a database query and learning the result without learning anything about other values stored in the database. At the same time, the server does not learn what was queried by the client. The presented system uses only SHE and is the first to allow both conjunction and disjunction queries.

4.2 Delegated Computing

One of the main reasons to adopt cloud services is the possibility to outsource computational expensive tasks from low power embedded devices to the cloud. Homomorphic encryption

can be used in such delegated computations in order to protect both, the data and the computed function from a hostile cloud actor. Extending the example from above, consider an application where a constrained device might need to perform an expensive computation in the form of a complex algorithm on the generated data. This setting inhabits several non-trivial problems that can not be solved with traditional cryptography. First, in order to preserve privacy the untrusted cloud server must be able to compute over the data without learning the value of the input data or the result. Second, there must be a possibility for the client to validate the correctness of the computation performed by the cloud server. In [GGP10] Gennaro proposes the notion of *verifiable computation* and constructs a protocol that allows a weak client to verify the correctness of an outsourced computation. Additionally the scheme uses homomorphic encryption to provide input and output privacy. An obvious constraint is that the preparation and validation, in total, must be less expensive than the computation itself, for the protocol to be viable. The scheme was further optimized in [CKV10] which reduces the key size and the workload of the delegator.

4.3 Multiparty Computation

Consider a setting similar to the EMR system presented in subsection 4.1. Users store their encrypted health monitor information on an untrusted cloud service and can request homomorphically computed statistics from the server. Different from the original setting, users might not only be interested in statistics computed only over their own data but also might be interested in computing the *mean* of all users measurements. In a privacy respecting solution, each user's value should stay private and all participants only receive the accumulated result. The problem is comparable to Yao's millionaires problem in [Ya82]. The field dealing with this class of problems is called Multiparty Computation (MPC). An early notable MPC system based on HE was presented by Cramer in [CDN01]. The system allows any number of distributed clients to jointly evaluate a function over their inputs and is secure against an active adversary corrupting a minority of the participating clients. A newer approach was presented in [LTV12] which allows *on-the-fly multiparty computation* with the help of multikey FHE and a computationally powerful but untrusted cloud server. The system could be applied to the EMR example as follows: Each user uploads his encrypted records to the cloud, the encryption and uploading is independent of the computed function and the number or identity of other users. The cloud then computes a function over multiple users data, chosen by either a single user or by the cloud itself, and receives a encrypted result. In the last step each affected user approves the function as well as the peers and all users work together in order to decrypt the computations result. Consequently the users only interaction is during the encryption and decryption phase, the workload of computing a function happens on the cloud server.

5 Conclusion

The adoption of IoT, especially in conjunction with cloud technology, presents a new challenge for privacy. Section 2 categorized privacy into user and network privacy, which can further be divided into *content-centric* privacy, dealing with the protection of actual data communicated over a network and *context-centric* privacy, which deals with the protection of privacy critical metadata communicated with network messages. This work showed that various research branches studying applications of homomorphic encryption, provide a theoretical foundation to the solution of *content-centric* privacy problems in cloud-based IoT applications. They enable privacy protected cloud storage solutions as well as protected delegated computing and multiparty computing, allowing outsourcing of computations without leaking critical data. The practicability of homomorphic encryption schemes is limited by the bad performance of existing schemes. Especially FHE schemes constructed using Gentry's bootstrapping technique, while being the most useful of HE schemes, add a considerable overhead. All measurements of existing HE schemes have been made on consumer desktop or laptop CPUs. In order to evaluate the practicability of the presented solutions, an evaluation of the performance of the proposed methods on embedded microprocessors, as they are used in sensor networks, is an important future step.

References

[BGN05] Boneh, D.; Goh, E.-J.; Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Theory of Cryptography Conference. Springer, pp. 325–341, 2005.

[BGV14] Brakerski, Z.; Gentry, C.; Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) 6/3, p. 13, 2014.

[Bo13] Boneh, D.; Gentry, C.; Halevi, S.; Wang, F.; Wu, D. J.: Private database queries using somewhat homomorphic encryption. In: International Conference on Applied Cryptography and Network Security. Springer, pp. 102–118, 2013.

[BV11] Brakerski, Z.; Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Annual cryptology conference. Springer, pp. 505–524, 2011.

[BV14] Brakerski, Z.; Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing 43/2, pp. 831–871, 2014.

[Ca17] Canetti, R.; Raghuraman, S.; Richelson, S.; Vaikuntanathan, V.: Chosen-Ciphertext Secure Fully Homomorphic Encryption. In: IACR International Workshop on Public Key Cryptography. Springer, pp. 213–240, 2017.

[CDN01] Cramer, R.; Damgård, I.; Nielsen, J. B.: Multiparty computation from threshold homomorphic encryption. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 280–300, 2001.

[Ch09] Chow, R.; Golle, P.; Jakobsson, M.; Shi, E.; Staddon, J.; Masuoka, R.; Molina, J.: Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM workshop on Cloud computing security. ACM, pp. 85–90, 2009.

[CKV10] Chung, K.-M.; Kalai, Y.; Vadhan, S.: Improved delegation of computation using fully homomorphic encryption. In: Annual Cryptology Conference. Springer, pp. 483–501, 2010.

[El85] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory 31/4, pp. 469–472, 1985.

[FV12] Fan, J.; Vercauteren, F.: Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive 2012/, p. 144, 2012.

[Ge09] Gentry, C. et al.: Fully homomorphic encryption using ideal lattices. In: STOC. Vol. 9. 2009, pp. 169–178, 2009.

[GGE15] Gahi, Y.; Guennoun, M.; El-Khatib, K.: A secure database system using homomorphic encryption schemes. arXiv preprint arXiv:1512.03498/, 2015.

[GGP10] Gennaro, R.; Gentry, C.; Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Annual Cryptology Conference. Springer, pp. 465–482, 2010.

[GH11] Gentry, C.; Halevi, S.: Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In: Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on. IEEE, pp. 107–109, 2011.

[GM82] Goldwasser, S.; Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the fourteenth annual ACM symposium on Theory of computing. ACM, pp. 365–377, 1982.

[GSW13] Gentry, C.; Sahai, A.; Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology–CRYPTO 2013. Springer, pp. 75–92, 2013.

[He16] Henze, M.; Hermerschmidt, L.; Kerpen, D.; Häußling, R.; Rumpe, B.; Wehrle, K.: A comprehensive approach to privacy in the cloud-based Internet of Things. Future Generation Computer Systems 56/, pp. 701–718, 2016.

[Io11] Ion, I.; Sachdeva, N.; Kumaraguru, P.; Čapkun, S.: Home is safer than the cloud!: privacy concerns for consumer cloud storage. In: Proceedings of the Seventh Symposium on Usable Privacy and Security. ACM, p. 13, 2011.

[Lo17] Lopez, J.; Rios, R.; Bao, F.; Wang, G.: Evolving privacy: From sensors to the Internet of Things. Future Generation Computer Systems/, 2017.

[LP16] Laine, K.; Player, R.: Simple Encrypted Arithmetic Library - SEAL (v2.0), tech. rep., Sept. 2016, URL: https://www.microsoft.com/en-us/research/publication/simple-encrypted-arithmetic-library-seal-v2-0/.

[LPR10] Lyubashevsky, V.; Peikert, C.; Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 1–23, 2010.

[LTV12] López-Alt, A.; Tromer, E.; Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Proceedings of the forty-fourth annual ACM symposium on Theory of computing. ACM, pp. 1219–1234, 2012.

[NLV11] Naehrig, M.; Lauter, K.; Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop. ACM, pp. 113–124, 2011.

[Pa99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 223–238, 1999.

[RAD78] Rivest, R. L.; Adleman, L.; Dertouzos, M. L.: On data banks and privacy homomorphisms. Foundations of secure computation 4/11, pp. 169–180, 1978.

[Re09] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) 56/6, p. 34, 2009.

[RSA78] Rivest, R. L.; Shamir, A.; Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21/2, pp. 120–126, 1978.

[SS10] Stehlé, D.; Steinfeld, R.: Faster fully homomorphic encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp. 377–394, 2010.

[SV10] Smart, N. P.; Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: International Workshop on Public Key Cryptography. Springer, pp. 420–443, 2010.

[TEE12] Tebaa, M.; El Hajji, S.; El Ghazi, A.: Homomorphic encryption applied to the cloud computing security. In: Proceedings of the World Congress on Engineering. Vol. 1, pp. 4–6, 2012.

[Va10] Van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, pp. 24–43, 2010.

[VFS12] di Vimercati, S. D. C.; Foresti, S.; Samarati, P.: Managing and accessing data in the cloud: Privacy risks and approaches. In: Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on. IEEE, pp. 1–9, 2012.

[Ya82] Yao, A. C.: Protocols for secure computations. In: Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on. IEEE, pp. 160–164, 1982.

Blockchain: Applications and Challenges for embedded networks

Maximilian Hünemörder[1]

Abstract: The concept of the Internet of Things (IoT) is a promise of a comfortable future. But with it new challenges arise for both the privacy of the user and the privacy of such embedded networks. In order to allow for automatic customization and personalization of any smart object connected to such a network, service providers need to analyze private data. This requires a collection of sensors. Such a collection could allow greedy providers to collect much more data from unsuspecting users than ever before. Addtionally, malicious attackers might be able to gain information by analyzing the context in which messages are transmitted in such a network. In order to solve this challenge a consensus between comfort and privacy must be found. The blockchain concept might offer a solution to this conundrum. A trustless distributed network might allow for innovative IoT applications that respect the privacy of all involved parties. Such applications might be smart homes, eHealth and smart governance. When comparing the privacy needs of the IoT to the different flavors of the blockchain concept it is apparent that the current state of the technology is not yet completely ready to be used for embedded networks. This is due to hardware constrains, mostly the size of current blockchain implementations. But there have been very recent applications and concepts that show that we are not that far away from a breakthrough towards an IoT based upon blockchain.

Keywords: blockchain; embedded networks; Internet of Things; privacy

1 Introduction

The concept of the Internet of Things (IoT) is a promise of a comfortable world. A world where objects around us are thinking, computers become invisible and our homes and environments are smart. Everything automatically adapts to our personal tastes and many arduous daily tasks can be avoided all together. But this gained comfort comes with a possible loss of privacy and that might be a steep price to pay considering the amount of private data being collected to achieve this comfort. Additionally, the IoT offers many security challenges that have to be addressed before completely stepping over the threshold to this new computing paradigm. Unfortunately, at least partially, the IoT has already arrived before all these concerns were adequately addressed. Now it is mostly based on centralized cloud services lead by big companies with commercial interests.
An alternative middleware for embedded networks could be distributed networks. [RZL13] Distributed networks are an idea that is as old as the invention of the internet itself. In

[1] Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539, München, Deutschland, max@huenemoerder.de

his 1964 paper *On distributed networks* Paul Baran defined commonly used categories for network structures. These categories are centralized, decentralized and distributed networks. A centralized network has one central node through which all data send from each node is directed. A decentralized network corresponds to a network of smaller centralized networks, whereas in a distributed network each node is equal in the sense that data is passed directly between the nodes. In this paper Baran then correctly predicts the future of a worldwide network as a centralized structure and already pleads for a distributed approach where data is passed directly between individual nodes and not controlled by large entities. [Ba64]
In the last decade technologies have been developed that might prove useful to allow the middleware of the IoT to be a distributed network. For example, developments in cryptocurrency, the advent of Blockchain, the concept behind services like bitcoin[2], ethereum[3] and many others could possibly offer fitting tools to allow for an distributed Internet of Things that addresses privacy and security risks attached to it.
The aim of this paper is to analyze which needs of the IoT are actually satisfied by the different flavors of the blockchain concept. The main focus will be on the privacy and security needs of IoT applications.

2 The Needs of Embedded Networks

In order to realize any embedded network, that respects the users privacy and protects the whole system from malicious attackers, several needs have to be addressed:

User-Centric Privacy UCP, that is to say protecting the personal data of the user from data hungry businesses or network providers, is a challenge to personal privacy that becomes even more important with the rise of the IoT. Since the IoT builds upon smart objects that 'think' by using a collection of different sensors to gain information about the user and react to this input, there is tons of personal data collected this way.
This data might even be voice recordings and video material depending on the application and as such could be used to spy on an unaware user. Furthermore, in a traditional internet setup a user has to actively interact to send data. In the context of the IoT, with computers fusing into our environment and becoming somewhat invisible, users might not even aware that they are online and their activities are being recorded. [Lo17]

- As such important aspects of UCP are **Anonymity** and **Access Control**. Many IoT Applications are supposed to be used in rather public settings, for example a hospital. So there might be occasions where it is necessary that only certain other users can access the data and know the identity of the originator. Whereas other users should only be allowed to read data, but not know excactly to whom it belongs. This would be especially true for applications in the health sector, since patient data should only

[2] https://bitcoin.org/, Seen on 08.06.17
[3] https://www.ethereum.org/, Seen on 08.06.17

be available to the corresponding doctors, while at the same time this data could be used anonymously for scientific studies. [Si14, We10]

- **Confidentiality**, i.e. transactional privacy is the "property that information is not made available or disclosed to unauthorized individuals, entities, or processesäs defined in the ISO/IEC 27000:2016(E) standard. [IS16]

On the other hand **Network-Oriented Privacy NOP** describes the danger of an external attacker trying to gain sensible information about the network itself, for example through analyzing the network traffic. This can be split into two categories: **Content-Oriented Privacy** and **Context-Oriented Privacy**. [Lo17]

Protecting the actual content of messages transmitted in a network from external attackers and eavesdroppers enables **content-oriented privacy**. In general many dangers can be avoided by using proper authentication and encryption, but there are still challenges that have to be regarded, when trying to ensure content-oriented privacy:

- **Unlinkability**, in other words an attacker should not be able to gain access to different sessions of the same user and aggregate small data amounts into actual meaningful information. [Ha17]
- **Query Privacy** or preventing attackers from finding out which nodes are involved, if any user queries the network for particular information. An attacker might realize that certain nodes are only used when specific queries are made. The attacker could then infer the nature of these nodes. [Lo17]

Context-oriented privacy is breached when an attacker can gain any kind of private data by analyzing the context in which messages are sent, instead of the actual content of the messages. [Lo17] Amongst other things this might be accomplished by analyzing

- the **Temporal Context**, which means that an attacker listening to messages being transmitted can predict future events by analyzing the timing of messages being received.
- the source and destination addresses in packet headers in order to get some information about the **Identity** of certain nodes.
- communcation patterns in a network in order to gain information about the physical **Location** of important nodes.

Next to privacy it might also be important to protect **Data Quality** [Si14] and have some sort of fallback in case of a node failure, both depending on the application the embedded network is used for.

An additional challenge facing any implementation of the IoT is that the smart objects that make up the IoT usually might not possess strong hardware capabilities, mostly due to their size.

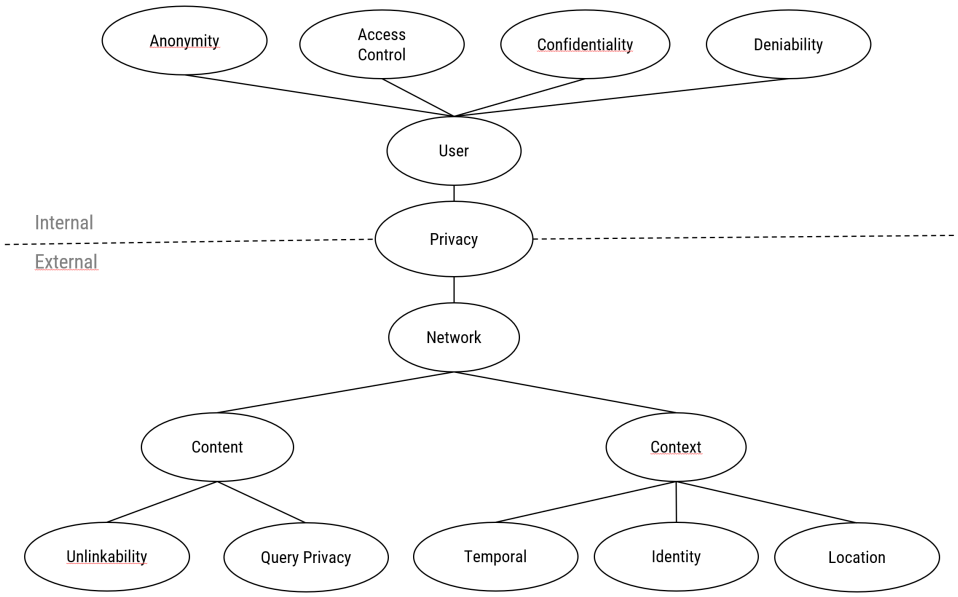


Fig. 1: A overview of the types of privacy relevant to embedded networks. (Adapted from [Lo17])
.

3 Blockchain

This section describes the technological background of blockchain. The first part is about the most basic version which was developed around financial applications and the second part lists all the different versions that were later derived from the basic one.

3.1 Bitcoin and the basic concept of Blockchain

The idea of a blockchain originated from the concept of the cryptocurrency bitcoin, which was published under a pseudonym in 2008. [Na08]
The idea behind bitcoin was to have a method of sending cash from one party to another without the need of a financial institution. Instead of having a trusted third party mediate the transactions, the author proposes a system based on cryptographic proof. The proposed

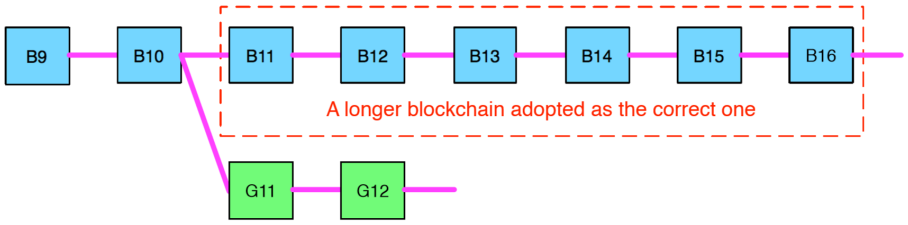


Fig. 2: A basic blockchain. One branch was accepted and the other one is left behind. Taken from [Zh16]
.

solution to this is a public ledger of transactions that is distributed amongst all members of a network. This ledger is a data structure consisting of timestamped blocks, where each is identifiable by its hash and has a reference to the block that came before it. This creates a chain of blocks and is thereby commonly refered to as the blockchain. This is achieved by having a majority decision on which transactions are accepted into the chain. This distributed method of reaching consensus differs between different implementations of Blockchain. [CD16]
In the case of bitcoin this is called Proof-Of-Work (POW) and is made possible by having each involved party compute a complex puzzle, that is complex to solve but easy to verify, to add a new block to the chain. This process is also referred to as mining. This way the miner has to use a lot of computing power to create a new block, but other nodes can easily check if this work was actually done by the mining node. Consequently the number of votes is not determined by the number of IP addresses that one individual possesses but rather dependent on the CPU Power controlled by this individual party. [Na08] In summary the advantage of using this basic version of block chain lies in the fact that it is:

- A fully **distributed** peer-to-peer system, because each node in the network owns a complete version of the chain and any changes, that have been decided by global consensus are passed on from node to node until each node has a copy of the latest verified version. It is very resistant to one of the nodes failing. [CD16]
- **Persistent**, because any blocks added to the chain can not be changed after consensus is reached.
- **Pseudonymous**, because blocks only contain the public key of the involved parties and not more information, though attackers might gain more data from the network context.
- Completely **retraceable** since all processes are verifiable and globally available and consequently auditable by every participant in the network.

3.2 Extended Blockchain and Smart Contracts

After taking a look at the basic version of blockchain it should be easy to see that a more general approach is needed to address the privacy needs of the IoT. Fortunately there already are a sheer plethora of extensions to the basic blockchain concept, developed both by commercially and scientifically interested research groups. In order to categorize these a taxonomy is needed. Christidis et al. [CD16] suggest three basic attributes with which blockchain based networks can be classified:

- Whether access to the network is public or private. Dependent on this the consensus strategy has to be chosen.
- If there is any kind of User Level, i.e. not all parties are allowed to add transactions.
- Does it allow to add arbitrary logic to the blockchain which can then be run on the involved nodes or does it only support bitcoin style transactions, where an assets is either added or removed.

Only the third category can be applied to differentiate between different versions of blockchain. Each of these versions can be implemented either in a private or a public network and with different user levels or not. But the possibility of running turing complete functions does allow for blockchain to be more than a mere replacement for a bank. It can be a peer-to-peer shared computer that can run user defined programs in a trustless environment. These functions are often referred to as smart contracts. An example for a service that implements a blockchain version with customizable transactions, i.e. smart contracts is ethereum.
Another way to categorize these different flavors of blockchain is their consensus strategy. Each of these strategies is based upon each node voting on which transaction gets to be added to the chain and ideally the majority decision is not influenced by one party using multiple identities to be able to have more than one vote. As such any consensus strategy has to be built to avoid this possibility. It was already stated, that bitcoin uses Proof-Of-Work to ensure the number of votes is dependent on the CPU Power. This way it is very expensive to have multiple identities and more lucrative to use all available computing power for only one mining process. [CD16]
One alternative to Proof-Of-Work are Proof-Of-Stake based methods of reaching consensus. Instead of depending on the computing power to proof the identity, here the age and amount of already owned coins are taken into consideration. Nodes proof their right to change the chain with the amount of old coins they are ready to put at stake. [TSed]
As hinted at earlier most versions of blockchain can be trusted for correctness and availability but not privacy due to the fact that in order to be verified the whole chain has to be openly available to all nodes in the network. This also means all transactions are public. In order to create a blockchain version without this attribute, projects like Zerocash[4] and ZCash[5]

[4] http://zerocash-project.org/, seen on 11.06.17
[5] https://z.cash/, seen on 11.06.17

have added Zero-Knowledge Proof ZKP to the blockchain concept. A Zero-Knowledge Proof is a way for a user to proof that a statement of another person is true, without actually gaining any other information about that person. For blockchain this could mean that miners could add transactions that are verifiable without the other nodes needing to know about the content of the transaction. [Mi13]
Hawk is a version of blockchain which combines both customizable smart-contracts with Zero-Knowledge Proof. [Ko16]

4 Blockchain and the IoT

As the research question that is discussed in this paper is wheter the amalgamation of embedded networks and blockchain can be of benefit to the IoT, this section looks into the concepts of real world applications that that were already explored by researchers. The second part then evaluates which of the different versions of blockchain meet the requirements of the IoT and which additional problems might arise from using them.

4.1 Possible Applications

One of the more basic applications could be firmware updates for smart objects. Basically the hash of the latest update is always stored on a blockchain that is distributed between all the IoT devices. Each node can then download the update from some kind of peer-to-peer (p2p) file system. At the beginning of each release, one node of the service provider still has to distribute the latest update to a few nodes, but after a certain percentage of nodes have been updated, the SP node can be shut off for the time being and the updates are distributed between the nodes of the users. New nodes can be automatically given the latest chain and thereby are easily updated to the latest version without the possibility of someone smuggling in modified code. [CD16]

Dorri et al. have explored the possibility of using a blockchain without a financial aspect nor POW to allow for both internal control and external communications of smart homes. In their concept each smart home comes with a high resource computer which is a always online miner in a blockchain based network connecting it to other smart homes. At the same time it keeps a separate private blockchain, that connects all devices inside of its own private network. This way the dataflow to the outside can be controlled. That means a service provider can still collect needed data from the external blockchain and analyze it to create global statistics, but sensitive private information from single devices can be stopped from being seen by any other nodes. The private network can also be used to defend against external attackers trying to hack sensitive devices inside, for example a smart lock, because all transactions have to be verified by the miner. [Do17]

Another very interesting idea is using blockchain to create immutable, distributed and smart Electronic Health Records. These can be automatically distributed amongst clinicians and

therapists. On example of this is MedRec, which provides a multi-institutional medical record that could be kept throughout the whole life of a patient. [Ek16]

Another area of application might be electronic elections. When outfitted with proper anonymytiy a blockchain based approach to electronic voting could prove as a world changing technology. Votes would be immutable, completely auditable and publically available. FollowMyVote [CC16] and BitCongress[6] are examples of such an idea.

4.2 Evaluation

Generally spoken the primary advantage of blockchain for user-oriented privacy is the fact that it enables IoT services to move away from centralized network structures. But as already mentioned in the last section the most apparent privacy concern of creating a basic blockchain based embedded network is that the content of the chain is completely public to all participants of the network.
The same is true for smart contracts. The code is publicly available since it is added directly to the chain. This includes the data sent to the contracts. This means while it might seem that the user is anonymous, the Context-Oriented Privacy of the network cannot be ensured, because every transaction of that user is signed with the same public key. This violates identity privacy and thereby anonymity since anyone with access to the blockchain can read all transactions and identify communication patterns to connect addresses to public keys. This might lead them to identify a specific user. [TSed]
This denies confidentiality as well, because each node has to read each transaction in order to verify it. This problem is only solved by ZKP based block chain networks like ZeroCash and Hawk, since here the content of the chain do not have to be revealed in order to allow any node to verify added transactions, thereby also decreasing the possibility of context based attacks.
The problem that may arise with ZPK based networks is that these do need a lot of CPU power which IoT devices usually not possess. This is a general problem since for example bitcoin has already reached over 100GB just to synchronize a single node[7].
This size problem is addressed by Kravitz and Cooper's Permissioned Blockchain, which was specifically designed with the IoT in mind. Borrowing a concept from git they propose to not only reference the direct predecessor in a block, but rather keep a reference to each previous transaction that is related to it. That way a single device does not always need the whole blockchain, but rather just a git-style branch that contains all the transactions related to the device or just all transactions from the last 24 hours.
They also suggest transaction expiration. Instead of keeping every transaction ever made in the blockchain, parts of it can be deleted after an arbitrary time period excluding the transactions that are related to any blocks that prevail.

[6] http://www.bitcongress.org/, seen on 11.06.17
[7] http://www.coinfox.info/news/6700-bitcoin-blockchain-size-reaches-100-gb, seen on 11.06.17

Furthermore they employ a system where only the hashcode of the content is stored on the chain, and the actual data can only be retrieved by a authorized requester. This ensures both Unlinkability and Confidentiality, but Context-Oriented Privacy might still be problematic. It seems that an attacker can identify a group of devices by their group name and so the attacker might be able to identify the user by knowing when a user adds a new device to the blockchain. Then they can further track and identify other devices by this user. Though this heavily depends on the implementation of such a network and would need to be investigated further. [KC17]

	POW / bitcoin	POS	smart contracts / ethereum	ZKP / ZeroCash	Hawk	Permissioned Blockchain
Anonymity	-	-	-	+	+	+
Confidentiality	-	-	-	+	+	+
Unlinkability	-	-	-	+	+	+
Context-Oriented	-	-	-	+	+	-
Compatible with IoT Hardware	-	+	/	- -	- -	+ +

Fig. 3: A comparison between the most common variants of blockchain and the privacy needs of the IoT

5 Conclusion

Judging from the evaluation and looking at figure 2 one might assume that blockchain is not actually the ideal solution for embedded networks. One of the most severe problems seems to be the scalability of the blockchain. But there are several factors that might still allow this idea to come to fruition, that are featured in some of the discussed implementations. Firstly by moving away from the cryptocurrency concept, the financial aspect of blockchain can be discarded. The resulting loss of monetary value of blocks might allow for additional global operations on the otherwise immutable blockchain, for example removing outdated parts of it to keep the size in check, as suggested by Kravitz and Cooper. Secondly future implementations should always consider what data is actually saved on the blockchain, and what data can be stored off-chain. As seen with the firmware update idea, where only the version management is done by the blockchain and the actual update is distributed via p2p. Additionally many blockchain based IoT applications might work better if they are used in a private network or an internal blockchain in a private network that is itself a node in a public one, i.e. the way the smart home solution by Dorri et al. was built. [Do17]. In conclusion blockchain does deliver a fresh concept for embedded networks and could definitly become very important in the near future. At the same time there is a need for a new holisitic definition of blockchain for IoT for such implementations to succeed, since at the moment the research and technology seems rather unfocused and dissected.

References

[Ba64] Baran, Paul: On distributed communications networks. IEEE transactions on Communications Systems, 12(1):1–9, 1964.

[CC16] Caiazzo, Francesca; Chow, Ming: A Block-Chain Implemented Voting System. 2016.

[CD16] CHRISTIDIS, KONSTANTINOS; DEVETSIKIOTIS, MICHAEL: Blockchains and Smart Contracts for the Internet of Things. 2016.

[Do17] Dorri, Ali; Kanhere, Salil S; Jurdak, Raja; Gauravaram, Praveen: Blockchain for IoT security and privacy: The case study of a smart home. In: Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on. IEEE, pp. 618–623, 2017.

[Ek16] Ekblaw, Ariel; Azaria, Asaph; Halamka, John D; Lippman, Andrew: , A Case Study for Blockchain in Healthcare:"MedRec" prototype for electronic health records and medical research data, 2016.

[Ha17] Haus, Michael; Waqas, Muhammad; Ding, Aaron Yi; Li, Yong; Tarkoma, Sasu; Ott, Jorg: Security and Privacy in Device-to-Device (D2D) Communication: A Review. IEEE Communications Surveys & Tutorials, 2017.

[IS16] ISO/IEC: 27000:2016(E):Information technology — Security techniques — Information security management systems — Overview and vocabulary. 2016.

[KC17] Kravitz, David W.; Cooper, Jason: Securing User Identity and Transactions Symbiotically: IoT Meets Blockchain. 2017.

[Ko16] Kosba, Ahmed; Miller, Andrew; Shi, Elaine; Wen, Zikai; Papamanthou, Charalampos: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: Security and Privacy (SP), 2016 IEEE Symposium on. IEEE, pp. 839–858, 2016.

[Lo17] Lopez, Javier; Rios, Ruben; Bao, Feng; Wang, Guilin: Evolving privacy: From sensors to the Internet of Things. Future Generation Computer Systems, 2017.

[Mi13] Miers, Ian; Garman, Christina; Green, Matthew; Rubin, Aviel D: Zerocoin: Anonymous distributed e-cash from bitcoin. In: Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, pp. 397–411, 2013.

[Na08] Nakamoto, Satoshi: , Bitcoin: A peer-to-peer electronic cash system, 2008.

[RZL13] Roman, Rodrigo; Zhou, Jianying; Lopez, Javier: On the features and challenges of security and privacy in distributed internet of things. 2013.

[Si14] Sicari, Sabrina; Rizzardi, Alessandra; Coen-Porisini, Alberto; Cappiello, Cinzia: A NFP model for internet of things applications. In: Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on. IEEE, pp. 265–272, 2014.

[TSed] Tschorsch, Florian; Scheuermann, Björn: Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. to be published.

[We10] Weber, Rolf H: Internet of Things–New security and privacy challenges. Computer law & security review, 26(1):23–30, 2010.

[Zh16] Zheng, Zibin; Xie, Shaoan; Dai, Hong-Ning; Wang, Huaimin: Blockchain Challenges and Opportunities: A Survey. 2016.

Neue Trends in eingebetteten Kommunikationssystemen

Praktische Anwendung von Fog-Computing am Beispiel von intelligenten Verkehrsmanagementsystemen und Smart Grid

Daniel Ertl [1], Philipp Götzinger [2]

Abstract:

Cloud-Computing und das Internet der Dinge finden sowohl in der Industrie als auch in der Forschung seit geraumer Zeit Beachtung. Ein recht neues Modell ist das Fog-Computing, welches das Cloud-Computing erweitert. Hierbei wird ein geografisch verteiltes Netz aus vielen kleinen Rechnern und Sensoren aufgespannt. Dies bietet den großen Vorteil, dass bei Anwendungen mit hohem Datenaufkommen die zu übermittelnde Datenmenge reduziert werden kann, da ein Großteil der Rechenarbeit schon vor dem Übermitteln der Daten ausgeführt werden kann. Dies verspricht schnellere Antwortzeiten und niedrigere Latenzen in verschiedensten Anwendungsszenarien. In diesem Paper werden sowohl die Architektur als auch anhand von zwei Anwendungsszenarien die Vor- und Nachteile von Fog-Computing erläutert, sowie die damit verbundenen Herausforderungen dargestellt.

Keywords: Fog-Computing; Smart Traffic; Smart Grid; Internet of Things; Car-To-X; Embedded Systems; intelligentes Verkehrsmanagement; E-Mobility

1 Einleitung

Mit dem immer weiter steigenden Verkehrsaufkommen geht auch eine höhere Wartezeit und Stauhäufigkeit im Straßenverkehr einher. Zur Reduktion dieser Effekte müsste man die Infrastruktur baulich vergrößern, was sehr hohe Kosten nach sich ziehen würde und nicht immer möglich ist. [AS15, S. 6] Eine günstigere Alternative wäre die bessere Auslastung des bereits vorhandenen Straßennetzes durch ein intelligentes Verkehrsmanagementsystem, welches die Verkehrsteilnehmer mittels Navigationsanweisungen möglichst gleichmäßig auf die verfügbare Infrastruktur verteilt. Die Analyse der Verkehrssituation sowie die Berechnung der optimalen Routen kann dabei über ein Fog-Computing-Netzwerk geschehen, das sich auf ein großes Gebiet ausdehnt und durch einfache Skalierbarkeit zukunftssicher ist. Auch das Stromnetz steht vor neuen Herausforderungen wie zusätzliche Last sowie erhöhte Unregelmäßigkeit von Last und Erzeugung - auch verursacht bei einer Umstellung auf erneuerbare Energien und Elektroautos. Zur Vermeidung von Instabilitäten kann ein

[1] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538, München, Deutschland, ertld@cip.ifi.lmu.de
[2] Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstraße 67, 80538, München, Deutschland, goetzinger@cip.ifi.lmu.de

intelligentes Steuerungssystem namens Smart Grid verwendet werden, welches die Last zeitlich auszubalancieren versucht.

Diese Arbeit beschäftigt sich mit den Fragen, wie Fog-Computing für eine bessere Auslastung der Verkehrsinfrastruktur helfen kann, und wie man in Echtzeit der Strompreis an die aktuell herrschende Nachfrage anpassen kann. In Kapitel zwei wird die Architektur eines Fog-Netzwerkes vorgestellt, und dargestellt, wie sich dessen Eigenschaften mit denen eines Cloud-Computing-Netzwerks unterscheidet. Kapitel 3 und 4 befassen sich mit Einsatzszenarien und Funktion von Fog-Computing an den Beispielen eines intelligenten Verkehrsmanagementsystems (Smart Traffic) und einem intelligenten Stromnetzen (Smart Grid). Dabei werden deren Vor- bzw. Nachteile gegenübergestellt. Kapitel 5 besteht aus einer Zusammenfassung der Erkentnisse dieser Arbeit und gibt einen Ausblick auf noch offene Fragen dieses Themas.

2 Fog-Computing-Architektur

Im Gegensatz zu Cloud-Computing besteht Fog-Computing nicht aus in einem Rack montierten High-End-Servern, die in großen Rechenzentren stehen, sondern aus vielen in der Nähe der Clients (meist aus der Desktop-Klasse) befindlichen geografisch verteilten Knoten, die logisch dezentralisiert sind. [ea17, S. 2] Damit können zum einen Latenzen und Ausfallrisiken gesenkt und zum andern die Belastung der Backbones reduziert werden. [Bo12, S. 1]

Ein Hauptmerkmal des Fog-Computing Paradigmas ist Mobilität. Dies muss bei Anwendungen die in Fog-Infrastrukturen eingesetzt werden berücksichtigt werden. Denn im Gegensatz zum Cloud-Computing, wo Anwendungen in nur einer Cloud zu einem Zeitpunkt eingesetzt werden (es sei denn, die Notwendigkeit der Skalierung ist jenseits der Kapazität eines einzelnen Cloud-Providers), kommt es in Fog-Netzwerken häufiger vor, dass Nutzer sich, während sie sich bewegen, nur kurz mit dem Netzwerk verbinden. [ea17, S. 2-3]

Mit Fog-Computing entstehen neben den Vorteilen auch eine Reihe von neuen Herausforderungen. Dadurch dass Daten zwischen verschiedenen physikalischen Geräten wie Fog-Knoten, Clients und Back-End-Server bewegt werden können, wird das Datenmanagement sehr viel komplexer. Beispielsweise entsteht die Frage, welche effiziente Algorithmen genutzt werden können, um Daten zwischen Geräten hin und her zu transportieren. Letztendlich ist ein signifikantes Problem jedoch das Thema Privacy und Security: Aufgrund des heterogenen Netzaufbaus werden diese beiden Themen oftmals dem Vorzug der generellen Funktionalität und Kompatibilität untergeordnet. Weiter werden komplexe Algorithmen und Security-Protokolle oft mit Fehlern implementiert oder konfiguriert und lassen so kritische Nutzerdaten für Angreifer offen. [ea17, S. 4]

3 Smart Traffic

3.1 Was bedeutet Smart Traffic?

Für viele Automobilhersteller findet E-Mobilität Beachtung. Zum einen der Wechsel von konventionellen Fahrzeugen mit Verbrennungsmotor hin zu Elektroantrieben als auch das autonome Fahren. Diese Entwicklung bringt einige neue Herausforderungen mit sich, wie beispielsweise den Bedarf von Systemen, die die Fahrzeuge miteinander kommunizieren lassen. Beide Themen profitieren von einem hohen Vernetzungsgrad, der mittels Fahrzeug-zu-X-Kommunikation (Car-To-X) die Fahrzeuge mit der Infrastruktur sowie untereinander vernetzt, um beispielsweise die nächstgelegene freie Ladestation anzuzeigen, oder um auf ein hinter einer Kurve liegen gebliebenes Fahrzeug hinzuweisen. Mit den Anforderungen wachsen auch die Lasten der in Fahrzeugen eingebetteten Rechensysteme, welche deshalb aufgerüstet werden müssen. Um das Potential dieser Entwicklung optimal auszunutzen, bietet es sich an, diese auch mittels Smart Traffic zur Minimierung Verkehrsbelastung vor allem in urbanen Gebieten zu verwenden.
Hierfür wird ein intelligentes Verkehrsmanagementsystem aufgebaut, welches in der Lage ist, mit Hilfe von IoT-Geräten in Echtzeit die momentane Verkehrssituation auf einer großen Fläche zu erfassen, zu analysieren und zu steuern. Aufgrund der hohen Datenmengen, welche ein solches System verarbeiten muss, wären die Kosten für die benötigte Infrastruktur sehr hoch. Abb. 1 zeigt einen schematischen Aufbau eines IVS am Beispiel von FOX (Fast Offset XPath). [ea15]

Durch eine dezentrale Auslagerung mit Hilfe von Fog-Computing und Wireless Sensor Networks (WSN) lägen diese deutlich unter der eines klassischen zentralen Back-Ends. [C15, S. 223]

Ein Fog-Computing-System lässt sich deutlich kleiner skalieren, da ein Großteil der benötigten Rechenarbeit bereits in der Infrastruktur erledigt wird. Zur Analyse und Steuerung des Verkehrs muss somit nur noch ein Bruchteil der sonst zu übertragenen Datenmenge versendet werden.

3.2 Verwandte Arbeiten

Zu den Problemen des intelligenten Verkehrsmanagements gibt es verschiedene Lösungsansätze. Grundsätzlich zielen die meisten Lösungen darauf ab, mit Hilfe des Internets der Dinge, Funknetzwerken und Cloud-Computing, dem steigenden Verkehrsaufkommen entgegenzutreten.

Das Fast Offset XPath (FOX) System, das mit Hilfe von Fog-Computing ein intelligentes Verkehrs System ermöglicht, um Staus und Wartezeit an Ampeln zu senken. FOX nutzt dazu (den Fog-Knoten entsprechende) Roadside Units (RSU) genannte Rechner, die am Straßenrand aufgestellt werden, die die Sammlung der Daten übernehmen. Die gesammelten

Daten werden verarbeitet und können via LTE an andere RSUs gesendet werden, welche dann Alternativrouten berechnen. [ea15]

Mit Hilfe von Type-Based Content Distribution (TBCD) und maschinellem Lernen (ML) simulieren Sahoo und Sahoo 2017 ein Software-defined Network (SDN), welches durch RSUs die Anzahl zu übertragender Datenpakete im Vergleich zur klassischen Unicastübertragung bei steigender Fahrzeuganzahl drastisch reduzieren konnte. Dabei stellen Fog-Geräte Daten, Speicher und Rechenleistung für den sich in der Nähe der RSU befindenden Nutzer zur Verfügung. [SS17]

Durch Technologien wie Funksensortechnik, RFID, GPS, Cloud-Computing und das Internet of Things (IoT), wird im Papier „Intelligent Traffic Information System Based on Integration of Internet of Things and Agent Technology“ von Al-Sakran ein Netzwerk für ein Verkehrsinformationssammel- und Kontrollsystem aufgebaut, um in Echtzeit Verkehrsinformationen zu erfassen, zu speichern und auszuwerten. Mit den erhaltenen Datensätzen lassen sich potenzielle Gefahren wie Unfälle oder Staus erkennen und unmittelbar Gegenmaßnahmen durch Verkehrsbeeinflussungsanlagen einleiten. [AS15]

3.3 Architektur eines intelligenten Verkehrsmanagementsystems

Zur Sammlung der Daten nutzt das System in RSUs verbaute Sensoren wie RFID, drahtlose Sensornetzwerke, Kameras und intelligente Terminals, um Daten der mobilen Objekte zu anderen Sensoren zu übertragen. [Ks15]

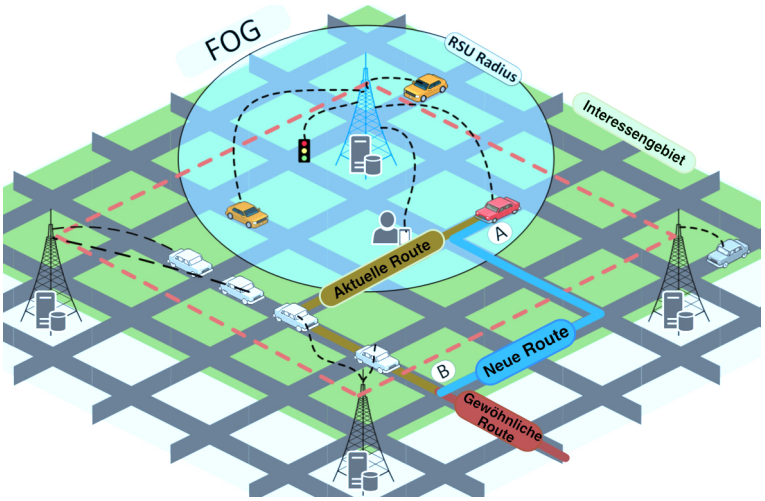


Abb. 1: Managementschema von FOX. Vgl. [ea15, S. 3]

3.4 Funktionsweise am Beispiel von FOX

Der Kommunikationsradius der RSUs hat direkten Einfluss auf die Kosten des Systems. Je kleiner der Radius, um so mehr Einheiten werden benötigt, um das geplante Areal netztechnisch abzudecken. Die Sammlung der Daten, die den Verkehrsfluss betreffen, werden von den RSUs von jedem Fog (Begriff analog zu Cloud) übernommen. Um das Verhalten der Fahrzeuge im entsprechenden Gebiet nachvollziehen zu können, senden diese periodisch Informationen wie Position, Geschwindigkeit, aktuelle Route, Fahrrichtung sowie die benötigte Zeit, um die jeweilige Straße auf der Route zu durchfahren über IEEE802.11p oder den GPRS-Nachfolger Long Term Evolution (LTE) an die RSU.
Nach einer Datenerfassung, folgt die Datenverarbeitung, und schließlich werden die Informationen an die umliegenden RSUs gesendet, die im Interessengebiet (IG) liegen. Vgl. Abb. 1. Die Größe der IG beeinflusst die Leistung in Effektivität in Bezug auf Wissen über die Karte die die RSU hat, und je kleiner die Karte, desto schneller die Verarbeitungszeit des Staukontrollsystems.
Das umleiten der Fahrzeuge wird periodisch ausgeführt, so dass jedes Fog-Netzwerk nur für die Umleitung derjenigen Fahrzeuge verantwortlich ist, die im jeweiligen Kommunikationsradius seiner RSU ist. So werden Fahrzeuge nur in Reichweite ihres IG umgeleitet. Ein Handoff zwischen Fahrzeug und RSU ist wegen dem periodischen Datenaustausch mit dem Fog-Netzwerk nicht nötig. Ist das Fog-Netzwerk also in der Routenplanung, empfangen die Fahrzeuge, die zu der Zeit Daten gesendet haben, eine neue Route, und das Fog nutzt die Gewichtung der Straße um eine Stauprognose zu berechnen. Danach führt das Fog-Netwerk eine Auslastungskontrolle durch, indem es Fahrzeuge auf Alternativrouten umleitet, und so die Auslastung der Straße reduziert. [ea15, S. 3] Während dieses Prozesses wird für jedes Fahrzeug eine Anzahl an Alternativrouten berechnet und mit Hilfe der Boltzmann-Statistik [SK83] ausgewählt. Somit wird ein erneuter Stau mit Fahrzeugen, deren Ziel in derselbe Richtung liegt, verhindert. Darauf folgt eine Evaluation, ob die Größe dieser Router im Algorithmus größer oder kleiner als die der neuen Route ist, und deren Auswirkung auf die Systemperformance be-

```
Input  : N // Set of vehicles within the radius of each RSU
         1 G // Graph created by each FOG (area of interest)
         2 K // number of alternate paths
         3 ACC // percentage of size of the new route
4  foreach v ∈ N do
       // sets of edges that make up the route of vehicles v
5      route ← v.getRoute();
       // returns the current edge of vehicle v to the last edge of the path
          of vehicle v contained in graph G
6      source ← v.getPosition();
7      lastEdge ← G.getLastEdge(route);
       // calculates the k shortest paths of the source point to lastEdge
          for vehicle v
8      alternativeRoutes ←
       G.getKShortestPaths(source, lastEdge, K);
       // selects a path from the set of alternate paths, concatenates the
          remainder of the old path to the new path and assigns the new path
          to vehicle v
9      newRoute ← boltzmann(alternativeRoutes, G);
10     interestEdges, remaningEdges ←
       route.split(lastEdge);
11     if routeSizeRelation(newRoute, interestEdges, ACC)
       then
12         if lastEdge! = route.getDestination() then
               // returns the remainder of the route of the vehicle v from
                  the lastEdge
13             newRoute.add(remaningEdges);
14             v.setRoute(newRoute);
15         end
16         else
17             v.setRoute(newRoute);
18         end
19     end
20 end
```

Abb. 2: Algorithmus von FOX. Vgl. [ea15, S. 4]

urteilt. Die endgültige Wahl der Route hängt dann von dieser Größe in Abhängigkeit von deren der anderen Alternativrouten ab. [ea15, S. 3]
Generell betrachtet besteht ein IVS aus 3 Schichten: Anwendungsschicht, Vermittlungsschicht und Erfassungsschicht, wobei die Anwendungsschicht noch aus den 4 Subsystemen intelligentes Verkehrsmanagement, intelligentes Fahrermanagement, Informationssammlung und Überwachung sowie Informationsdienst besteht.
Hauptfunktionen der Anwendungsschicht sind das Sammeln, Speichern und Auswerten der Verkehrsdaten für einen Mehrwertdienst, um diese den Nutzern zur Verfügung zu stellen, sowie die Analyse der empfangenen Daten der Erfassungsschicht. Die Vermittlungsschicht besteht aus verschiedensten privaten Netzwerken, dem Internet, kabelgebundenen und kabellosen Netzwerken, Netzwerkmanagementsystemen, GPS, etc. Zum Sammeln der Daten nutzt die Erfassungsschicht Sensoren wie RFID, drahtlose Sensornetzwerke, Kameras und intelligente Terminals, um Daten der mobilen Objekte zu anderen Sensoren zu übertragen. [Ks15]

Schicht	Subsystem 1	Subsystem 2	Subsystem 3	Subsystem 4
Anwendungsschicht	Intelligentes Verkehrsmanagement	Intelligentes Fahrermanagement	Informationssammlung und Überwachung	Informationsdienst
Vermittlungsschicht	Internet	WiFi, 3G/4G	WiMax	GPS, GPRS
Erfassungsschicht	RFID	RFID Reader	WSN	Intelligente Terminals

Tab. 1: Vgl. [Ks15, S. 39]

3.5 Vor- und Nachteile

Vorteile Durch die Reduzierung der Datenmenge und die im Vergleich zu herkömmlichen Systemen günstigere Infrastruktur lässt sich mit relativ geringem Einsatz der Verkehrsfluss besser verteilen. Bei der Simulation verschiedener Metriken ergaben sich eine Reduzierung der Fahrzeit um 32 % sowie ein Rückgang von 59 % der Standzeit. [ea15]

Nachteile Größter Nachteil eines intelligenten Verkehrsmanagementsystems ist der Umgang mit sensiblen Daten. Durch Angriffe auf ein IVS können neben hohen wirtschaftlichen auch gesundheitliche Schäden entstehen. Weiter wächst durch die Digitalisierung der Verkehrssteuerung die Anzahl der Komponenten in einem System, die ausfallen können.

3.6 Datenschutz

Ein wichtiger Bestandteil eines intelligenten Verkehrsmanagementsystems ist die Absicherung der Infrastruktur vor unbefugten Zugriffen und Manipulationen wie auch der Datenschutz der Verkehrsteilnehmer.
Daten der Nutzer sollten möglichst keinen Personenbezug aufweisen und „Privacy by default“ herrschen. Hierzu wäre es von Vorteil, die erfassten Rohdaten schon fahrzeugintern

aufzuarbeiten. Wenn für den jeweiligen Zweck kein Personenbezug erforderlich ist, sollten die Daten anonymisiert werden. Ein Schutz vor externen Attacken bei der Kommunikation muss unter allen Umständen gegeben sein. Datenbasierte Dienste dürfen nur im notwendigen Umfang auf personenbezogene Daten zugreifen. [Vo17]

4 Smart Grid

4.1 Was ist ein Smart Grid?

Allgemein wird unter dem Begriff **Smart Grid** den Einbau von Computer- und Kommunikationstechnik in alle relevanten Komponenten eines Stromnetzes zum Zwecke der Fernüberwachung und -steuerung - insbesondere der Verbraucher - und einer Automatisierung derselben um die Flexibilität, Robustheit und Effizienz zu verbessern. Ein Schlüsselelement sind vernetzte Stromzähler, welche mit Zwei-Wege-Kommunikationstechnik und Fernwirksystemen ausgerüstet sind. [oE]

4.2 Abgrenzung

Das Smart Grid umfasst das Stromnetz mit seinen Kraft- und Speicherwerken, Vermittlungs-, Schalt- und Verteilsystemen bis zum vernetzten Stromzähler (welcher auch zum Smart Home gehört). Elektroautos und -Ladestationen, deren Akkus als Puffer sekundärbenützt werden, können für die Dauer der Verbindung ebenfalls dazugezählt werden.

4.3 Hintergrund

Die Umstellung auf erneuerbare Energieträger erhöht unteranderem die Volatilität des Leistungsangebots (z. B. Solar- und Windkraft), was das zeitlche Ungleichgewicht zwischen Angebot und Nachfrage verschärfen kann (siehe Abbildung 4.3).

Ausgleichsoptionen sind Zwischenspeicherung (z. B. in Pumpspeicherwerken), Regelenergiereserven, Anpassung des Angebots (Leistungsregelung anderer Kraftwerke) sowie Anpassung der Nachfrage. Die gröbste Form des Letzteren ist ein manueller oder teilautomatischer Lastabwurf, welcher jedoch größere Stromausfälle und Folgestörungen beinhalten kann.

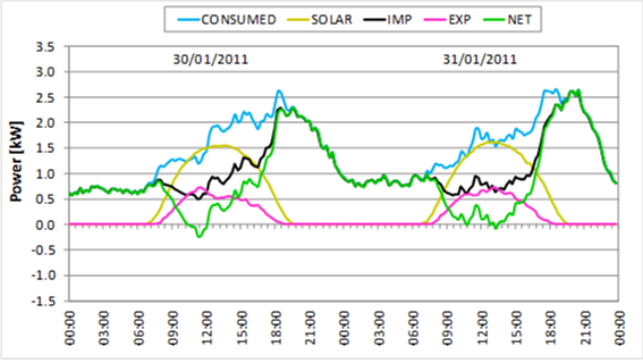


Abb. 3: Beispiel für die Unterschiede zwischen Produktion und Verbrauch [oSASoE17]Sek. 13

4.4 Aufbau

Das Smart Grid besteht aus über ein Kommunikationsnetz verbundenen Sensoren (z. B. Stromzähler), Aktoren (z. B. Schalter) sowie Auswertungs- und Steuereinheiten. Die Kommunikation kann dabei über Funk (z. B. LoRa [Kn16], GPRS und Nachfolger), über die Stromleitung selbst (PLC - Powerline Communication) oder über einen vorhandenen Internetanschluss im Haushalt (DSL, DOCSIS, FTTH, Ethernet, WiFi) erfolgen.
Vorteilhaft ist hierbei eine Fog-Computing-Architektur, da eine vereinzelte lokale Serverstörung weniger Teilnehmer betrifft als eine Störung an einer zentralen Stelle [Bo12]. Da Zuverlässigkeit bei Stromnetzen schon immer eine sehr hohe Priorität hatte, sind auch „dumme“ Stromnetze redundant konzipiert.

4.5 Funktionsweise am Beispiel mit OSGP vernetzter Stromzähler

Bei einem Smart Meter können werden die Verbrauchs-Messdaten in Echtzeit (fast beliebig fein getaktet) an eine Gegenstelle des Stromversorgers geschickt werden, die den Stromzähler mit Informationen über Preis und Auslastung versorgt. Zudem bietet der vernetzte Stromzähler dem Versorger eine Möglichkeit zur Fernschaltung um Lasten individuell abzuwerfen. Optional kann die Last-/Preisinformation zur Steuerung vernetzter Haushaltsgeräte benützt werden. [ET16]

Ein in Europa übliches Protokoll ist das **Open Smart Grid Protocol** (OSGP) [ET16].
Dessen Protokollstack basiert nicht auf IP, sondern benützt das Inter-Domain-Routing-freie LonTalk [EC14] mit dessen eigenem hierarchischen Adressierungsschema sowie Powerline (PLC) als physische Schicht. [Oa16]
Die Anwendungsschicht folgt dabei einem Client-Server-Schema, wie es in Abbildung 4.5 dargestellt wird. Der „Data Concentrator“ (entspr. Fog-Knoten) arbeitet hierbei als Client; die überwachten/gesteuerten Geräte (z. B. Smart Meter) als Server, wobei Proxies die Nachrichten weiterleiten können. [ET16, S. 18f]

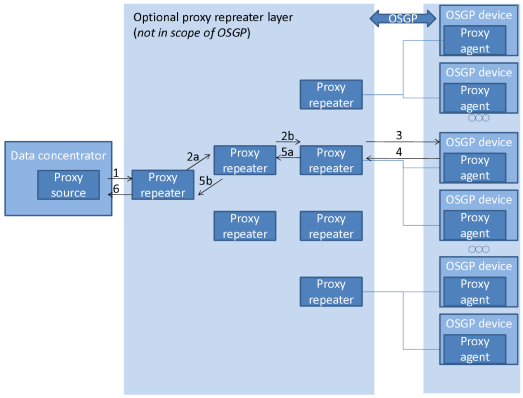


Abb. 4: Kommunikationsmodell von OSGP auf Layer 7 [ET16, S. 18]

Dabei werden in Tabellen angeordnete „Variablen“ gelesen bzw. geschrieben; zudem können vordefinierte Prozeduren aufgerufen werden.

Weiter können OSGP-Meter dem Versorger u. a. folgende Funktionen bieten:

1. Preisänderung in Echtzeit („manual override“) [ET16, S. 33]
2. Tarifauswahl basierend auf einem vorprogrammierten Kalender [ET16, S. 32f]
3. Überleistungstarif bei Überschreitung einer Leistungsgrenze [ET16, S. 33]
4. automatischer und manueller Lastabwurf [ET16, S. 43f]
5. Fernabfrage gesammelter Daten [ET16, S. 43f]
6. detaillierte Aufzeichnung von Messwerten (z. B. Verbrauch) [ET16, S. 35]

OSGP kann als Fog-Computing bezeichnet werden, da der „Data Concentrator“ bzw. dessen Proxies im lokalen Powerline-Netz liegen - mit der Besonderheit, dass die Rollen von Client und Server vertauscht sind (Server bei der „Kundschaft“).

4.6 Vor- und Nachteile

Vorteile Durch variable Preise kann der Stromversorger die Kundschaft zu einem für das Netz optimalen Verbrauchsprofil motivieren: Falls möglich, werden zeitlich unkritische „Jobs“ auf günstige Zeiträume verschoben. Umgekehrt kann die Kundschaft von den variablen Strompreisen an der Börse profitieren.
Wenn der Zähler vom Kunden automatisert (über andere Protokolle) ausgelesen werden kannn, kann dies sogar z. B. über Smart-Home-Protokolle - automatisiert werden.
Durch mehr genauere Messdaten kann der Stromversorger mittels Machine Learning Informationen über den Stromverbrauch gewinnen, Vorhersagen treffen und anhand derer sein Grid weiter optimieren. Der physische Zugangsschutz der Stromleitungen (gegen „Stromdiebstahl“) zwischen Zähler und Versorger schützt implizit auch die Datenübertragung zwischen Zähler und Gegenstellen.

Nachteile Aus den Verbrauchsdaten lassen sich offensichtlich detaillierte Informationen über das Privatleben der Verbraucher herleiten (z. B. Tagesablauf, Urlaub) - sogar die Identifikation Fernsehsendungen ist möglich [JL11].
Auch konnten bei manchen handelsüblichen Smart Metern Messfehler von bis 582 % bei bestimmten Lasttypen festgestellt werden [LKM, S. 6], woraus neben einem finanziellen Schaden für eine Partei (meist der Verbraucher) auch eine Destabilisierung des Stromnetzes eintreten kann (aus falscher Regelleistung resultierendes Ungleichgewicht von Angebot und Nachfrage).
Zudem erhöht Digitalisierung allgemein die Komplexität eines Systems und mit der Anzahl der potenziellen Fehlerquellen die Gesamtanfälligkeit.
Nicht zuletzt können Fernabschaltungen bei kritischen Geräten erhebliche Sach- oder sogar Personenschäden verursachen (Beatmungsgeräte als Extrembeispiel).

5 Zusammenfassung

In dieser Arbeit wurde zunächst einen Überblick über die Architektur von Fog-Computing gegeben und deren Vor- und Nachteile, sowie Herausforderungen erläutert. Hierbei wurde ein Vergleich mit Cloud-Computing-Netzwerken gezogen. Darauf folgte die Vorstellung des Anwendungsszenario Smart Traffic und dessen Begriffserklärung. Am Beisiel des Fast Offset XPath Systems wurde die Architektur und Funktionsweise von intelligenten Verkehrsmanagementsystemen eingeführt, sowie Vor- und Nachteile vorgestellt. Weiter wurde für den Umgang mit sensiblen Nutzerdaten Empfehlungen gegeben. Am Beispiel von Smart Grid wurde eine weitere Einsatzmöglichkeiten von Fog-Computing dargestellt, und wie dies helfen kann Probleme wie Lastspitzen im Stromnetz zu lösen. Weiter wurde die Funktion von vernetzter Stromzähler erläutert und die Vor- sowie Nachteile aufgezählt und die mit Fog-Computing einhergehende Herausforderungen Privacy und Security aufgezeigt.

In Kapitel 2 wurde zunächst das Konzept Fog-Computing vorgestellt und dessen Vor- und Nachteile aufgezeigt. Dann wurde in Kapitel 3 die Anwendung von Fog-Computing in intelligenten Verkehrsleitsystemen anhand von FOX sowie in Kapitel 4 in intelligenten Stromnetzen anhand von OSGP präsentiert.

Literaturverzeichnis

[AS15] Al-Sakran, Hasan Omar: Intelligent Traffic Information System Based on Integration of Internet of Things and Agent Technology. International Journal of Advanced Computer Science and Applications, 6, jan 2015. http://dx.doi.org/10.14569/ijacsa.2015.060206 (aufgerufen am 11.6.2017).

[Bo12] Bonomi, Flavio; Milito, Rodolfo; Zhu, Jiang; Addepalli, Sateesh: Fog Computing and Its Role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. MCC '12, ACM, New York, NY, USA, S. 13–16, 2012. http://doi.acm.org/10.1145/2342509.2342513 (aufgerufen am 7.6.2017).

[C15] C, Venkatesh H; Shrivatsa D Perur; Jagadish M: Application of RFID Technology and the Maximum Spanning Tree Algorithm for Solving Vehicle Emissions in Cities on Internet of Things. International Journal of Electrical and Electronics Research, 3(1):218–223, mar 2015. https://www.researchgate.net/profile/Shrivatsa_Perur/publication/313841738_An_Approach_to_Make_Way_for_Intelligent_Ambulance_Using_IoT/links/58a9991f4585150402ff138a/An-Approach-to-Make-Way-for-Intelligent-Ambulance-Using-IoT.pdf (aufgerufen am 11.6.2017).

[ea15] et al., Celso Brennand: FOX: A Traffic Management System of Computer-Based Vehicles FOG. International Journal of Grid Distribution Computing, 8(3):201–206, 2015. https://www.researchgate.net/profile/Leandro_Villas/publication/306304740_FOX_A_traffic_management_system_of_computer-based_vehicles_FOG/links/57b9bdbe08aedfe0ec96e825/FOX-A-traffic-management-system-of-computer-based-vehicles-FOG.pdf (aufgerufen am 11.6.2017).

[ea17] et al., Z. Hao: Challenges and Software Architecture for Fog Computing. IEEE Internet Computing, 21(2):2–4, 2017. http://www.cs.wm.edu/~liqun/paper/IC17.pdf (aufgerufen am 7.6.2017).

[EC14] Echelon-Corporation: , Open Data Communication in Building Automation, Controls and Building Management - Control Network Protocol - Part 1: Protocol Stack, 2014. EN 14908-1:2014.

[ET16] ETSI: , Open Smart Grid Protocol (ETSI TS 104 001 V2.1.1), 2016. http://www.etsi.org/deliver/etsi_ts/104000_104099/104001/02.01.01_60/ts_104001v020101p.pdf (aufgerufen am 7.6.2017).

[JL11] Justus, U. Greveler; B.; Löhr, D.: , Hintergrund und experimentelle Ergebnisse zum Thema „Smart Meter und Datenschutz“, 2011. http://1lab.de/pub/smartmeter_sep11_v06.pdf (aufgerufen am 11.6.2017).

[Kn16] Knight, Matt: , Reversing LoRa, 2016. https://static1.squarespace.com/static/54cecce7e4b054df1848b5f9/t/57489e6e07eaa0105215dc6c/1464376943218/Reversing-Lora-Knight.pdf (aufgerufen am 7.6.2017).

[Ks15] Kshirsagar, Sonali P.; Mantala, Priyanka H.; Parjane, Gayatri D.; Teke, Kalyani G.: Intelligent Traffic Information System Based on Integration of Internet of Things and Agent Technology. International Journal of Computer Applications, 6(2):37–42, 2015. http://thesai.org/Downloads/Volume6No2/Paper_6-Intelligent_Traffic_Information_System_Based.pdf (aufgerufen am 11.6.2017).

[LKM] Leferink, Frank; Keyer, Cees; Melentjev3, Anton: , Static Energy Meter Errors Caused by Conducted Electromagnetic Interference. http://doc.utwente.nl/103993/1/2016EMCMag-Static_Energy_Meter_Errors_Caused_by_Conducted_Electromagnetic_Interference_-_proof.pdf (aufgerufen am 11.6.2017).

[Oa16] OSGP-alliance: , technical information, 2016. http://www.osgp.org/en/technical (aufgerufen am 7.6.2017).

[oE] of Energy, United States Department: , Grid Modernization and the Smart Grid. https://www.energy.gov/oe/services/technology-development/smart-grid (aufgerufen am 11.6.2017).

[oSASoE17] of South Australia School of Engineering, University: , Lochiel Park Research, 2017. http://www.unisa.edu.au/IT-Engineering-and-the-Environment/School-of-Engineering/Research/Research-Node-for-Low-Carbon-Living/Lochiel-Park-Research/ (aufgerufen am 7.6.2017).

[SK83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi et al.: Optimization by simulated annealing. Science, 220(4598):671–680, 1983. https://pdfs.semanticscholar.org/beb2/1ee4a3721484b5d2c7ad04e6babd8d67af1d.pdf (aufgerufen am 7.6.2017).

[SS17] Sahoo, K.S.; Sahoo, B.: SDN Architecture on Fog Devices for Realtime Traffic Management: A Case Study. In (Lobiyal D., Mohapatra D., Nagar A. Sahoo M. (eds), Hrsg.): Proceedings of the International Conference on Signal, Networks, Computing, and Systems. Jgg. 395 in Lecture Notes in Electrical Engineering, Springer, New Delhi, 2017. http://dspace.nitrkl.ac.in/dspace/bitstream/2080/2510/1/2016%20ICSNCS_SahooKS.pdf (aufgerufen am 11.6.2017).

[Vo17] Datenschutzrechtliche Empfehlungen der Bundesbeauftragen für Datenschutz und Informationsfreiheit zum automatisierten und vernetzten Fahren, https://www.bfdi.bund.de/SharedDocs/Publikationen/Allgemein/DatenschutzrechtlicheEmpfehlungenVernetztesAuto.pdf?__blob=publicationFile (aufgerufen am 11.6.2017).

Danksagung Unser besonderer Dank gilt unserem Betreuer Herrn Bastian Kemmler für die kompetente Beratung.

Autorenverzeichnis