TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur Betriebssysteme

# Diplomarbeit

## Untersuchung von Methoden zur Laufzeitmessung in Wireless LAN Netzwerken zum Zwecke der Positionsbestimmung

Mario Haustein

Chemnitz, 16. März 2011

**Betreuer:** Prof. Dr.-Ing. habil. Matthias Werner
**Gutachter:** Prof. Dr.-Ing. habil. Matthias Werner
Dr.-Ing. Jörg Anders

***In tiefer Dankbarkeit.***

*Herta Schuster (†Juli 2007)*
*Hilde Haustein (†August 2007)*

# Aufgabenstellung

## Untersuchung von Methoden zur Laufzeitmessung in Wireless LAN Netzwerken zum Zwecke der Positionsbestimmung

Ortsbasierte Dienste erfreuen sich in den letzten Jahren starker Beliebtheit. Für deren Umsetzung sind sog. Lokalisierungsdienste notwendig, welche eine Ortung von Mobilgeräten erlauben. Das GPS stellt den wohl populärsten Lokalisierungsdienst dar, ist jedoch innerhalb von Gebäuden nur sehr beschränkt einsetzbar. In der Vergangenheit wurden deshalb Methoden vorgeschlagen, die zur Positionsbestimmung auf die Messung der Empfangsfeldstärke von WLAN-Aussendungen zurückgreifen.

Im Rahmen der Diplomarbeit soll untersucht werden, ob sich ebenfalls eine Postionsbestimmung anhand von Laufzeiten der WLAN-Signale umsetzen lässt. Bedingung hierbei ist, dass der Lokalisierungsdienst

- eine reine Softwarelösung darstellt und keine Modifikationen an Hard- oder Firmware voraussetzt und
- die Lokalisierung ohne für diese Zwecke ausgelegte Spezialhardware umsetzbar ist.

Diese Anforderungen sollen sicherstellen, dass der zu entwickelnde Lokalisierungsdienst mit bereits installierter, handelsübliche Hardware umsetzbar ist. Es sind in Frage kommende Verfahren zur Bestimmung der Signallaufzeit zu erörtern. Für die Laufzeitmessung in Frage kommenden Zeitquellen sollen zugänglich gemacht und auf ihre Tauglichkeit untersucht werden. Durch Messreihen ist zu untersuchen, ob mit den vorgeschlagenen Messverfahren eine Lokalisierung möglich ist und in welchem Rahmen sich die zu erwartende Genauigkeit bewegt. Die in dieser Arbeit beschriebenen Konzepte sollen im Rahmen einer Proof of Concept Anwendung implementiert werden. Die Software soll unter dem Gesichtspunkt der Wiederverwendbarkeit entwickelt werden, um eine spätere Nutzung im Rahmen anderer Projekte zu ermöglichen.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

# Abkürzungsverzeichnis

| | |
|---|---|
| **ACK** | Bestätigung (Acknowledgement) |
| **AP** | Access Point |
| **BSS** | Basic Service Set |
| **CTS** | Clear to send |
| **DA** | Zieladresse (Destination Address) |
| **DCF** | Verteilte Koordinierung (Distributed Coordination Function) |
| **DIFS** | DCF Interframe Space |
| **DS** | Distribution System |
| **DSSS** | Direct Sequence Spread Spectrum |
| **ESS** | Extended Service Set |
| **FHSS** | Frequency Hopping Spread Spectrum |
| **HPET** | High Precision Event Timer |
| **IBSS** | Independent Basic Service Set |
| **LLC** | Logical Link Control Layer |
| **MAC** | Medienzugriffsschicht (Medium Access Control Layer) |
| **MLME** | MAC Layer Management Entity |
| **OFDM** | Orthogonal Frequency-Division Multiplexing |
| **PCAP** | Packet Capture library |
| **PCF** | Zentrale Koordinierung (Point Coordination Function) |
| **PDU** | Protocol Data Unit |
| **PHY** | Darstellungsschicht (Physical Layer) |
| **PIFS** | PCF Interframe Space |
| **PLME** | Physical Layer Management Entity |
| **RA** | Empfängeradresse (Receiving Station Address) |
| **RTS** | Request to send |
| **RX** | Empfänger, empfangen |
| **SA** | Quelladresse (Source Address) |

**SDU** Service Data Unit
**SIFS** Short Interframe Space
**TA** Senderadresse (Transmitting Station Address)
**TDOA** Time Difference of Arrival
**TOA** Time of Arrival
**TOF** Time of Flight
**TSC** Time Stamp Counter
**TSF** Timing Synchronization Function
**TX** Sender, senden
**VIF** Virtual Interface

# Symbole und Bezeichner

| | |
|---|---|
| $c, \varepsilon, \mu$ | Lichtgeschwindigkeit, Permittivität und Permeabilität eines Mediums |
| $c_0, \varepsilon_0, \mu_0$ | $c, \varepsilon$ und $\mu$ im Vakuum |
| $n = c_0/c$ | Brechungsindex eines Mediums |
| $\lambda$ | Wellenlänge |
| $r, d$ | geometrischer Abstand |
| $\vec{r}$ | Ortsvektor in Bezug auf ein Koordinatensystem |
| $\|\vec{r}\|$ | euklidsche Norm von $\vec{r}$ |
| $x, y, z$ | Komponenten eines dreidimensionalen Vektors |
| $t$ | Signallaufzeit |
| | |
| $L$ | Länge einer IEEE 802.11 MAC-PDU in Bits |
| $R$ | IEEE 802.11 Datenrate in MBit/s |
| | |
| $A$ | Station, die eine Laufzeitmessung auslöst |
| $B$ | Station, die auf $A$ antwortet |
| $M$ | Station, die den Informationsaustausch zwischen $A$ und $B$ beobachtet |
| $r_{X,Y}$ | Entfernung zwischen den Stationen $X$ und $Y$ |
| $t_{X,Y}$ | Signallaufzeit zwischen den Stationen $X$ und $Y$ |
| | |
| $d$ | Übertragungsdauer eines IEEE 802.11 Frames |
| $g$ | Zeit zwischen Anfrage und Antwort einer Frame-Sequenz |
| $\Delta$ | Gesamtdauer eines Time-of-Flight-Verfahrens |
| $T$ | Ereignis bzw. dessen Globalzeit |
| $\widetilde{T}$ | Der Zeitstempel einer Uhr bei Eintreten von $T$ |
| $\widetilde{\Delta}$ | Die Zeitstempeldifferenz der Ereignisse, die $\Delta$ festlegen |
| $\widehat{\Delta}$ | Eine Schätzung für $\Delta$ basierend auf $\widetilde{\Delta}$ |
| | |
| $E[X]$ | Erwartungswert der Zufallsgröße $X$ |
| $D^2[X]$ | Varianz der Zufallsgröße $X$ |
| $\mu$ | empirischer Mittelwert einer Stichprobe |
| $\sigma$ | empirische Standardabweichung einer Stichprobe |
| $\lfloor x \rfloor$ | $x$ ganzzahlig abgerundet |
| $\lceil x \rceil$ | $x$ ganzzahlig aufgerundet |
| $\langle x \rangle$ | Nachkommaanteil von $x$ |

# Kapitel 1

# Einleitung

Neben der Informationsübertragung ist die Funknavigation ein wesentliches Anwendungsgebiet der drahtlosen Nachrichtentechnik und bereits seit langer Zeit Gegenstand der praktischen Anwendung. Die Navigationssysteme LORAN[1] und DECCA wurden bereits während des zweiten Weltkriegs für die Luft- und Seefahrt entwickelt und erlaubten eine Positionsbestimmung mit einer Genauigkeit von bis zu 100 m. Mit VOR und DME stehen in der Luftfahrt weit verbreitete Systeme zur Kursbestimmung und Entfernungsmessung zur Verfügung. Diese werden zunehmend durch satellitengestützte Navigationssysteme verdrängt. Prominentestes Beispiel ist NAVSTAR GPS (kurz GPS)[2], welches eine globale Abdeckung bietet und eine horizontale Genauigkeit von bis zu 4 m erreicht. GPS ist allerdings nur ein spezieller Vertreter einer ganzen Reihe von Satellitennavigationssystemen, in die sich z. B. GLONASS (russ. ГЛОНАСС)[3], Galileo[4] und Compass einreihen bzw. zukünftig einreihen werden.

Durch die Verfügbarkeit verhältnismäßig kostengünstiger und empfindlicher Empfänger hat GPS in den vergangen zehn Jahren stark an Popularität gewonnen. GPS-Empfänger können heutzutage einschließlich Antennen leicht in Mobilgeräte wie Handies, Smartphones, Kameras, PDAs oder Netbooks integriert werden. Erst damit wird die Möglichkeit eröffnet, eine Reihe von standortbezogenen Diensten (engl. Location Based Services, LBS) umzusetzen. Standortbezogene Dienste zeichnen sich dadurch aus, dass die Kenntnis über den Aufenthaltsort des Nutzers oder des Mobilgeräts wesentlich für die Diensterfüllung ist. Folgenden Szenarien sind Anwendungen von LBS.

**Indoor-Navigation** Wo befinde ich mich derzeit im Gebäude? Welchen Weg muss ich nehmen, um zum Ziel $Z$ zu gelangen?

**Ortung** Ein Smartphone ist verloren gegangen, kann aber noch über eine Drahtlosverbindung erreicht werden. Auf welches Gebiet sollte die Suche beschränkt werden?

**Elektronischer Stadtführer** Blende historische Informationen über die Sehenwürdigkeiten ein, die sich in unmittelbarer Nähe meines Standorts befinden.

[1] `http://www.navcen.uscg.gov/?pageName=loranMain`
[2] `http://www.navcen.uscg.gov/?pageName=GPS`
[3] `http://www.glonass-ianc.rsa.ru/pls/htmldb/f?p=202:1:3687279510959123`
[4] `http://www.esa.int/esaNA/galileo.html`

**Fußgänger-Navigation** Wo ist der/die/das nächste Bushaltestelle, Bankautomat, Arzt, WC, ... ?

Bereits bei den ersten beiden Beispielen werden die Grenzen von Satellitennavigationssystemen deutlich. Sie sind aufgrund der großen Signaldämpfung nicht innerhalb von Gebäuden anwendbar. Eine potentielle Alternative stellt in diesem Fall Wireless LAN (kurz WLAN) dar. Sendeleistung und Frequenzspektrum von WLAN-Geräten sind so ausgelegt, dass Wände weitestgehend durchdrungen werden können. Zudem ist in größeren Einrichtungen wie Universitäten, Krankenhäusern und Firmenzentralen eine flächendeckende WLAN-Versorgung gegeben. Satellitennavigationssysteme ermöglichen ferner nur dem Mobilgerät eine Positionsbestimmung. Die Ortung anderer Teilnehmer durch ein Satellitennavigationssystem ist prinzipbedingt unmöglich, da nur Daten vom Satellit zum Empfänger übertragen werden. Dank der symmetrischen Beziehung zwischen WLAN-Stationen besteht die Möglichkeit, auch die Fremdlokalisierung umsetzen zu können.

Mit der Ausnahme von VOR, welches allein zur Kursbestimmung dient, basieren die oben genannten Systeme auf der Tatsache, dass sich elektromagnetische Wellen mit der Lichtgeschwindigkeit $c$ ausbreiten. Durch Auswertung von Signallaufzeiten oder Laufzeitdifferenzen kann auf die Weglänge geschlossen werden, die eine Welle zurückgelegt hat. Anhand der Weglängen zwischen Mobilgerät und bekannten Referenzpunkten kann letztendlich eine absolute Positionsangabe berechnet werden.

Thema der vorliegenden Arbeit ist es deshalb, Untersuchungen anzustellen, ob es auch mittels handelsüblicher Wireless-LAN-Hardware möglich ist, Signallaufzeiten zu messen und für die Lokalisierung heranzuziehen. Es sollen dazu die entsprechenden Modifikationen am Betriebssystem und den WLAN-Treibern vorgenommen werden. Alle relevanten Messabläufe sind in einem Framework zu implementieren, auf welches später andere Lokalisierungssysteme zurückgreifen können.

## 1.1 Warum Signallaufzeit, warum WLAN?

Zunächst stellen sich jedoch bei genauerer Betrachtung der Aufgabenstellung zwei Fragen.

1. Warum soll Wireless LAN zur Lokalisierung eingesetzt werden, wo doch bereits bestehende Lösungen für andere Funkstandards verfügbar sind [FFH09] bzw. spezielle Lokalisierungsstandards[5] existieren?
2. Warum sollen Signallaufzeiten zur Lokalisierung genutzt werden?

Die erste Frage ist recht schnell dadurch zu beantworten, dass Wireless LAN nach IEEE 802.11 der De-facto-Standard für lokale Funknetze ist. WLAN-Adapter sind heutzutage in fast jedem Mobilgerät eingebaut. Darüber hinaus ist WLAN-Hardware durch Massenproduktion preiswerter als auf die Lokalisierung ausgelegte Spezialprodukte. WLAN-Lokalisierung bietet durch die weite Verbreitung somit ein enormes Anwendungspotential. Ein weiterer Aspekt in Funknetzen ist der Störpegel, der durch andere Sender im selben Frequenzband hervorgerufen wird. Ein Lokalisierungssystem, welches parallel zu Wireless LAN in den lizenzfreien ISM-

[5] vgl. `http://www.nanotron.com/EN/index.php`

Bändern betrieben wird, kann zu einem Einbruch der Nettodatenrate führen oder selbst durch WLAN gestört werden.

Signallaufzeiten sind für die Lokalisierung interessant, da sie sehr eng mit der Entfernung zwischen Sender und Empfänger korreliert sind. Die Signallaufzeit $t$ und die Entfernung $r$ zwischen zwei Stationen sind über die einfache Gleichung

$$r = c \cdot t$$

miteinander verknüpft. Die Konstante $c$ steht dabei für die Lichtgeschwindigkeit im Übertragungsmedium, mit der sich die Signale von WLAN-Aussendungen ausbreiten. Der Zusammenhang ist linear. Die Unsicherheit von $r$ hängt demnach nur von der Unsicherheit der Zeit $t$ ab, nicht aber von deren eigentlicher Größe. Die Feldstärke nimmt im Fernfeld quadratisch mit der Entfernung ab. Ein Messfehler der Feldstärke hat folglich in großen Entfernungen eine viel stärkere Auswirkung als ein gleich großer Messfehler für einen kleineren Wert von $r$. Im Gegensatz dazu gibt es für die Beschreibung der Signaldämpfung eine Vielzahl komplexer Modelle, die sich je nach Einsatzgebiet zum Teil stark unterscheiden [SA07]. Bedingt durch die kurzen Wellenlängen entstehen in bebauten Umgebungen Interferenzmuster, die bereits bei kleinen Abstandsänderungen zu einem starken Schwanken der Feldstärke führen. Für die Positionierung anhand von Feldstärkewerten wird dadurch meist eine Datenbank notwendig, die jedem Ort charakteristische Messwerte zuordnet. Dabei hat auch der Mensch durch seinen großen Wasseranteil im Körpergewebe einen nicht zu vernachlässigenden Einfluss auf die Signalausbreitung [KK04]. Diese Effekte lassen sich nur begrenzt modellieren.

Natürlich bleiben auch die Signallaufzeiten nicht unbeeinträchtigt von solchen Effekten. Durch Reflexion kann es zu einer nichtlinearen Ausbreitung der Wellen kommen, wodurch die gemessene Laufzeit den direkten Abstand zwischen den Stationen überschätzt. Weiterhin variiert die Lichtgeschwindigkeit je nach durchquertem Medium leicht. Dennoch verspricht die Messung von Signallaufzeiten einen interessanten Ansatz für einen Lokalisierungsdienst.

## 1.2 Bisherige Arbeiten und Ansätze

Die Idee, mobile Geräte mit Hilfe von Funknetzwerken zu lokalisieren, ist nicht neu, und es gibt bereits eine Reihe von Arbeiten auf diesem Gebiet. Einen nach Funkstandard und Lokalisierungsverfahren gegliederten Überblick über verschiedene Arbeiten zu diesem Thema geben die Autoren von [LDBL07].

Zumeist nutzen die verwendeten Systeme Identifikationsinformationen der empfangenen Funkzellen (z. B. SSID oder MAC-Adressen) bzw. werten die Empfangssignalstärke (auch als RSSI-Wert[6] bezeichnet) von in der Nähe befindlichen Access Points aus. Der RSSI-Wert hängt u. a. von der Entfernung zwischen Mobilgerät und Access Point ab. Eine der ersten Arbeiten auf diesem Gebiet war RADAR [BP00]. Bei Horus [YA05] und MagicMap[7] handelt es sich um vergleichbare Systeme, welche jedoch vornehmlich dem universitären Umfeld entstammen. Zudem wird der RSSI-Ansatz auch in kommerziellen Produkten wie Ekahau[8]

[6] Received Signal Strength Indication
[7] `http://www.magicmap.de/`
[8] `http://www.ekahau.com/`

oder der Wireless Location Appliance[9] des Netzwerkausrüsters Cisco verfolgt. Die eigentliche Positionsbestimmung erfolgt mittels einer als „Fingerprinting“ bezeichneten Technik. In einer Trainingsphase werden zunächst die Feldstärken an bestimmten Orten gemessen. In der späteren Lokalisierungsphase werden die gemessenen Feldstärken am unbekannten Ort mit den Referenzmessungen verglichen. Unternehmen wie Google[10] oder Skyhook[11] haben es sich zum Ziel gesetzt, die dafür notwendigen Datenbasen im städtischen Umfeld aufzubauen.

Auf Signallaufzeiten basierende Lokalisierungsverfahren werden im Gegensatz zu den oben genannten Ansätzen verstärkt im Umfeld der Forschung diskutiert und finden nur begrenzt praktische Anwendung. In [CBAI07] und [WKP09] wird die Immediate Acknowledge Funktion der IEEE 802.11 MAC-Schicht ausgenutzt, mit deren Hilfe dem Sender eines Datenpakets der erfolgreiche Empfang mitgeteilt wird. In beiden Arbeiten wird jedoch spezielle bzw. eigens modifizierte Hardware verwendet. Diese Technik wird ebenfalls in [LPLaY00] vorgeschlagen. Hauptaspekt dieser Arbeit ist allerdings der Vergleich zweier Modulationstechniken in Hinblick auf die Nutzbarkeit zur Lokalisierung. Die Autoren von [MDF$^+$00] und [VK04] gehen auf Schranken für die erzielbare Genauigkeit bei der Positionsbestimmung ein. Die erstgenannte Arbeit bezieht sich jedoch nicht speziell auf Wireless LAN und zieht somit nicht genauigkeitsbestimmende Effekte von WLAN-Hardware in Betracht. Im zweitgenannten Artikel werden lediglich Methoden betrachtet, die eine gemeinsame Zeitbasis für alle Teilnehmer des Lokalisierungsverfahrens erfordert. Sehr themennah zur vorliegenden Arbeit sind [GH05] und [HW08]. Die Autoren schlagen ein Verfahren zur Messung der Roundtrip-Time zwischen zwei WLAN-Geräten vor und messen die benötigte Zeit mittels eines 1 $\mu$s-Zählers, den jedes WLAN-Gerät besitzt. Durch statistische Auswertung sehr vieler Messungen lässt sich die Signallaufzeit im ns-Bereich bestimmen.

## 1.3 Anforderungen an einen WLAN-Lokalisierungsdienst

Eine Reihe von Anforderungen soll den Lösungsraum für das im Rahmen dieser Arbeit zu entwickelnde System sinnvoll einschränken. Diese Einschränkungen dienen als Leitfäden zur Lösungsfindung und stellen sicher, dass keine Möglichkeiten in Betracht gezogen werden, die nur eine geringe praktische Relevanz besitzen.

**Eigenlokalisierung** Das System soll einem Mobilgerät die Möglichkeit bereitstellen, seine Position durch Ausführung geeigneter Messungen zu bestimmen. Von allen beteiligten Komponenten – bis auf das Mobilgerät selbst – soll dabei keine Funktionalität gefordert werden, die über das übliche Maß hinausgeht. An einem Access Point sollen also beispielsweise keine Modifikationen vorgenommen werden, die ausschließlich für die Durchführung der beabsichtigten Messungen notwendig sind. Die Verfügbarkeit eines speziellen Servers ist nach Möglichkeit zu vermeiden.

**Fremdlokalisierung** Das System soll die Fähigkeit besitzen, ein Mobilgerät zu orten, ohne dass am Mobilgerät Modifikationen extra für diesen Zweck vorgenommen werden müssen.

[9] `http://www.cisco.com/en/US/prod/collateral/wireless/ps5755/ps6301/ps6386/product_data_sheet0900aecd80293728.pdf`

[10] `http://www.google.com/`

[11] `http://www.skyhookwireless.com/`

**Hardware-unabhängig** Die Positionsbestimmung soll mit einer breiten Palette von WLAN-Geräten möglich sein. Die Implementierung des Lokalisierungsdienstes soll somit nur auf Betriebssystemschnittstellen zurückgreifen und keine hardwarespezifischen Anforderungen verlangen.

**Hardware-generisch** Das System soll nicht auf WLAN-Hardware beschränkt sein, die über spezielle Sende- und Empfangseinrichtungen (z. B. software defined radios, SDR) verfügt, sondern mit marktüblichen Hardwarekomponenten funktionsfähig sein.

**Software-Lösung** Für die Durchführung der Messung sollen hardwareseitige Veränderungen, wie die Modifikation von Schaltungen, oder ein Austausch von Firmwarekomponenten keine Voraussetzungen sein. Die Implementierung soll sich rein auf Treibermodifikationen und Softwarekomponenten stützen.

**Koexistenz** Der Lokalisierungsdienst soll nicht zu einer Einschränkung dritter Dienste führen, die über ein vertretbares Maß hinausgeht.

**Integrierbarkeit** Der Lokalisierungsdienst soll die Grenzen bestehender Standards nicht überschreiten, wodurch schlimmstenfalls ein unerwünschtes Verhalten involvierter Kommunikationspartner hervorgerufen werden könnte.

## 1.4 Verwendete Hard- und Software

Aufgrund der in [Tim10] gemachten Erfahrungen soll in dieser Arbeit eine größere Auswahl von WLAN-Adpatern untersucht werden. Eine Auflistung der zur Verfügung stehenden Hardware ist im Anhang A enthalten. Um verschiedene Produkte mit vertretbarem Aufwand testen zu können, kamen hauptsächlich USB-WLAN-Adapater zum Einsatz.

Alle Versuche wurden auf Linux-Systemen durchgeführt, da hier sehr einfach Modifikationen an den WLAN-Treibern vorgenommen werden können. Zur Durchführung der Versuche kam ausschließlich Standardsoftware zum Einsatz, welche für alle gängigen Distributionen verfügbar ist. Folgende Versionen wurden dabei verwendet.

- Linux 2.6.35 (Gentoo r12)
- iw 0.9.21
- libpcap 1.1.1
- Wireshark 1.2.13
- Lua 5.1.4

## 1.5 Weiterer Aufbau dieser Arbeit

Im weiteren Verlauf dieser Arbeit werden in Kapitel 2 zunächst die wichtigsten Grundlagen zur Lokalisierung, WLAN-Technologie und der Konfiguration von WLAN-Geräten unter dem Betriebssystem Linux dargelegt. Kapitel 3 diskutiert anschließend, welche prinzipiellen Möglichkeiten zur Laufzeitmessung existieren. Es werden Realisierungsmöglichkeiten vorgeschlagen und verglichen. In Kapitel 4 werden zu den betrachteten Messverfahren praktische

Umsetzungsmöglichkeiten vorgeschlagen und auf die Auswertung der Messdaten eingegangen. Das letzte Kapitel beschreibt abschließend die Anwendung der entwickelten Methoden in Beispielsituationen.

# Kapitel 2

# Grundlagen

## 2.1 Ausbreitung elektromagnetischer Wellen

In einem homogenen Raum sendet ein Sender $S$ über eine elektromagnetische Welle ein Signal zum Empfänger $E$. Zwischen Aussendung und Empfang des Signals vergehe dabei die Zeit $t$. Dann ist die Entfernung $r$ zwischen $S$ und $E$ über folgende Beziehung mit $t$ verknüpft.

$$r = c \cdot t \tag{2.1}$$

Die Größe $c$ steht dabei für die Lichtgeschwindigkeit und ergibt sich anhand von Gleichung (2.2) aus der Permittivität $\varepsilon$ und der Permeabilität $\mu$ des Mediums, in dem sich diese Welle ausbreitet.[1]

$$c = \frac{1}{\sqrt{\varepsilon \cdot \mu}} \tag{2.2}$$

Das Verhältnis zwischen $c$ und $c_0$ wird als Brechungsindex $n$ bezeichnet. Da die Materialkonstanten $\varepsilon = \varepsilon_r \varepsilon_0$ und $\mu = \mu_r \mu_0$ für gewöhnlich durch Faktoren $\varepsilon_r$ und $\mu_r$ auf die Vakuumwerte $\varepsilon_0$ und $\mu_0$ bezogen sind, gilt

$$n = \frac{c_0}{c} = \sqrt{\varepsilon_r \cdot \mu_r}. \tag{2.3}$$

Im allgemeinen Fall sind $\varepsilon_r$ und $\mu_r$ (und damit auch $n$) von der Frequenz des übertragenen Signals abhängig. Für verlustbehaftete Übertragungsmedien sind sie sogar komplexwertig. Die Zeit $t$, die eine Welle benötigt, um die Strecke $\mathfrak{S}$ zurückzulegen, ist durch das Integral (2.4) gegeben.

$$t = \int_{\mathfrak{S}} c^{-1}(\vec{x}) \mathrm{d}\vec{x} \tag{2.4}$$

Die Wellenfront ist dabei immer senkrecht zu $\mathfrak{S}$ ausgerichtet. Allerdings muss $\mathfrak{S}$ nicht dem direkten Weg zwischen $S$ und $E$ entsprechen. Durch Veränderung von $n$ entlang des Ausbreitungsweges kann es zu Reflexion und Brechung der Welle beim Übergang zwischen verschiedenen Medien kommen. Für eine Welle, die aus einem Medium mit Brechzahl $n_1$ mit einem

[1] Die Größen $c$, $\varepsilon$ und $\mu$ für das Vakuum werden durch eine 0 im Index kenntlich gemacht. Es gilt: $c_0 = 2,99792458 \cdot 10^8 \, \frac{\mathrm{m}}{\mathrm{s}}$, $\varepsilon_0 = 8,854 \cdot 10^{-12} \, \frac{\mathrm{As}}{\mathrm{Vm}}$, $\mu_0 = 4\pi \cdot 10^{-7} \, \frac{\mathrm{Vs}}{\mathrm{Am}}$

Winkel $\alpha$ zum Lot einer ebenen Grenzfläche in ein Medium mit der Brechzahl $n_2$ übergeht, gilt das Snelliussche Brechungsgesetz.

$$\frac{\sin\alpha}{\sin\beta} = \frac{n_2}{n_1} = n \tag{2.5}$$

Wir definieren dazu $n$ als den Quotienten der beiden Brechzahlen. Der Winkel $\beta$ wird von der Ausbreitungsrichtung der gebrochenen Welle und dem Lot der Grenzfläche eingeschlossen. Mitunter wird jedoch ein Teil der eintreffenden Welle von der Grenzfläche reflektiert. Die Ausbreitungsrichtung der reflektierten Welle liegt mit dem Flächenlot und der Ausbreitungsrichtung der Hauptwelle in einer Ebene und schließt den Winkel $\alpha$ mit dem Lot auf der zur Eingangswelle abgewandten Seite ein. Der Anteil zwischen reflektierter Welle $r_\parallel, r_\perp$ und gebrochener Welle $t_\parallel, t_\perp$ ergibt sich dabei aus den Fresnelschen Formeln, die hier für den Spezialfall $\mu_1 = \mu_2$ und idealer dielektrischer Medien (keine Fortpflanzungsverluste) gegeben sind.

$$\begin{aligned} r_\parallel &= \frac{n^2\cos\alpha - \sqrt{n^2 - sin^2\alpha}}{n^2\cos\alpha + \sqrt{n^2 - sin^2\alpha}} \quad r_\perp = \frac{\cos\alpha - \sqrt{n^2 - sin^2\alpha}}{\cos\alpha + \sqrt{n^2 - sin^2\alpha}} \\ t_\parallel &= \frac{2n\cos\alpha}{n^2\cos\alpha + \sqrt{n^2 - sin^2\alpha}} \quad t_\perp = \frac{2\cos\alpha}{\cos\alpha + \sqrt{n^2 - sin^2\alpha}} \end{aligned} \tag{2.6}$$

Der Grad der Reflexion hängt dabei von der Polarisation der Hauptwelle ab. Die Polarisation bezieht sich auf die Einfallsebene, also die durch Grenzflächenlot und Einfallsrichtung aufgespannte Ebene. Es fällt auf, dass mit steigendem Verhältnis von $n_2/n_1$ ein immer größerer Teil der Welle reflektiert wird.

Für Indoor-Anwendungen stellen somit Materialien mit hohem Wassergehalt (Wasser besitzt eine sehr hohe Permittivitätszahl), wie z. B. nasse Gemäuer, ein Hindernis dar. Nicht zu unterschätzen ist ebenfalls der Einfluss des Menschen auf die Ausbreitung von Mikrowellen, dessen Wasseranteil über die Hälfte des Körpergewichts ausmacht. In [KK04] wird u. a. genau dieser Effekt untersucht. Zudem werden elektromagnetische Wellen durch leitfähige Materialien, wie sie z. B. in Fahrstuhl- und Lüftungsschächten eingesetzt werden, stark gedämpft.

Es gibt allerdings ein Kriterium, mit dem sich sicherstellen lässt, dass eine Welle den direkten Weg vom Sender zum Empfänger (line of sight) nimmt. Die Fresnelzone[2] ist der Rotationsellipsoid, in dessen Brennpunkten sich Sender und Empfänger befinden. Der Streckenzug Sender → Ellipsoidpunkt → Empfänger ist genau um den Betrag $\lambda/2$ länger als die Entfernung $r$ zwischen Sender und Empfänger (siehe Abbildung 2.1). Dabei bezeichnet $\lambda = c/f$ die Wellenlänge des Signals mit der Frequenz $f$.

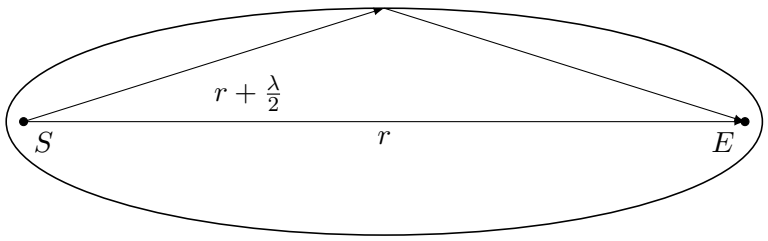



Abbildung 2.1: Fresnelzone zwischen $S$ und $E$

[2] genauer: die Fresnelzone erster Ordnung

Eine direkte Ausbreitung der Welle liegt vor, wenn die Fresnelzone weitgehend frei von Hindernissen ist. Der größte Durchmesser $b$ der Fresnelzone berechnet sich nach den Ellipsengleichungen durch

$$b = \sqrt{r\lambda - \frac{\lambda^2}{2}} \xrightarrow{r \gg \lambda} \sqrt{r\lambda}. \tag{2.7}$$

Erreicht das Signal nicht nur auf einem Weg den Empfänger, findet am Empfängerort eine Interferenz des Signals mit sich selbst statt, welches durch Streuung, Reflexion oder Brechung unterschiedlich verzögernde Wege genommen hat. In diesem Fall spricht man von sogenannten Multipath-Problemen. Multipath-Effekte variieren meist im Laufe der Zeit und rufen das als Schwund bzw. Fading bezeichneten Schwanken der Empfangsfeldstärke beim Empfänger hervor. Bereits eine Bewegung des Empfängers um den Abstand $\lambda/2$ bewirkt daher eine starke Veränderung der Feldstärke, da ein Umschlag von konstruktiver zu destruktiver Interferenz stattfindet. Bei der Feldstärkenlokalisierung muss dieser Effekt durch einen ausreichend langen Beobachtungszeitraum kompensiert werden, um eine vollständige Periode des zeitabhängigen Fadings aufzunehmen.

Eine vertiefte Betrachtung der Signalausbreitung wird u. a. in [Siz04] und [SA07] vorgenommen. Speziell für WLAN-Signale relevante Modelle werden in [PLM02, PKB02, SGAK05, Has93] diskutiert, wobei zumeist jedoch nur auf Feldstärkeaspekte eingegangen wird.

## 2.2 Lokalisierung

Unter Lokalisierung soll im Rahmen dieser Arbeit die Bestimmung der Position $\vec{r} = (x, y, z)^T$ eines Objekts in Bezug auf ein Bezugskoordinatensystem verstanden werden. Für den Abstand $d_i$ zwischen $\vec{r}$ und einer Menge von Referenzpunkten $\vec{r}_i$ gilt

$$d_i^2 = \|\vec{r}_i - \vec{r}\|^2 = (x_i - x)^2 + (y_i - y)^2 + (z_i + z)^2. \tag{2.8}$$

Die Größe $\vec{r}$ kann nun durch mehrere Messungen und anschließende Trilateration bestimmt werden. Im dreidimensionalen Fall sind zur eindeutigen Bestimmung der Unbekannten $x$, $y$ und $z$ im quadratischen Gleichungssystem (2.8) vier Referenzpunkte $\vec{r}_i$ samt der entsprechenden $d_i$ notwendig. Die $\vec{r}_i$ müssen sich zudem in allgmeiner Lage befinden. Auf Methoden zur Lösung eines solchen Gleichungssystems wird u. a. in [Mur07] eingegangen. Zwei dort vorgestellte grundlegende Verfahren sollen hier im Folgenden näher betrachtet werden.

Im nichtlinearen Gleichungssystem (2.8) können durch Aufstellen der Differenzen $d_i^2 - d_j^2$ die Terme $x^2$, $y^2$ und $z^2$ eliminiert werden.

$$\begin{aligned} d_i^2 - d_j^2 &= \sum_{\alpha \in \{x,y,z\}} \left((\alpha_i - \alpha)^2 - (\alpha_j - \alpha)^2\right) \\ &= \sum_{\alpha \in \{x,y,z\}} \left((\alpha_i^2 - \alpha_j^2) - 2(\alpha_i - \alpha_j)\alpha\right) \\ d_i^2 - d_j^2 + \underbrace{\sum_{\alpha \in \{x,y,z\}} \alpha_j^2 - \alpha_i^2}_{d_{ij}} &= \sum_{\alpha \in \{x,y,z\}} 2(\alpha_j - \alpha_i)\alpha \end{aligned}$$

In Matrixschreibweise erhält man

$$\underbrace{\begin{pmatrix} x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \\ x_4-x_1 & y_4-y_1 & z_4-z_1 \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\vec{x}} = \underbrace{\frac{1}{2}\begin{pmatrix} d_1^2-d_2^2+d_{12} \\ d_1^2-d_3^2+d_{13} \\ d_1^2-d_4^2+d_{14} \end{pmatrix}}_{\vec{b}} \tag{2.9}$$

wobei für $j$ o. B. d. A. der Index 1 gewählt wurde. Das durch Linearisierung erhaltene Gleichungssystem (2.9) lässt sich nun durch Methoden der linearen Algebra nach $(x, y, z)^T$ auflösen. Die Qualität der Lösung hängt dabei von der Konditionszahl ab,

$$\kappa(A) = \kappa(A^{-1}) = \|A\| \cdot \|A^{-1}\| \tag{2.10}$$

wobei für die Systemmatrix $A$ die folgende Matrixnorm verwendet werden kann

$$\|A\| = \max_{\|x\|=1} \|Ax\|. \tag{2.11}$$

Die Konditionszahl ist eine obere Schranke für die Verstärkung des in $\vec{b}' = \vec{b} + \vec{\varepsilon}$ enthaltenen Messfehlers $\vec{\varepsilon}$ der ungestörten Größe $\vec{b}$. Durch Linksmultiplikation von (2.9) mit $A^{-1}$ erhält man unter Einbeziehung der Fehlerterme die Lösung für $\vec{x}'$ mit

$$\vec{x}' = A^{-1}\vec{b} + A^{-1}\vec{\varepsilon} \qquad \text{und} \qquad \frac{\|A^{-1}\vec{\varepsilon}\|}{\|\vec{x}\|} \leq \kappa(A^{-1}) \cdot \frac{\|\vec{\varepsilon}\|}{\|\vec{b}\|}. \tag{2.12}$$

Für Matrizen mit großer Konditionszahl wird der Fehler somit stark verstärkt.

Eine andere Möglichkeit zur Lösung der Gleichungen (2.8) bietet die Ausgleichsrechnung in Kombination mit numerischen Berechnungsverfahren. Geht man wieder von fehlerbehafteten Messgrößen aus, müssen die Gleichungen (2.8) um Fehlerterme $\varepsilon_i$ erweitert werden, damit das Gleicheitszeichen gerechtfertigt ist.

$$(x_i - x)^2 + (y_i - y)^2 + (z_i + z)^2 = (d_i + \varepsilon_i)^2 \tag{2.13}$$

Das Ziel der Methode der kleinsten Quadrate besteht darin, eine Lösung $(x\ y\ z)^T$ zu finden, sodass die Summe der Fehlerquadrate minimal ist:

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \underbrace{\left(\sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} - d_i\right)^2}_{f_i(x,y,z)} = F(x, y, z) \to \min. \tag{2.14}$$

Die Funktion $F(x, y, z)$ ist minimal, wenn ihr Gradientenvektor an dieser Stelle verschwindet:

$$\nabla F(x, y, z) = \vec{0} \quad \Longleftrightarrow \quad \frac{\partial F}{\partial x} = \frac{\partial F}{\partial y} = \frac{\partial F}{\partial z} = 0. \tag{2.15}$$

Die Ableitung von $F$ nach $x$ lautet

$$\frac{\partial F}{\partial x} = \sum_{i=1}^{n} \frac{\partial}{\partial x}(f_i^2) = 2\sum_{i=1}^{n} f_i \cdot \frac{\partial}{\partial x} f_i = \sum_{i=1}^{n} \frac{f_i(x - x_i)}{f_i + d_i}. \tag{2.16}$$

Analog berechnen sich auch die Ableitungen nach $y$ und $z$. Kompakter lassen sich die Ableitungen in Matrixschreibweise

$$\nabla F = \begin{pmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \\ \frac{\partial F}{\partial z} \end{pmatrix} = J^T \cdot \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \qquad \text{mit der Jabcobi-Matrix} \qquad J = \begin{pmatrix} \frac{\partial f_1}{x} & \frac{\partial f_1}{y} & \frac{\partial f_1}{z} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{x} & \frac{\partial f_n}{y} & \frac{\partial f_n}{z} \end{pmatrix}$$

darstellen. Die Nullstelle von $\nabla F$ kann nun mit dem Newton-Verfahren bestimmt werden, welches nach der Iteratiosvorschrift

$$\vec{d}_{(k+1)} = \vec{d}_{(k)} - \left( J_{(k)}^T J_{(k)} \right)^{-1} J_{(k)}^T \vec{f}_{(k)} \tag{2.17}$$

einen Positionsvektor $\vec{d}_{(k+1)} = (x\ y\ z)^T$ bestimmt, der näher am Extrempunkt von $F$ liegt als $\vec{d}_{(k)}$. Ein geeigneter Startwert $\vec{d}_{(0)}$ kann mit der Methode der Linearisierung bestimmt werden. Die Methode der kleinsten Quadrate ist zudem auf überbestimmte Gleichungssysteme anwendbar, für die die Linearisierungsmethode durch den Einfluss der Messfehler scheitern kann.

## 2.3 IEEE 802.11

Maßgebend für die Wireless LAN Technologie ist der 11. Teil der Norm IEEE 802 [IEE07]. Sie definiert „eine Medienzugriffsschicht (MAC) und verschiedene Darstellungsschichten (PHY) für die drahtlose Kommunikation zwischen festen, mobilen und sich bewegenden Teilnehmern“.[3] Sie wurde seit ihrer ersten Verabschiedung 1997 mehrfach durch verschiedene Ergänzungen erweitert. Am bekanntesten sind die Ergänzungen IEEE 802.11a, 802.11b, 802.11g und 802.11n, die allesamt den Ursprungsstandard um weitere Darstellungsschichten ergänzen.

Bevor auf die konkreten Eigenschaften der Medienzugriffsschicht und der Darstellungsschichten eingegangen wird, sollen zunächst grundlegende Begriffe und Zusammenhänge erläutert werden, die bei IEEE 802.11 Netzwerken relevant sind. Sie sind in den Kapiteln 3 und 5 von IEEE 802.11 definiert. Neben dem bereits zitierten Standard sei noch auf Sekundärliteratur wie [Gas05] oder [Rec08] hingewiesen, die sich als allgemeinverständliche Erläuterungen der 802.11-Norm verstehen. Erstgenannte Quelle ist wesentlich präziser als letztere, geht jedoch nicht auf die Darstellungsschicht nach IEEE 802.11n ein.

**Station (STA)** Eine Station ist ein Gerät, welches eine Medienzugriffsschicht und mindestens eine Darstellungsschicht implementiert, die konform zu IEEE 802.11 ist. Jede Station (und ausschließlich Stationen) ist eine adressierbare Einheit bzgl. IEEE 802.11.

**Basic Service Set (BSS)** Ein BSS ist die grundlegende Einheit eines IEEE 802.11 Drahtlosnetzwerks und stellt eine Menge von Stationen dar. Das BSS kann sich bildlich als eine Funkzelle vorgestellt werden. Eine Drahtloskommunikation zwischen zwei Stationen ist nur möglich, wenn beide Mitglied des selben BSS sind. Es kann jedoch nicht davon ausgegangen werden, dass eine Station eines BSS in der Lage ist, mit jeder anderen Station desselben BSS zu kommunizieren.

[3] vgl. Abschnitt 1.1 in [IEE07]

**Distribution System (DS)** Ein DS ist ein zumeist drahtgebundenes Netzwerk, welches mehrere BSS und lokale Netzwerke (LANs) miteinander verbindet.

**Extended Service Set (ESS)** Ein ESS ist eine Menge von BSS, die durch ein Distribution System verbunden sind und sich für die Sicherungsschicht aller involvierten Stationen wie ein BSS darstellt.

**Independent BSS (IBSS)** Ein IBSS stellt die einfachste Form eines IEEE 802.11 Netzwerks dar. Innerhalb eines IBSS kommunizieren die Stationen direkt miteinander. Für IBSS-Zellen wird auch der Begriff „Ad-hoc-Netzwerk“ verwendet.

**Access Point (AP)** Ein Access Point ist eine Station, die einen Zugang zwischen dem drahtlosem Medium und dem DS herstellt. Jeder Access Point definiert ein Basic Service Set. Jede Kommunikation von Stationen innerhalb eines solchen BSS erfolgt ausschließlich über den Access Point. Die Kommunikation zweier Stationen erfordert somit einen Hop über den Access Point. Diesem Prinzip folgende Netzwerke werden als „managed“ bzw. als „Infrastrukturnetzwerke“ bezeichnet.

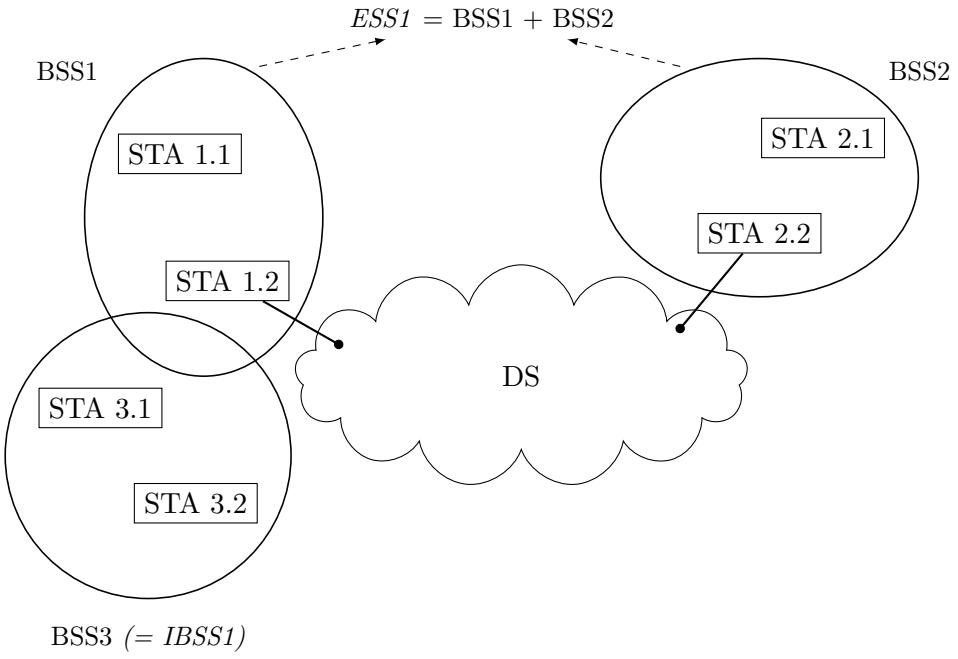



Abbildung 2.2: Darstellung der verschiedenen Begriffe der IEEE 802.11 Architektur

**Source/Destination Address (SA/DA)** Die SA und die DA stellen die Adressen der Netzwerkteilnehmer dar, die das zu übertragende Datenpaket erzeugt haben bzw. finales Ziel des Datenpakets darstellen. Es kommen 48 Bit-Adressen nach IEEE 802.2 (für gewöhnlich als „MAC-Adressen“ bezeichnet) zum Einsatz.

**Transmitting/Receiving Station Address (TA/RA)** Die TA und die RA stellen die Adressen der Stationen dar, die ein Datenpaket über das Drahtlosmedium aussenden bzw. für die eine solche Aussendung bestimmt ist. Es kommen ebenfalls 48 Bit-Adressen nach IEEE 802.2 zum Einsatz.

**PLME** Die „Physical Layer Management Entity“ ist die Menge von Soft- und Hardware, die für die Steuerung der Darstellungsschicht verantwortlich ist. Aufgrund strenger Timing-Anforderungen ist die PLME zumeist in Form von Firmware und speziellen Signalprozessoren implementiert.

**MLME** Die „MAC Layer Management Entity“ ist das Äquivalent zur PLME auf der Sicherungsschicht. Netzwerkgeräte, die die MLME in Form von Betriebssystemtreibern implementieren, nennt man „Soft-MAC Geräte“. Sie erlauben prinzipbedingt eine stärkere Kontrolle über den Empfang und den Versand von WLAN-Paketen als Hard-MAC Geräte, die auch die MAC-Abläufe über eine Firmware steuern und nur über Ethernet-Frames mit dem Betriebssystemkern kommunizieren. Zeitkritische Abläufe in der Sicherungsschicht werden jedoch auch zumeist bei Soft-MAC Geräten mittels einer Firmware behandelt.

### 2.3.1 Der IEEE 802.11 MAC-Layer

Die Sicherungs- oder MAC-Schicht von Funknetzen muss eine Reihe von Problemen bewältigen, die in drahtgebundenen Netzen nicht auftreten können. Übertragungsfehler sind in Funknetzen keine Ausnahme, sondern zu erwarten. Die MAC-Schicht sieht deswegen einen Mechanismus vor, der dem ursprünglichen Sender die fehlerfreie Übertragung von Paketen bestätigt. Die Kommunikation läuft vollständig über ein Broadcast-Medium ab, wodurch es ferner zu Kollisionen kommen kann. Bei klassischem Ethernet wird zunächst überprüft, ob das Medium belegt ist, bevor gesendet wird. Eine Station $A$, die zu $B$ senden will, kann jedoch nicht wissen, ob $C$ bereits das Medium bei $B$ belegt. Das ist genau dann der Fall, wenn $C$ in Reichweite von $B$, aber nicht von $A$ liegt (siehe Abbildung 2.3). Dieses Problem wird als „Hidden Station Problem“ bezeichnet.

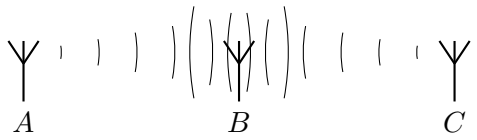



Abbildung 2.3: Das „Hidden Station“ Problem

Die Datenübertragung erfolgt nach einem festen Format, das in Kapitel 7 von IEEE 802.11 spezifiziert ist. Der IEEE 802.11 Frame ist in seiner allgemeinen Form in Abb. 2.4 dargestellt.

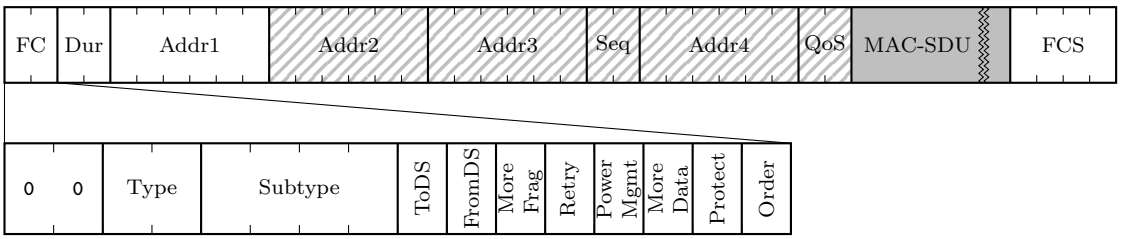



Abbildung 2.4: IEEE 802.11 Frame

Der Nutzlast MAC-SDU wird ein Header vorangestellt. Zusätzlich wird der Frame durch ein CRC-Prüfzeichen (FCS) abgeschlossen. Die grau schattierten Elemente des Frames sind

optional. Ob sie zum Frame gehören oder nicht, hängt vom Frame-Typ ab, der im Frame Control Feld (FC) kodiert wird. Einem optionalen Element des Frame-Kopfes kann nur die Nutzlast oder das rechte Nachbarelement des allgmeinen Frame-Formats folgen. Die Receiving Address ist in Addr1 und die Transmitting Address in Addr2 kodiert. Wie die Source Address und die Destination Address in den Adressfeldern kodiert sind, hängt von den Flags ToDS und FromDS ab und ist in Tabelle 2.1 aufgeschlüsselt. Die Flags sind gesetzt, wenn sich Ziel und Sender bzw. Quelle und Empfänger unterscheiden und somit das Distribution System in die Übertragung involviert ist.

| To | From | Addr1 | Addr2 | Addr3 | Addr4 | Bedeutung |
|---|---|---|---|---|---|---|
| 0 | 0 | RA = DA | TA = SA | BSSID | - | Ad-hoc-Transfer |
| 1 | 0 | RA = BSSID | TA = SA | DA | - | Transfer STA → AP |
| 0 | 1 | RA = DA | TA = BSSID | SA | - | Transfer AP → STA |
| 1 | 1 | RA | TA | DA | SA | WLAN-Bridge |

Tabelle 2.1: Bedeutung der Adressfelder im IEEE 802.11 Frame

IEEE 802.11 kennt drei verschiedene Frame-Kategorien, die im Type-Feld kodiert werden:

**Daten-Frames (Data)** Sie transportieren Nutzlast, die über das IEEE 802.2 Logical Link Control Protokoll in die MAC-SDU eingebettet ist.

**Steuer-Frames (Control)** Sie dienen der Steuerung des Medienzugriffs. Die MAC-SDU von Control-Frames ist leer.

**Verwaltungs-Frames (Management)** Sie dienen der Ausführung von Verwaltungsfunktionen wie der Bekanntgabe der ESSID eines Netzes, der Synchronisierung von Timern oder der Energieverwaltung von WLAN-Stationen. Management-Frames können eine Nutzlast transportieren.

Im Duration-Feld (Dur) eines Frames ist eine Zeitangabe in Mikrosekunden vermerkt.[4] Jede Station verwaltet einen sog. NAV-Wert.[5] Von jeder Station wird gefordert, den Duration-Wert eines Frames auszuwerten, auch wenn sie nicht Empfänger desselben ist. Ist ihr NAV-Wert kleiner als der Duration-Wert, wird der NAV-Wert auf den Duration-Wert erhöht. Positive NAV-Werte werden jede Mikrosekunde um den Betrag 1 dekrementiert. Eine Station kann erst dann einen Frame versenden, wenn ihr NAV-Wert 0 erreicht hat. Der NAV-Mechanismus realisiert eine sog. virtuelle Carrier Sense Funktion. Wie die Duration-Werte berechnet werden, regelt IEEE 802.11 in Kapitel 7.2 für jedes Frame-Format gesondert.

Für einen möglichst kollisionsfreien Zugriff auf das Drahtlosmedium müssen sich die Stationen gegenseitig koordinieren. IEEE 802.11 legt dazu drei verschiedene Zugriffsverfahren fest: die verteilte Koordinierung (distributed coordination function, DCF), die zentrale Koordinierung (point coordination function) und eine Mischung aus beiden (hybrid coordination function, HCF). Die DCF ist das einfachste und meistverwendete Zugriffsverfahren. Die PCF benötigt eine zentrale Instanz – den Access Point – und ist damit nicht in Ad-hoc-Netzwerken anwendbar. Ihre Implementierung ist optional und kommt nur recht selten zur Anwendung.[6]

[4] Bei bestimmten Frames haben die Duration-Werte eine andere Bedeutung. Diese Fälle sind im Rahmen der Arbeit nicht relevant. Ist das MSB des Duration-Werts 0, handelt es sich immer um eine gültige Zeitangabe.

[5] Abk. für Network Allocation Vector

[6] vgl. Kapitel 9 in [Gas05]

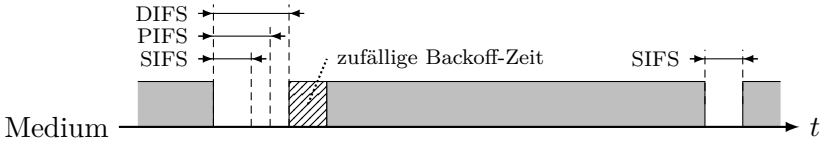



Abbildung 2.5: Frame-Abstände

Zwischen der Übertragung zweier Frames muss für eine gewisse Zeit Ruhe auf dem Übertragungsmedium herrschen. Diese Frame-Abstände sind in Abbildung 2.5 dargestellt. Nach Ende einer Übertragung ist ein Zugriff auf das Medium noch für den Zeitraum SIFS (short interframe space) verboten. Ein Medienzugriff über DCF kann erst erfolgen, wenn seit dem Ende der letzten Übertragung mindestens der Zeitraum DIFS (distributed interframe space) vergangen ist. Analoges gilt für die PCF und den Zeitraum PIFS. Nach Ablauf des DIFS können sich prinzipiell mehrere Stationen um das Medium bewerben. Um die Kollisionswahrscheinlichkeit zu verringern, wird zur DIFS-Zeit zusätzlich eine zufällige Wartezeit nach dem Backoff-Verfahren addiert. Eine Frame-Sequenz, bei der die Abstände zwischen allen konsekutiven Frames höchstens SIFS ist, nennt man unteilbar bzw. atomar. Sie kann nicht unterbrochen werden, da zu jedem Zeitpunkt geregelt ist, wer mit dem folgenden Frame das Medium belegen darf. Frames, die nur einen SIFS-Abstand erfordern, haben somit die höchste Priorität und können ohne besondere Sicherungsverfahren ausgesandt werden. Auf die konkreten Werte der symbolischen Platzhalter SIFS, PIFS und DIFS wird im folgenden Abschnitt eingegangen. Es gilt jedoch stets

$$PIFS = SIFS + s \qquad \text{und} \qquad DIFS = SIFS + 2s,$$

wobei $s$ die sog. Slot-Zeit darstellt und ebenfalls von der PHY-Implementierung abhängt.

Bevor im Infrastrukturmodus eine Datenübertragung überhaupt möglich wird, muss sich eine Station gegenüber dem Access Point durch entsprechende Verwaltungsframes authentifizieren und anschließend mit ihm assoziieren. Die Authentifikation soll dabei die Identität der Station gegenüber dem Access Point beweisen.[7] Die Assoziation wird notwendig, weil innerhalb des Verteilungssystems zwischen den Access Points eine Vermittlung der Datenframes erfolgen muss. Erst durch die Assoziation wird klar, zu welchem Access Point die Datenframes geleitet werden müssen, und ob evt. an anderen Access Points gepufferte Frames umzuleiten sind. Je nachdem, ob die Authentifikation und die Assoziation etabliert sind oder nicht, können gewisse Frames verworfen werden. Welche Frames in allen drei Zuständen „nicht authentifiziert, nicht assoziiert“, „authentifiziert, aber nicht assoziiert“ und „authentifiziert und assoziiert“ nicht verworfen werden, regelt IEEE 802.11 in Abschnitt 11.3. Die folgenden drei Fälle sind für diese Arbeit von Bedeutung.

- Die Steuerframes „RTS“, „CTS“ und „ACK“
- Die Verwaltungsframes „Beacon“, „Probe-Request“ und „Probe-Response“
- Datenframes, sofern die Flags „To DS“ und „From DS“ nicht gesetzt sind.

[7] Der Begriff „Authentifikation“ leitet aber fehl, was dessen Stärke angeht. Das einzige obligatorische Authentifikationsverfahren nach IEEE 802.11 nimmt die von der Station übermittelte Identität ohne weitere Prüfung als echt an.

In Ad-hoc-Netzwerken sind „To DS“ und „From DS“ immer gelöscht. Datenframes werden somit nie gefiltert. Die Assoziation und Authentifikation entfällt.

Laut Abschnitt 11.1 in IEEE 802.11 muss jede MLME eine sog. „timing synchronising function“ (TSF) implementieren. Basis für die TSF ist ein 64 Bit-Zähler, der mit einer Rate von $1\,\text{MHz} \pm 0.01\%$ inkrementiert wird. Der Standard legt zudem Verfahren fest, wie diese Zähler innerhalb eines BSS synchronisiert werden. Dazu senden in Infrastrukturnetzen die Access Points in regelmäßigen Abständen sog. Beacon-Frames aus, die einen TSF-Zeitstempel enthalten. Der Wert dieses Zeitstempels soll dem entsprechen, den der TSF-Zähler zu dem Zeitpunkt annimmt, zu dem das Symbol, welches das erste Bit dieses Zeitstempels enthält, an das Übertragungsmedium ausgestrahlt wird.[8] In Ad-hoc-Netzwerken werden die Beacon-Frames von allen teilnehmenden Stationen erzeugt. Eine Station, die für eine Zeit größer als die festgelegte Beacon-Periode keinen Beacon-Frame empfangen hat, initiiert dessen Aussendung. Immer wenn eine Station einen Beacon-Frame für ihr BSS erfolgreich empfängt, kann eine Modifikation ihres TSF-Zähler notwendig werden. In Ad-hoc-Netzen setzt sich der Zeitstempel mit dem größtem Wert durch. Es wird also der lokale TSF-Stand mit dem Zeitstempel im Beacon-Frame verglichen und ggf. nach vorn korrigiert. Über kurz oder lang wird sich also die schnellste Uhr aller Stationen durchsetzen. In Infrastrukturnetzen wird immer der Zeitstempel des Access Points in den lokalen Zähler übernommen.

### 2.3.2 Die IEEE 802.11 PHY-Layer

Die Darstellungsschichten bzw. „Physical Layer“ – kurz PHY – übernehmen die Aufgabe, die Frames der MAC-Schicht in einer für die Drahtloskommunikation geeigneten Form über einen Sender abzustrahlen und die an einem Empfänger abgegriffenen Signale wieder in 802.11-Frames umzuwandeln. Eine weitere Aufgabe des PHY ist das „clear channel assessment“, die Meldung an den MAC-Layer, ob das Medium derzeit durch eine Übertragung belegt wird oder nicht. Das Medium wird von der MAC-Schicht nur als frei angesehen, wenn das CCA keine laufende Übertragung meldet und der NAV-Wert 0 ist.

In seiner ursprünglichen Form definierte IEEE 802.11 drei verschiedene PHY-Layer:

- den „Frequency hopping spread spectrum PHY“ (FHSS),
- den „Direct sequence spread spectrum PHY“ (DSSS) und
- den „Infrared PHY“ (IR).

Wegen der lichtartigen Ausbreitung von Infrarotstrahlung wäre der letztgenannte PHY-Layer am besten für die Lokalisierung geeignet. Jedoch gab es laut [Gas05] nie ernstzunehmende WLAN-Geräte, die diese Bitübertragungsschicht realisierten.[9] Dem Autor ist nicht bekannt, ob FHSS PHYs heutzutage überhaupt noch betrieben werden, da sie ein historisches Relikt bei der Entwicklung von IEEE 802.11 darstellen. Aus diesem Grund soll hier nur auf die Funktionsweise des DSSS PHY eingegangen werden. Die konkreten Modulationsverfahren werden nicht weiter beschrieben, da sie im Rahmen dieser Arbeit von nachrangigem Interesse sind. Das DSSS PHY hat durch die Ergänzungen von IEEE 802.11 im Laufe der Zeit eine Reihe von Modifikationen erfahren:

[8] vgl. Abschnitt 11.1.2 in [IEE07]
[9] vgl. Kapitel 10 in [Gas05]

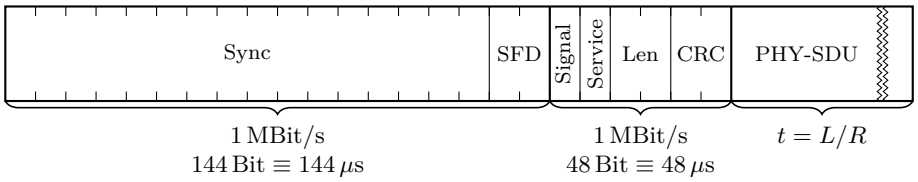
- IEEE 802.11 spezifiert Datenraten von 1 und 2 MBit/s im 2.4 GHz ISM-Band.
- IEEE 802.11a spezifiziert den Betrieb von Wireless LAN im 5 GHz ISM-Band. Es kommt orthogonales Frequenzmultiplexing (Orthogonal Frequency-Division Multiplexing, OFDM) als Modulationsverfahren zum Einsatz, was Übertragungsraten von 6 bis 54 MBit/s ermöglicht.
- IEEE 802.11b erweitert IEEE 802.11 durch Verwendung neuer Modulationsverfahren um die Datenraten 5.5 und 11 MBit/s.
- IEEE 802.11g setzt die Modulationsverfahren aus IEEE 802.11a im 2.4 GHz-Band um. Es werden entsprechende Vorkehrungen getroffen, damit nicht-OFDM-fähige Stationen keine Kollisionen auf dem Medium verursachen.
- IEEE 802.11n beschreibt die aktuellste Ausbaustufe der möglichen Übertragungsraten. Die Werte von bis zu 600 MBit/s werden allerdings nicht durch neue Modulationsverfahren erreicht, sondern durch die MIMO-Technik, bei der gleichzeitig mehrere Antennen am Sender und am Empfänger zum Einsatz kommen. Einserseits kann durch das Einschleifen von Verzögerungsgliedern in jedem Antennenpfad und anschließende Kombination der einzelnen Antennensignale ein besseres Signal-Rausch-Verhältnis erzielt werden. Andererseits ist es auch möglich, mehrere unterschiedliche Datenströme gleichzeitig zu übertragen. Eine knappe Einführung in die MIMO-Technik im Hinblick auf IEEE 802.11n kann in [HHSW09] nachgeschlagen werden.


Abbildung 2.6: Aufbau einer WLAN-Aussendung

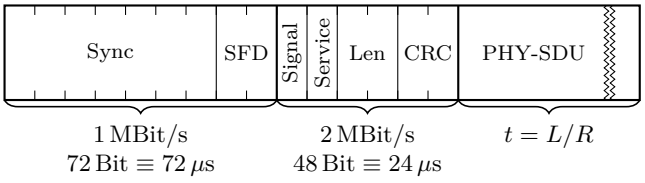
Der grundsätzliche Aufbau einer WLAN-Aussendung ist unabhängig vom verwendeten Modulationsverfahren in Abbildung 2.6 dargestellt. Zunächst wird eine Präambel ausgesandt, die dazu dient, den Empfänger auf den folgenden Datenstrom zu synchronisieren. Zudem kann die Präambel genutzt werden, um bei Empfängern mit mehreren Antennen diejenige zu bestimmen, die die besten Empfangsbedingungen verspricht. Der PLCP-Header kodiert die verwendete Datenrate und die Länge der folgenden PHY-SDU. Wesentliche Unterschiede der Frame-Formate bestehen nur zwischen den DSSS-PHYs (802.11 und 802.11b) und den OFDM-PHYs (802.11a und 802.11g).

**DSSS-PHYs** modulieren die zu übertragenden Daten auf die Phasenlage der Trägerfrequenz. Abbildung 2.7a zeigt das ursprüngliche Frame-Format der Darstellungsschicht. Die Symbole werden mit einer Rate von 1 MHz übertragen. Zunächst wird mit jedem Symbol genau ein Bit kodiert, was einer Datenrate von 1 MBit/s entspricht. Nach einer 144 Bit langen Präambel, die zur Synchronisation der Empfangshardware dient, folgt der PLCP-Header. Im Signal-Feld des Headers wird die Datenrate, in der die PHY-SDU übertragen wird, kodiert. Das Len-Feld speichert die Dauer der Übertragung in $\mu$s. Mit Beginn der PHY-SDU wird abhängig vom Inhalt des Signal-Felds auf ein Modulationsverfahren umgeschaltet, welches 2 Bits in einem Symbol kodiert und damit eine Datenrate von 2 MBit/s erlaubt.

Mit der Verabschiedung von IEEE 802.11b wurden weitere Übertragungsraten definiert. Dabei wird nach dem PLCP-Header auf eine Symbolrate von 1.375 MHz umgeschaltet und je 4 bzw.




(a) 802.11 PLCP-Frame bzw. 802.11b „Long Frame“




(b) 802.11b „Short Frame“

Abbildung 2.7: Frame-Aufbau der DSSS-PHYs

8 Bits pro Symbol übertragen, was zu einer Bruttodatenrate von 5.5 bzw. 11 MBit/s führt. Da so allerdings die Nettodatenrate durch die lange Präambel stark eingeschränkt wird, wurde durch 802.11b die sogenannte „short preamble“ eingeführt (siehe Abbildung 2.7b). Nach dieser verkürzten Präambel wird sofort auf 2 MBit/s umgeschaltet und es folgt beim Übergang zur PHY-SDU ggf. eine weitere Bitratenumschaltung. In Gleichungen ausgedrückt ergibt sich die Übertragungszeit $t_{TX}$ einer MAC-PDU mit der Länge von $L$ Bits durch

$$t_{P\&H} = \begin{cases} 192\,\mu\text{s} & \text{für 802.11 oder 802.11b „Long Frame“} \\ 96\,\mu\text{s} & \text{für 802.11b „Short Frame“} \end{cases} \tag{2.18}$$
$$t_{TX} = t_{P\&H} + L/R.$$

Es gilt weiterhin

$$SIFS = 10\,\mu\text{s} \qquad \text{und} \qquad s = 20\,\mu\text{s}. \tag{2.19}$$

**OFDM-PHYs** verwenden eine vergleichsweise lange Symboldauer von 4 µs. Um die hohen Datenraten zu erzielen, werden mit jedem OFDM-Symbol allerdings bis zu 216 Datenbits kodiert. Die ersten 24 Bit des PLCP-Headers werden mit der kleinsten Datenrate von 6 MBit/s, also genau einem OFDM-Symbol, übertragen. Im Rate-Feld wird kodiert, mit welcher Datenrate ab dem Service-Feld übertragen wird. Das Verfahren, mit dem die Datenbits kodiert werden, erfordert am Ende jedes Datenstroms sechs Tail-Bits, die allesamt auf 0 gesetzt sein müssen. Ist die Länge der PHY-SDU inklusive der Tail-Bits noch kein Vielfaches der Anzahl an Datenbits pro Symbol ($N_{DBPS}$), wird mit Pad-Bits bis zur nächsten Symbolgrenze aufgefüllt.

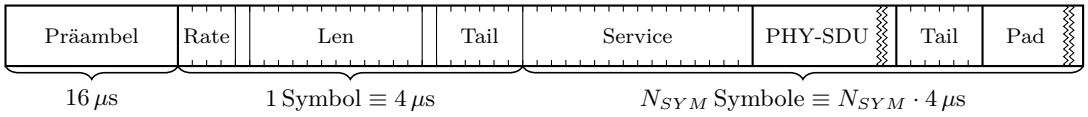



Abbildung 2.8: Frame-Aufbau der OFDM-PHYs

Das in Abbildung 2.8 dargestellte Format ist aber in der beschriebenen Form nur für 802.11a gültig. Für das 2.4 GHz-Band fordert 802.11g kleine Modifikationen.

- Einem OFDM-Frame folgt eine Ruhezeit von 6 $\mu$s. Diese Ausweitung wird aber durch eine um den gleichen Betrag verkürzte SIFS-Zeit kompensiert.
- IEEE 802.11g definiert ein weiteres Frame-Format, was mit einem DSSS-Header beginnt, aber anschließend auf OFDM umschaltet. Diese Variante ist als optional eingestuft und nur sehr selten realisiert.[10]
- Werden Stationen entdeckt, die nur ein 802.11b-PHY enthalten, wird die SLOT-Zeit $s$ vergrößert.

Die Übertragungszeit $t_{TX}$ berechnet sich folglich durch:

$$\begin{aligned} N_{DBPS} &= R \cdot 4\,\mu\text{s} \\ N_{SYM} &= \lceil (22 + 8 \cdot L)/N_{DBPS} \rceil \\ t_{ext} &= \begin{cases} 0\,\mu\text{s} & \text{für 802.11a} \\ 6\,\mu\text{s} & \text{für 802.11g} \end{cases} \\ t_{TX} &= 16\,\mu\text{s} + (1 + N_{SYM}) \cdot 4\,\mu\text{s} + t_{ext} \end{aligned} \tag{2.20}$$

Für die Frame-Zwischenräume gilt:

$$\begin{aligned} SIFS &= \begin{cases} 16\,\mu\text{s} & \text{für 802.11a} \\ 10\,\mu\text{s} & \text{für 802.11g} \end{cases} \\ s &= \begin{cases} 20\,\mu\text{s} & \text{für 802.11g, wenn 802.11b-Stationen anwesend sind} \\ 9\,\mu\text{s} & \text{sonst} \end{cases} \end{aligned} \tag{2.21}$$

## 2.4 Wireless LAN unter Linux

Der Zugriff auf WLAN-Hardware erfolgte unter Linux lange Zeit über die sog. „Wireless Extension“ [Tou]. Sie nutzte eine Reihe von `ioctl()`-Aufrufen als Schnittstelle und ermöglichte das Setzen und Auslesen von Parametern, die nur bei Funknetzen auftreten (z. B. Sendeleistung, Kanal, Signalstärke, ...). Der Nutzer konnte über die Kommandozeilenprogramme `iwconfig`, `iwlist` und `iwspy` auf die Wireless Extension zugreifen. Die eigentliche Schnittstelle zu den WLAN-Geräten war als gewöhnliche Ethernet-Schnittstelle implementiert. Sie konnte allerdings erst genutzt werden, wenn über die Wireless Extension alle benötigten Parameter wie die ESSID etc. gesetzt waren. Die Treiber enthielten meist ihre eigene Implementierung der MLME, sofern es sich um Soft-MAC-Hardware handelte.

Die Wireless Extension wird derzeit nicht mehr weiterentwickelt. Mit Kernelversion 2.6.22[11] ging das MAC80211-Subsystem in den Linux-Kernel ein. MAC80211 ist die Implementierung einer MLME, unabhängig von der verwendeten Hardware. Aktueller Maintainer ist Johannes

[10] vgl. Kapitel 14 in [Gas05]
[11] vgl. `http://www.kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.22`

Berg,[12] der in [Berb] einen, nicht mehr in allen Teilen aktuellen, Überblick über das Subsystem gibt. Alle Informationen zu MAC80211 werden von den Kernel-Entwicklern in [MAC] zusammengetragen, woraus die API-Dokumentation [Bera] entstand. Im Gegensatz zur Wireless Extension werden die Netzwerkschnittstellen nicht durch den WLAN-Treiber, sondern durch MAC80211 bereitgestellt. Es ist sogar möglich, jedem WLAN-Gerät – im Sprachgebrauch von MAC80211 als „PHY“ bezeichnet – mehrere Netzwerk-Schnittstellen zuzuweisen. Diese Schnittstellen werden auch „Virtual Interface (VIF)“ genannt. Das Erzeugen und Löschen von VIFs ist mit dem Kommandozeilenprogramm `iw` möglich.

```
$ iw phy ⟨PHY⟩ interface add ⟨Interface⟩ type ⟨Typ⟩
$ iw dev ⟨Interface⟩ del
```

Der Typ bezeichnet dabei den Betriebsmodus der Schnittstelle und kann auch direkt mittels `iw` für bestehende VIFs gesetzt werden. Vorher muss das Interface ggf. deaktiviert werden.

```
$ ip link set ⟨Interface⟩ down
$ iw dev ⟨Interface⟩ set type ⟨Typ⟩
$ ip link set ⟨Interface⟩ up
```

Mögliche Betriebsmodi sind:

**managed** Das Interface repräsentiert eine Station in einem Infrastrukturnetzwerk. Eine Verbindung mit einer ESS kann durch

```
$ iw dev ⟨Interface⟩ connect ⟨SSID⟩ [⟨Frequenz⟩] [⟨BSSID⟩]
$ iw dev ⟨Interface⟩ disconnect
```

hergestellt bzw. getrennt werden. Über das Netzwerkinterface werden Ethernet-Frames ausgetauscht.

**ibss** Über IBSS-Schnittstellen wird der Zugang zu Ad-hoc-Netzwerken ermöglicht. Das Beitreten und Verlassen einer Ad-hoc-Zelle erfolgt über:

```
$ iw dev ⟨Interface⟩ ibss join ⟨SSID⟩ ⟨Frequenz⟩
$ iw dev ⟨Interface⟩ ibss leave
```

Es werden ebenfalls Ethernet-Frames ausgetauscht.

**monitor** Im Monitor-Modus werden alle empfangenen 802.11-Frames, also insbesondere auch Steuer- und Management-Frames, an den Prozessbereich weitergeleitet. Dem eigentlichen Frame wird dabei ein sogenannter Radiotap-Header vorangestellt, auf dessen Aufbau im folgenden Unterabschnitt näher eingegangen wird. Neben der Möglichkeit, den Datenverkehr über das Funkmedium zu überwachen, können auch mit einem Radiotap-Header versehene 802.11-Frames an das MAC80211-Subsystem übergeben werden. Die Frames werden anschließend ohne weitere Bearbeitung ausgesandt. Diese Technik wird als „Frame Injection“ bezeichnet.

**master** Sofern die Hardware dazu fähig ist, kann sie mittels Master-Modus als Access Point betrieben werden. Es wird jedoch nur ein kleiner Teil der Master-MLME im Linux-Kern

[12] vgl. `http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob_plain;f=MAINTAINERS;hb=HEAD`

implementiert. Für die Anbindung des WLAN-Geräts an das Distribution System und die Realisierung einer Verschlüsselung ist der Dienst hostapd[13] zuständig. Zudem kann der Master-Modus nicht durch das Programm `iw` aktiviert werden.

Welche Modi ein PHY unterstützt, kann ebenfalls durch `iw` ermittelt werden.

```
$ iw phy ⟨PHY⟩ info
Wiphy PHY
...
        Supported interface modes:
                 * IBSS
                 * managed
                 * monitor
        Supported commands:
                 * new_interface
                 * set_interface
                 * set_bss
                 * join_ibss
                 * set_tx_bitrate_mask
                 * set_channel
                 * connect
                 * disconnect
...
```

Durch `iw` können noch eine Reihe weiterer Einstellungen vorgenommen werden. Im Rahmen dieser Arbeit sind vor allem der RTS-Threshold, der Fragmentation-Threshold und die Bitraten-Maske von Relevanz.

```
$ iw phy ⟨PHY⟩ set rts ⟨Schwellwert⟩
$ iw phy ⟨PHY⟩ set rts off
$ iw phy ⟨PHY⟩ set frag ⟨Schwellwert⟩
$ iw phy ⟨PHY⟩ set frag off
$ iw dev ⟨Interface⟩ set bitrates legacy-2.4 ⟨Bitraten⟩
$ iw dev ⟨Interface⟩ set bitrates legacy-5   ⟨Bitraten⟩
```

Der RTS-Schwellwert gibt an, ab welcher Framegröße ein sog. RTS-CTS-Handshake vor der eigentlichen Datenübertragung ausgeführt werden soll. Dieser Mechanismus soll sicherstellen, dass die Übertragung des Datenframes keine Kollision beim Empfänger hervorruft, was langfristig zu einer höheren Nettodatenrate führt. Der Fragmentierungs-Schwellwert legt die Maximalgröße eines Datenframes fest. Bei Überschreitung der Grenze wird das Datenpaket innerhalb der MAC-Schicht in Fragmente zerlegt, die höchstens die Größe des Schwellwerts haben. In Abschnitt 3.3 wird näher auf die genauere Funktionsweise dieser Verfahren eingegangen.

Zur weiteren Steigerung der Nettodatenrate wählt ein Rate-Control-Algorithmus eine Datenrate für die Übertragung von Frames aus. Er versucht, das Optimum zwischen Bruttodatenrate und der Wahrscheinlichkeit eines Übertragungsfehlers, die mit der Bruttodatenrate steigt,

[13] `http://hostap.epitest.fi/hostapd/`

zu finden. Aktuell kommt hierzu der Minstrel-Algorithmus[14] zum Einsatz. Mittels Bitraten-Masken wird der Linux-Kern angewiesen, nur die angegebenen Bitraten als Ergebnis für die Bitraten-Auswahl zuzulassen. Diese Einstellung wirkt sich aber stets auf alle VIFs eines WLAN-Geräts aus. Weiterhin kann ein WLAN-Treiber das Ergebnis des Rate-Controllers auch ignorieren, indem er ein eigenes Verfahren implementiert oder die Entscheidung der Firmware des WLAN-Geräts überlässt.

### 2.4.1 Der Radiotap-Header

Der Radiotap-Header dient dazu, Informationen zwischen WLAN-Treiber und Anwendungsprogrammen zu transportieren, die über die im IEEE 802.11 MAC-Header kodierten Informationen hinausgehen. Im Wesentlichen handelt es sich dabei um Daten, die für die PHY-Schicht relevant sind, oder Statusinformationen, die vom WLAN-Treiber gesammelt werden. Das Radiotap-Protokoll erlaubt es, nur die benötigten Datenfelder im Protokollkopf zu kodieren. Alle definierten Datenfelder, Kandidaten für spätere Erweiterungen und ihre Bedeutung werden von der Entwicklergemeinde unter [Rad] zusammengetragen.

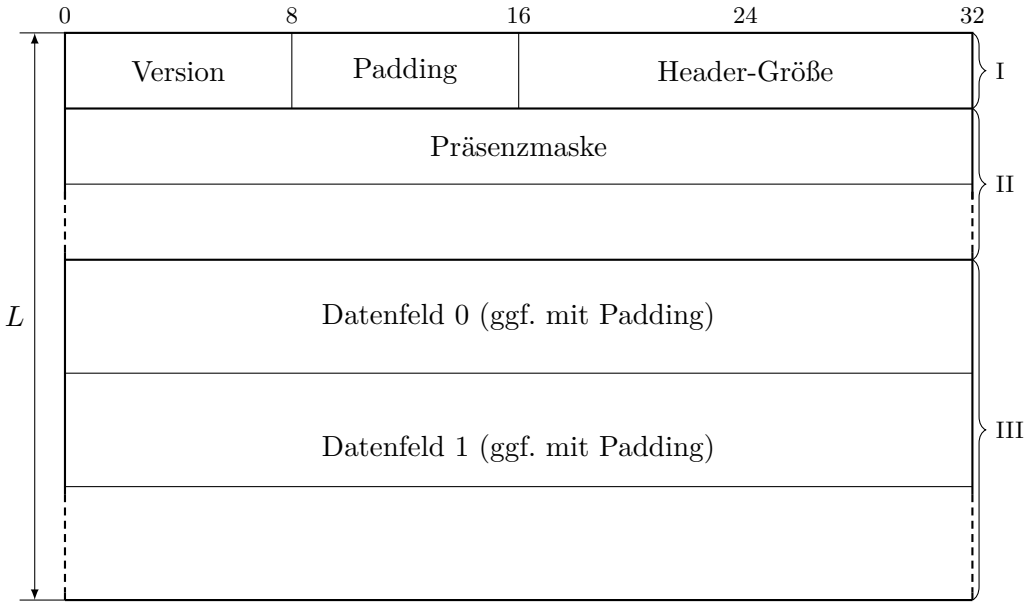



Abbildung 2.9: Der Radiotap-Header

Der grundsätzliche Aufbau eines Radiotap-Header ist in Abbildung 2.9 dargestellt. Er besteht aus drei Teilen. Den ersten Teil bildet ein Vorspann, der die Version des enthaltenen Radiotap-Protokolls kodiert (derzeit 0) und die Gesamtgröße $L$ des Radiotap-Headers speichert. Auf den folgenden MAC-Frame kann somit nur durch Auswertung von Teil I zugegriffen werden.

Der Präsenzteil (II) beschreibt den Aufbau des Datenteils (III) und besteht aus einer Folge von 32 Bit-Worten. Jedem definierten Datenfeld ist eine Bitnummer, beginnend bei 0, zugeordnet. Die Nummern $31, 63, \ldots$ werden dabei ausgelassen. Das höchstwertige Bit (Erweiterungsbit)

[14] vgl. `http://wireless.kernel.org/en/developers/Documentation/mac80211/RateControl/minstrel`

einer Präsenzbitmaske ist gesetzt, wenn ihr eine weitere Bitmaske folgt. Teil III beginnt unmittelbar nach dem ersten Präsenz-Wort mit nicht gesetztem Erweiterungsbit. Der Datenteil enthält nur die Datenfelder, deren korrespondierendes Bit in Präsenzteil gesetzt ist. Die Datenfelder werden in aufsteigender Reihenfolge ihrer Bitnummer im Datenteil abgelegt. Jedem Datenfeld ist zudem eine Basisgröße zugeordnet. Vor Beginn eines Datums werden ggf. Füllbytes eingefügt, sodass das Datum an einem Vielfachen seiner Basisgröße in Bezug auf den Beginn von Teil III ausgerichtet ist.

Alle Angaben im Radiotap-Header werden in Little-Endian-Reihenfolge kodiert. Tabelle 2.2 gibt einen Überblick über die wichtigsten Radiotap-Datenfelder. Die mit ($\star$) markierten Felder sind derzeit nur Vorschläge für eine spätere Erweiterung des Radiotap-Protokolls.

| Bit | Größe / Alignm. | Name | Bedeutung |
|---|---|---|---|
| 0 | 8/8 Byte | TSFT | Wert des TSF-Timers zu dem Zeitpunkt, zu dem das erste Bit der MAC-PDU an die MAC-Schicht übergeben wurde. Dieses Datenfeld ist nur bei empfangenen Frames vorhanden. |
| 1 | 1/1 Byte | Flags | Verschiedene Flags, wie z. B.: „Frame ist ein 802.11b Short-Frame“ (`0x02`), „WEP-Verschlüsselung erfolgt“ (`0x04`), „Frame unterliegt Fragmentierung“ (`0x08`), „der MAC-Frame enthält die FCS“ (`0x10`). Das Feld ist für empfangene und zu sendende Frames gültig. |
| 2 | 2/2 Byte | Rate | Datenrate, die für die Übertragung genutzt wurde bzw. werden soll. Die Rate wird in Vielfachen von 500 kBit/s angegeben. |
| 3 | 4/2 Byte | Kanal | Mittenfrequenz des Übertragungskanals in MHz (unteres Halbwort) und diverse Flags, die das verwendete Modulationsverfahren anzeigen (oberes Halbwort). Dieses Datenfeld ist nur für empfangende Frames relevant. |
| 10 | 1/2 Byte | dBm TX power | Antennenseitige Sendeleistung für den auszusendenden Frame in Dezibel bezogen auf 1 mW. |
| 11 | 1/1 Byte | Antenna | Index der Antenne, die für den Empfang genutzt wurde bzw. für das Versenden genutzt werden soll. Diverse Treiber implementieren statt eines Index auch eine Bitmaske. |
| 12 | 1/1 Byte | dB Antenna | Signalstärke an der Antenne in Dezibel bezogen auf eine vom Hersteller frei wählbare aber feste Referenz. |
| 14 | 2/2 Byte | RX Flags | Informationen über den empfangenen Frame. Bisher ist nur ein Flag definiert: „CRC-Prüfung im PLCP-Header schlug fehlt“ (`0x0002`). |
| 15 | 2/2 Byte | TX Flags ($\star$) | Informationen über den zu sendenden Frame. Relevante Flags für diese Arbeit sind: „nutze RTS-CTS-Handshake“ (`0x0004`), „erwarte ACK-Frame und versende Frame erneut, wenn ACK ausbleibt“ (`0x0008`), „nehme keine treiberseitige Modifikation am SEQ-Feld des MAC-Headers vor“ (`0x0010`). |
| 16 | 2/1 Byte | RSSI ($\star$) | Signalqualität beim Empfang des Frames (unteres Byte), wobei der RSSI maximal den Wert des oberen Bytes annehmen kann. |
| 19 | 3/1 Byte | MCS ($\star$) | Der „Modulation and Coding Scheme“-Index für Frames nach 802.11n. Er ist vergleichbar mit dem Rate-Feld für 802.11abg-Frames. Die Zuordnung von MCS-Werten zu Übertragungsraten und Modulationsverfahren wird in Abschnitt 20.3.5 von [IEE09] vorgenommen. |

Tabelle 2.2: Relevante Radiotap-Felder

# Kapitel 3

# Ansätze zur Laufzeitmessung

## 3.1 Grundprinzipien

Es existieren drei verschiedene Grundprinzipien zur Bestimmung der Signallaufzeit: Time of Arrival (TOA), Time Difference of Arrival (TDOA) und Time of Flight (TOF). Im weiteren Verlauf dieses Abschnitts finden folgende Konventionen Verwendung:

- Sei $X$ ein Ereignis. Der Ausdruck $\mathcal{C}^U(X)$ bezeichne den Zeitstempel, der an der Uhr $U$ beim Eintreten von $X$ abgelesen wird.
- Es existiert eine globale Zeit $\mathcal{C}$.
- Es gelte $\forall U, X : \mathcal{C}^U(X) = r_U \cdot \mathcal{C}(X) + \delta_U$ bzw. in Kurzschreibweise $\mathcal{C}^U = r_U \cdot \mathcal{C} + \delta_U$. In anderen Worten ausgedrückt, besitzt jede Uhr einen konstanten Drift und einen konstanten Offset zur Globalzeit.

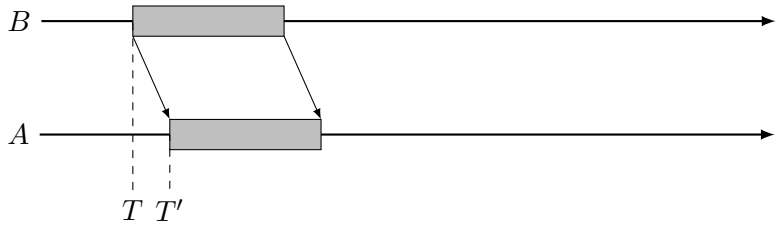



Abbildung 3.1: Time of Arrival

Beim TOA-Verfahren wird die Laufzeit direkt, wie im Raum-Zeit-Diagramm 3.1 gezeigt, durch den Vergleich zweier Uhren festgestellt. Der Sender $B$ überträgt eine Nachricht zum Empfänger $A$. Der Zeitpunkt, zu dem die Aussendung beginnt, stelle das Ereignis $T$ dar. Nach der Zeit $t$, die das Signal benötigt, um sich über die Entfernung $d = c \cdot t$ auszubreiten, stellt der Empfänger den Beginn der Aussendung fest (Ereignis $T'$). Der Zeitstempel $\mathcal{C}^B(T)$ werde von $B$ in der Nachricht kodiert oder in einer zweiten Nachricht an $A$ gesendet. In Globalzeit dargestellt, gilt für $t$ nun

$$t = \mathcal{C}(T') - \mathcal{C}(T) = \frac{\mathcal{C}^B(T') - \delta_B}{r_B} - \frac{\mathcal{C}^A(T) - \delta_A}{r_A}. \tag{3.1}$$

Es fällt auf, dass zur Bestimmung der Laufzeitdifferenz Wissen über $r_A$, $r_B$, $\delta_A$ und $\delta_B$ notwendig ist oder alternativ sichergestellt werden kann, dass sich die Uhren von $A$ und $B$ sehr gut synchronisieren lassen. Eine Abweichung von 3 ns kann dabei schon einen Fehler von ungefähr einem Meter hervorrufen. Die Frequenzstabilität eines gewöhnlichen Quarzosziallators wird solchen Anforderungen nicht genügen, womit dieser Ansatz für die praktische Anwendung eher untauglich ist.

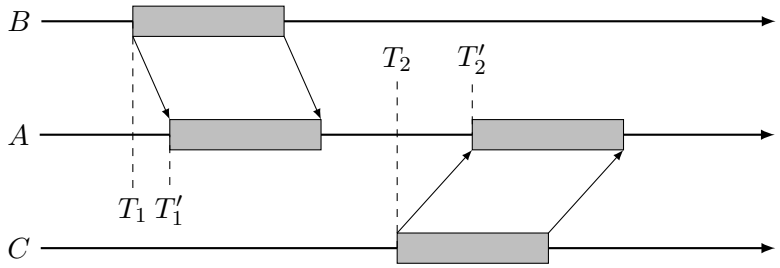



Abbildung 3.2: Time Difference of Arrival

Diesem Problem kann dadurch begegnet werden, dass man ein Messverfahren anwendet, welches Zeitdifferenzen nur durch Ablesen der selben Uhr ermittelt. Das TDOA-Verfahren misst die Zeit, die zwischen dem Empfang zweier Aussendungen verstreicht (siehe Abbildung 3.2). Die Signale werden indes von verschiedenen Sendern $B$ und $C$ abgegeben, die im Allgemeinen einen unterschiedlichen Abstand von $A$ – und somit auch eine unterschiedliche Laufzeit zu $A$ – haben. Ist die Zeitdifferenz zwischen dem Beginn der Aussendungen bekannt, kann durch Vergleich mit den Empfangszeitpunkten bei $A$ die Laufzeitdifferenz beider Signale wie folgt bestimmt werden.

$$\underbrace{(\mathcal{C}(T_1') - \mathcal{C}(T_1))}_{t_{A,B}} - \underbrace{(\mathcal{C}(T_2') - \mathcal{C}(T_2))}_{t_{A,C}} = \frac{\mathcal{C}^A(T_1') - \mathcal{C}^A(T_2')}{r_A} - \left(\frac{\mathcal{C}^B(T_1) - \delta_B}{r_B} - \frac{\mathcal{C}^C(T_2) - \delta_C}{r_C}\right) \tag{3.2}$$

Allein mit diesen Informationen kann allerdings noch nicht auf die Entfernung $|\overline{AB}|$ bzw. $|\overline{AC}|$ geschlossen werden. Für $A$ kommen alle Punkte in Frage, für die die Differenz $|\overline{AB}| - |\overline{AC}|$ konstant ist. $A$ liegt im ebenen Fall also auf einer Hyperbel mit den Brennpunkten $B$ und $C$. Eine eindeutige Bestimmung der Entfernungen ist für den ebenen Fall dennoch möglich, wenn ein zusätzlicher Teilnehmer $D$ eingeführt wird. Befinden sich $B$, $C$ und $D$ in allgemeiner Lage, sind also nicht kollinear, schneiden sich alle drei Hyperbeln $(B, C)$, $(B, D)$ und $(C, D)$ im Punkt $A$. Praktisch werden solche Verfahren beim GPS verwendet, da man nicht davon ausgehen kann, dass ein GPS-Empfänger Zugriff auf eine von einer Atomuhr abgeleiteten hochgenauen Zeitbasis hat. Das Vorkommen von $\delta_B$ und $\delta_C$ in obiger Gleichung erfordert dennoch, dass sich die Uhren $B$ und $C$ sehr genau synchronisieren lassen bzw. der zeitliche Abstand zwischen den Ereignissen $T_1$ und $T_2$ im Vorfeld bekannt ist. In Infrastrukturnetzwerken könnten die Access Points diese spezielle Rolle wahrnehmen. Dem Autor ist jedoch nicht bekannt, ob Access Points mit solch genauer Zeitbasis kommerziell verfügbar sind. Zudem würde es der geforderten Allgemeinheit der zu verwendenden Hardware widersprechen. Die Ganggenauigkeit der Uhr von $A$ stellt ebenfalls einen genauigkeitsbegrenzenden Faktor

dar. Je größer die Zeitdifferenz zwischen $T_1'$ und $T_2'$ ist, um so stärker wirkt sich eine Abweichung von $r_A$ vom Nominalwert auf die gemessene Zeitdifferenz aus. Nimmt man für $r_A$ eine Genauigkeit von 100 ppm an, liegt die Unsicherheit einer Messung von 1000 $\mu$s bereits bei 100 ns bzw. ca. 33 m. Aufgrund dieser Probleme scheint das TDOA-Verfahren für die WLAN-Entfernungsmessung nicht anwendbar zu sein.

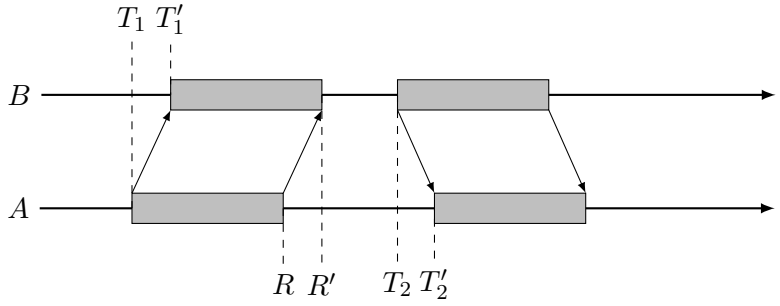



Abbildung 3.3: Time of Flight

Um den Synchronisierungsproblemen komplett aus dem Weg zu gehen, werden beim TOF-Verfahren Nachrichten in beide Richtungen ausgetauscht (siehe Abbildung 3.3). Die Zeit, die zwischen den Ereignissen $T_1$ und $T_2'$ vergeht, kann entsprechend folgender Gleichung in vier Teilabschnitte zerlegt werden.

$$\begin{aligned} \mathcal{C}(T_2') - \mathcal{C}(T_1) = &\underbrace{(\mathcal{C}(T_1') - \mathcal{C}(T_1))}_{t} + \underbrace{(\mathcal{C}(R') - \mathcal{C}(T_1'))}_{d} + \\ &\underbrace{(\mathcal{C}(T_2) - \mathcal{C}(R'))}_{g} + \underbrace{(\mathcal{C}(T_2') - \mathcal{C}(T_2))}_{t} \\ = &\ 2t + d + g \end{aligned} \tag{3.3}$$

Es folgt nun für $t$

$$t = \frac{\mathcal{C}^A(T_2') - \mathcal{C}^A(T_1)}{2r_A} - \frac{d+g}{2}. \tag{3.4}$$

Aus Symmetriegründen können die Signallaufzeit von $A$ nach $B$ und von $B$ nach $A$ als gleich angenommen werden. Die Zeit, die zwischen $R$ und $T_1$ bzw. $R'$ und $T_1'$ vergeht, entspricht genau der Länge der Aussendung, die im folgenden mit $d$ bezeichnet wird. Diese Länge kann sowohl von $A$ als auch von $B$ unter alleiniger Verwendung der lokalen Uhren bestimmt werden. Eine Uhrensynchronisation ist nicht notwendig. Die Lücke $g$ zwischen Empfang des ersten Signals und der Aussendung der Antwortnachricht kann ebenfalls allein durch die Uhr von $B$ bestimmt werden. Der Wert für $g$ kann im Bedarfsfall in einer dritten Nachricht kodiert an $A$ übermittelt werden. Unter Umständen ist es sogar möglich, die Werte von $d$ und $g$ von $A$ aus zu beeinflussen bzw. als konstant anzusetzen. Richtet man die Messungen jeweils am Ende der Aussendungen und nicht am Anfang aus, gilt obige Gleichung weiterhin. Für $d$ ist lediglich die Dauer der Antwortnachricht einzusetzen.

Durch die Elimination aller $\delta$ in Gleichung (3.4) ist das TOF-Verfahren die vielversprechendste Variante für eine praktische Umsetzung, da es keine hohen Anforderung an die Uhrensynchronisation stellt. Nach einigen allgemeinen Betrachtungen zum TOF-Verfahren wird in Abschnitt 3.3 auf Protokollabläufe eingegangen, die zur Realisierung einer TOF-Messung genutzt werden können.

## 3.2 Ein Verhaltensmodell für die gemessenen Signallaufzeiten

Aus den Erläuterungen im letzten Abschnitt wird klar, dass nur Verfahren für eine nähere Betrachtung in Frage kommen, bei denen die Differenz von Zeitstempeln gebildet wird, die an der selben Uhr abgelesen werden. Die Schreibweise $\mathcal{C}^X(T_2) - \mathcal{C}^X(T_1)$ wird demnach im Folgenden zu $T_2 - T_1$ vereinfacht.

Der Wert $\Delta = T_2 - T_1$, wie er sich z. B. aus Gleichung (3.4) ergibt, wird durch Messung ermittelt. In diesem Abschnitt soll diskutiert werden, wie sich der tatsächliche Wert $\Delta$ auf eine Reihe von Messwerten $\widetilde{\Delta}_i = \widetilde{T}_2^{(i)} - \widetilde{T}_1^{(i)}$ auswirkt. Jede Uhr besitzt eine Mindestauflösung $P$. Nach Ablauf dieser Periodenzeit vergrößert sich der von der Uhr angezeigte Zeitstempel um 1. Wechselte der Zeitstempel $\widetilde{T}$ dieser Uhr genau zum Zeitpunkt $T = 0$ der kontinuierlichen Globalzeit von Null auf Eins, lesen wir zu jedem Zeitpunkt $T$ der Globalzeit den Wert $\widetilde{T} = \lceil \frac{T}{P} \rceil$ ab. In den folgenden Rechnungen stellt $\langle X \rangle$ den Nachkommananteil der Größe $X$ dar.

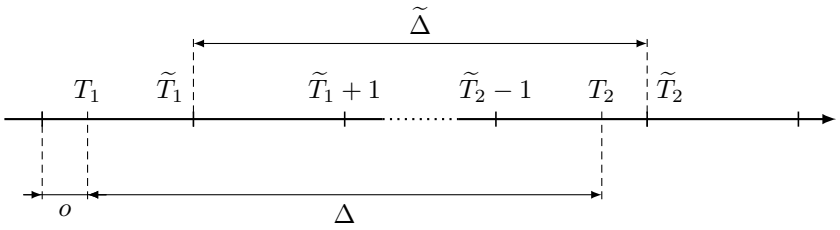



Abbildung 3.4: Bestimmung von $\Delta$ im Fall einer unabhängigen Uhr

Zunächst betrachten wir den Fall, dass die Ereignisse, zwischen denen die zu messende Zeitspanne vergeht, unabhängig vom Lauf der Uhr sind. Die Ereignisse sind auf dem Zeitstrahl in Abbildung 3.4 samt der erfassten Zeitstempel $\widetilde{T}_1$ und $\widetilde{T}_2$ dargestellt. Da wegen der Unabhängigkeit von Ereignis und Uhr der Phasenversatz zwischen $T_1$ und den Umschaltpunkten der Uhr unbekannt ist, wird er hier mit der Variable $o$ bezeichnet. Diese Größe hat Einfluss auf den Wert von $\widetilde{\Delta}$, denn es gilt

$$\begin{aligned}
\widetilde{\Delta} &= \left\lceil \frac{T_2}{P} \right\rceil && - \left\lceil \frac{T_1}{P} \right\rceil \\
&= \left\lceil \frac{T_1 + \Delta}{P} \right\rceil && - \left\lceil \frac{T_1}{P} \right\rceil \\
&= \left\lceil \left\lfloor \frac{T_1}{P} \right\rfloor + \left\langle \frac{T_1}{P} \right\rangle + \frac{\Delta}{P} \right\rceil && - \left\lceil \left\lfloor \frac{T_1}{P} \right\rfloor + \left\langle \frac{T_1}{P} \right\rangle \right\rceil \\
&= \left\lceil \left\langle \frac{T_1}{P} \right\rangle + \frac{\Delta}{P} \right\rceil && - \left\lceil \left\langle \frac{T_1}{P} \right\rangle \right\rceil
\end{aligned} \tag{3.5}$$

wobei der Term $\langle \frac{T_1}{P} \rangle$ durch $\frac{o}{P}$ ersetzt werden kann:

$$\widetilde{\Delta} = \left\lceil \frac{o}{P} + \frac{\Delta}{P} \right\rceil - \left\lceil \frac{o}{P} \right\rceil \tag{3.6}$$

Durch Fallunterscheidung erhält man schließlich:

$$\widetilde{\Delta} = \begin{cases} \left\lceil \frac{\Delta}{P} \right\rceil & 0 = o \\ \left\lfloor \frac{\Delta}{P} \right\rfloor & 0 < \frac{o}{P} < 1 - \left\langle \frac{\Delta}{P} \right\rangle \\ \left\lfloor \frac{\Delta}{P} \right\rfloor + 1 & 1 > \frac{o}{P} \geq 1 - \left\langle \frac{\Delta}{P} \right\rangle \end{cases} \tag{3.7}$$

Wegen der Unabhängigkeit kann man $o$ gleichverteilt im Intervall $[\,0, P\,[$ annehmen. Selbst wenn die Periode der Ereignisse $T_1^{(i)}$ bei wiederholter Messung in die Nähe eines ganzzahligen Vielfachen von $P$ kommt, wird durch unvermeidliche Bauteiltoleranzen der Zeitbasis ein Drift der Uhr eintreten. Innerhalb einer Periode der resultierenden Schwebung wird dann $o$ das gesamte Intervall überstrichen haben. So ist für hinreichend große Abstände der Messungen die Gleichverteilung gegeben. Bei zufällig verteiltem $o$ ist $\widetilde{\Delta}$ selbst eine Zufallsgröße, die einer Zweipunktverteilung folgt. Ihr Erwartungswert berechnet sich zu:

$$\begin{aligned} E\left[\widetilde{\Delta}\right] &= \left(1 - \left\langle \frac{\Delta}{P} \right\rangle\right) \cdot \left\lfloor \frac{\Delta}{P} \right\rfloor + \left\langle \frac{\Delta}{P} \right\rangle \cdot \left( \left\lfloor \frac{\Delta}{P} \right\rfloor + 1 \right) \\ &= \left\lfloor \frac{\Delta}{P} \right\rfloor - \left\langle \frac{\Delta}{P} \right\rangle \cdot \left\lfloor \frac{\Delta}{P} \right\rfloor + \left\langle \frac{\Delta}{P} \right\rangle \cdot \left\lfloor \frac{\Delta}{P} \right\rfloor + \left\langle \frac{\Delta}{P} \right\rangle \\ &= \left\lfloor \frac{\Delta}{P} \right\rfloor + \left\langle \frac{\Delta}{P} \right\rangle \\ &= \frac{\Delta}{P} \end{aligned} \tag{3.8}$$

Für $n$ unabhängige, aber identisch verteilte Messwerte $\widetilde{\Delta}_i$ ist die Funktion

$$\widehat{\Delta} = \frac{P}{n} \sum_{i=1}^{n} \widetilde{\Delta}_i \tag{3.9}$$

ein erwartungstreuer Schätzer für $\Delta$. Die Zweipunktverteilung von $\widetilde{\Delta}$ lässt sich auch als Summe der Konstanten $\left\lfloor \frac{\Delta}{P} \right\rfloor$ und einer Bernoulli-Verteilung mit dem Parameter $\left\langle \frac{\Delta}{P} \right\rangle$ darstellen. Die Konstante kann als Term $n \cdot \left\lfloor \frac{\Delta}{P} \right\rfloor$ aus der Summe herausgezogen werden, sodass $n$ unabhängige Bernoulli-verteilte Größen addiert werden. Die $n$-fache Faltung dieser Bernoulli-Verteilung ergibt eine $B\left(n, \left\langle \frac{\Delta}{P} \right\rangle\right)$-Binomialverteilung. Die Verteilung von $\widehat{\Delta}$ lässt sich formal als

$$\widehat{\Delta} \sim P \cdot \left\lfloor \frac{\Delta}{P} \right\rfloor + \frac{P}{n} \cdot \mathrm{B}\left(n, \left\langle \frac{\Delta}{P} \right\rangle\right) \tag{3.10}$$

darstellen. Die Varianz von $\widehat{\Delta}$ ist ein Maß für die Genauigkeit des Schätzers und bestimmt sich zu

$$D^2\left[\widehat{\Delta}\right] = \frac{P^2}{n} \cdot \left\langle \frac{\Delta}{P} \right\rangle \cdot \left(1 - \left\langle \frac{\Delta}{P} \right\rangle\right) \leq \frac{P^2}{4n}. \tag{3.11}$$

Wie bereits intuitiv klar ist, wird der Schätzer um so genauer, je kleiner die Auflösung $P$ ist oder je mehr Messwerte aufgenommen werden. Es bleibt hervorzuheben, dass keine Annahmen über das Verhältnis von $P$ zu $\Delta$ gemacht wurden. Obige Betrachtungen behalten

ihre Gültigkeit, auch wenn $P$ um mehrere Zehnerpotenzen größer ist als $\Delta$. Lediglich die Genauigkeit der Schätzung leidet bei konstantem $n$. Anders ausgedrückt ist es mit großem $n$ prinzipiell möglich, Zeiten im ns-Bereich mit einer Uhr in $\mu$s-Auflösung zu bestimmen.

Kommt als Zeitquelle eine Uhr zum Einsatz, die durch die WLAN-Hardware bereitgestellt wird, kann jedoch nicht mehr von einer Unabhängigkeit zwischen Uhr und den Ereignissen ausgegangen werden. Vielmehr wird ein synchroner Automat bewirken, dass die Aussendung von Frames nur bei Beginn des nächsten Takts startet bzw. der Empfänger nur zu Beginn eines Takts eine Aussendung registriert. Diesen Ansatz beschreiben auch die Autoren von [GH05]. Die folgende Berechnung von $\widehat{\Delta}$ lehnt sich an den Ausführungen der genannten Quelle an und wird um die mittlere Antwortzeit des Reflektors im TOF-Ablauf ergänzt.

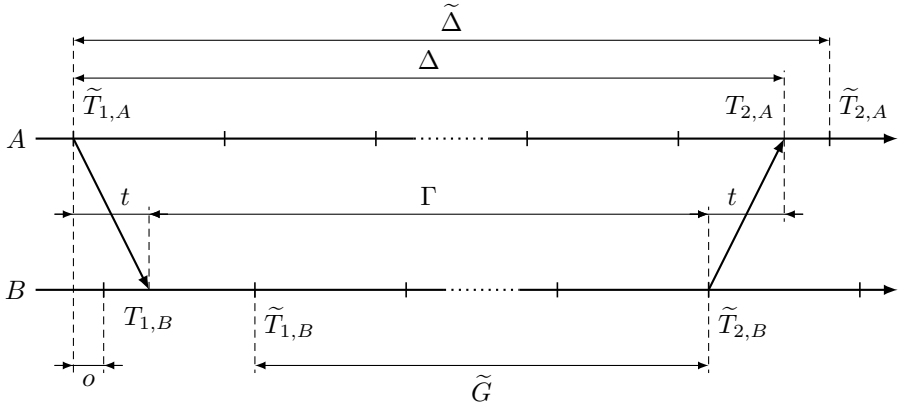



Abbildung 3.5: Bestimmung von $\Delta$ im Fall einer abhängigen Uhr

Abbildung 3.5 zeigt den TOF-Ablauf, wenn das Senden und Empfangen nur zu Timerticks geschieht. Zwischen der Uhr von $A$ und der Uhr von $B$ bestehe ein Phasenversatz von $o$. Aus den Elementargleichungen

$$\begin{aligned} T_{1,B} &= \widetilde{T}_{1,A} \cdot P + t - o \quad \widetilde{T}_{1,B} = \left\lceil \frac{T_{1,B}}{P} \right\rceil \\ T_{2,A} &= \widetilde{T}_{2,B} \cdot P + t + o \quad \widetilde{T}_{2,A} = \left\lceil \frac{T_{2,A}}{P} \right\rceil \\ \widetilde{T}_{2,B} &= \widetilde{T}_{1,B} + \widetilde{G} \end{aligned}$$

folgt für die Antwortzeit $\Gamma$ bei $B$

$$\begin{aligned} \Gamma &= \widetilde{T}_{2,B} \cdot P - T_{1,B} \\ &= \left\lceil \frac{T_{1,B}}{P} \right\rceil \cdot P + \widetilde{G} \cdot P - T_{1,B} \\ &= \left(1 - \left\langle \frac{T_{1,B}}{P} \right\rangle\right) \cdot P + \widetilde{G} \cdot P. \end{aligned} \tag{3.12}$$

Nimmt man wieder an, dass die Uhren einen langsamen Drift besitzen und $o$ sich somit für viele Messungen gleichmäßig über eine Periode verteilt, dann überstreicht der Nachkommaanteil

des Bruchs das Intervall $[\,0,1\,[$. Die Erwartungswerte von $\Gamma$ und $\Delta$ betragen folglich

$$\begin{aligned} E\left[\Gamma\right] &= (0.5 + \widetilde{G}) \cdot P \\ E\left[\Delta\right] &= (0.5 + \widetilde{G}) \cdot P + 2t. \end{aligned} \tag{3.13}$$

Der Messwert ist also im Mittel um eine halbe Periodendauer der Uhr von $B$ größer als die tatsächliche Antwortzeit. In die bei $A$ gemessene Größe $\widetilde{\Delta}$ geht noch eine weitere phasenabhängige Verzögerung bis zum Zeitpunkt $\widetilde{T}_{2,A}$ ein. Letztendlich ergibt sich

$$\begin{aligned} \widetilde{\Delta} &= \widetilde{T}_{2,A} - \widetilde{T}_{1,A} \\ &= \left\lceil \widetilde{T}_{1,B} + \widetilde{G} + \frac{t+o}{P} \right\rceil - \widetilde{T}_{1,A} \\ &= \left\lceil \frac{t-o}{P} \right\rceil + \left\lceil \frac{t+o}{P} \right\rceil + \widetilde{G} \\ &= \left\lceil \left\langle \frac{t}{P} \right\rangle - \frac{o}{P} \right\rceil + \left\lceil \left\langle \frac{t}{P} \right\rangle + \frac{o}{P} \right\rceil + \widetilde{G} + 2 \left\lfloor \frac{t}{P} \right\rfloor. \end{aligned} \tag{3.14}$$

Mit einer analogen Fallunterscheidung wie in Gleichung (3.7) lässt sich der Erwartungswert von $\widetilde{\Delta}$ bestimmen:

$$E\left[\widetilde{\Delta}\right] = \widetilde{G} + 1 + \frac{2t}{P} \tag{3.15}$$

Zur Bestimmung von $\Delta$ aus den Werten einer Messreihe kann wieder der Schätzer (3.8) zur Anwendung kommen. Um eine Aussage über dessen Genauigkeit treffen zu können, muss zunächst die Verteilung von $\widetilde{\Delta}$ bekannt sein. Es reicht aus, nur die von $o$ abhängigen Terme in (3.14) gesondert zu betrachten, die sich durch Fallunterscheidung nach der Substitution

$$\tau := \left\langle \frac{t}{P} \right\rangle \qquad \omega := \frac{o}{P}$$

wie folgt darstellen lassen.

$$X = \lceil \tau - \omega \rceil + \lceil \tau + \omega \rceil = \begin{cases} 0 & \tau \le \omega < 1 - \tau \text{ und } \tau < 1 - \tau \\ 1 & 0 \le \omega < \min(\tau, 1-\tau) \\ 1 & 1 > \omega \ge \max(\tau, 1-\tau) \\ 2 & \tau > \omega \ge 1 - \tau \text{ und } \tau > 1 - \tau \end{cases} \tag{3.16}$$

Bei genauerer Betrachtung wird deutlich, dass dies einer Zweipunktverteilung entspricht, die sich als Summe aus der Konstanten $\lfloor 2\tau \rfloor + 1$ und einer von $\tau$ abhängigen Bernoulli-Verteilung zusammensetzt.

$$X \sim \lfloor 2\tau \rfloor + 1 + \begin{cases} -\mathrm{Bn}(1-2\tau) & \tau < 1 - \tau \\ \mathrm{Bn}(2\tau - 1) & \tau \ge 1 - \tau \end{cases} \tag{3.17}$$

Analog zur Argumentation oben folgt $\widehat{\Delta}$ damit wieder einer Binomialverteilung. Die Varianz berechnet sich, unter Einbeziehung der Rücksubstituion, entsprechend der Gesetze der Binomialverteilung zu

$$D^2\left[\widehat{\Delta}\right] = \frac{P^2}{n^2} \cdot \begin{cases} n \cdot \left(2\left\langle \frac{t}{P} \right\rangle - 1\right) \cdot \left(2 - 2\left\langle \frac{t}{P} \right\rangle\right) & \left\langle \frac{t}{P} \right\rangle \ge 1 - \left\langle \frac{t}{P} \right\rangle \\ n \cdot \left(1 - 2\left\langle \frac{t}{P} \right\rangle\right) \cdot \left(2\left\langle \frac{t}{P} \right\rangle\right) & \left\langle \frac{t}{P} \right\rangle < 1 - \left\langle \frac{t}{P} \right\rangle \end{cases} \tag{3.18}$$

Für beide Fälle ist die Abschätzung

$$D^2\left[\widehat{\Delta}\right] \leq \frac{P^2}{4n} \tag{3.19}$$

gültig und scharf. Unter den gemachten Annahmen ist folglich keine Beeinträchtigung der Messgenauigkeit durch die Abhängigkeit der Sende- und Empfangsereignisse mit den Uhren zu erwarten.

## 3.3 Methoden zur Messwerterzeugung

Bisher wurde nur allgemein auf die Prinzipien der Laufzeitbestimmung mittels TOF-Verfahren eingegangen und deren praktische Realisierung vorübergehend ausgeblendet. Die Größen $d$ und $g$ waren ausschließlich abstrakte Variablen. Im Verlauf dieses Abschnitts sollen nun Methoden identifiziert werden, die einen TOF-Ablauf darstellen und die Bestimmung von $d$ und $g$ erlauben. Die Berechnung der Übertragungsdauer $d$ eines WLAN-Frames ist bereits mittels der in Abschnitt 2.3.2 gelegten Grundlagen möglich. Der genaue Wert von $g$ muss je nach betrachtetem Verfahren entweder durch Messung festgestellt werden, oder er wird vom IEEE 802.11-Standard festgelegt. Sofern Frame-Sequenzen des 802.11-Standards herangezogen werden sollen, legt Kapitel 9.12 des Standards alle erlaubten atomaren Sequenzen in Form einer kontextfreien Grammatik fest. Zu beachten ist nur, dass nicht alle Sequenzen durch eine MLME implementiert werden müssen. Aus dem Grund ist es bereits an dieser Stelle sinnvoll, sich nur auf DCF-Sequenzen zu beschränken (vgl. Abschnitt 2.3.1).

### 3.3.1 Das Ping-Verfahren

Ein einfaches TOF-Verfahren wird durch das simple Programm `ping` initiiert. Es greift auf das Internet Control Message Protocol (ICMP) zurück und sendet ein ICMP Echo-Request an einen entfernten Host. Dort wird es, sofern Firewall-Einstellungen dem nicht entgegenstehen, durch ein ICMP Echo-Reply quittiert. Der Protokollablauf ist im Raum-Zeit-Diagramm 3.6 dargestellt.

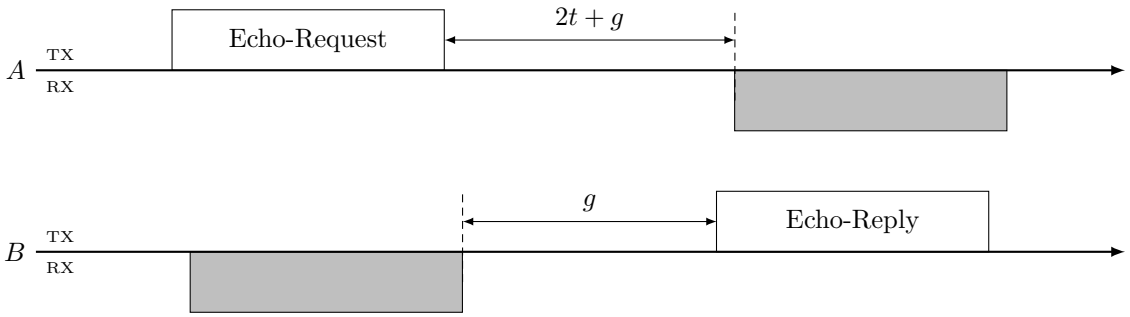



Abbildung 3.6: ICMP Echo-Request und Reply

Über den Zeitraum $g$ kann prinzipbedingt keine Aussage getroffen werden. Der Wert von $g$ ist aufgrund variabler Durchlaufzeiten in der Implementierung des IP-Stacks gestört. Selbst wenn

es gelänge, die Antwortzeit des IP-Stack sehr genau vorherzusagen, stellt die Request-Reply-Sequenz keine atomare Sequenz im Sinne von IEEE 802.11 dar. Eine unbeteiligte Station könnte im entscheidenden Moment das Funkmedium belegen und so eine Verzögerung des Reply-Pakets hervorrufen. Der konkrete Wert von $g$ muss folglich für jede Sequenz durch Messung erfasst werden. Dem Autor ist nicht bekannt, ob diese Methode bereits Gegenstand der Lokalisierung war. Allerdings gelangen Versuche, die Lichtgeschwindigkeit mittels ICMP Echo-Paketen in drahtgebundenen Netzen zu messen [LC02], was gewissermaßen dem inversen Problem zur Laufzeitlokalisierung entspricht.

### 3.3.2 Das DATA-ACK-Verfahren

Die IEEE 802.11 MAC-Schicht definiert ein sogenanntes Immediate-Acknowledge mit dem Ziel, den nicht selten auftretenden Fall verstümmelter Übertragungen abzufangen. Auf Frames, die eine solche Bestätigung erfordern, folgt nach einer SIFS-Zeit der entsprechende Acknowledge-Frame (ACK) wie in Abbildung 3.7 dargestellt.

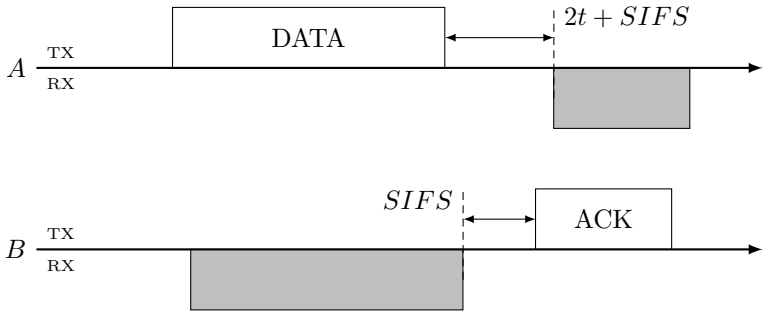



Abbildung 3.7: DATA-ACK-Sequenz

Eine Übersicht, welche Frames konkret bestätigt werden, gibt 802.11 durch die Definition aller atomaren Frame-Sequenzen in Kapitel 9.12. Unter Vernachlässigung einiger Fälle, die sich bei Benutzung des PCF-Zugriffsverfahrens ergeben, werden ACK-Frames für folgende Frames erwartet.

- Management- oder Datenframes, in deren Empfängeradresse (RA) das niederwertigste Bit (Individual/Group-Bit) nicht gesetzt ist. Diese Frames sind folglich nicht für Broad- oder Multicast-Adressen bestimmt.
- Fragmente der oben genannten Frames, sofern eine Fragmentierung erfolgt.
- Frames, die zur Realisierung von Stromsparfunktionen dienen (Power-Save Poll).

Für die SIFS-Zeit ist eine Toleranz von ±10% der Slot-Zeit zulässig. Bei Ausbleiben eines ACK-Frames an der Quelle des vermutlich verstümmelten Frames[1] wird die dortige MLME einen erneuten Übertragungsversuch starten.

Gegenüber dem Ping-Verfahren besitzt das DATA-ACK-Verfahren den Vorteil, dass der Wert von $g$ vorhersagbar ist, da er genau einem SIFS entspricht. Die Übertragungszeiten $d_{\text{DATA}}$

[1] Es kann auch vorkommen, dass der ACK-Frame verstümmelt wurde. Für diesen Fall nimmt die MLME beim Ziel eine Duplikaterkennung für eingehende Frames vor.

und $d_{\mathrm{ACK}}$ sind zudem über die Regelungen in Kapitel 9.6 des 802.11-Standards miteinander verknüpft. Ein ACK-Frame wird grundsätzlich mit der selben Modulation übertragen wie der zugehörige DATA-Frame. Als Bitrate wird für den ACK-Frame die größte Bitrate gewählt, die nicht größer ist als die Bitrate des DATA-Frames und nicht als optional für die PHY-Implementierung eingestuft ist. Ein Datenframe, der mit 54 MBit/s übertragen wurde, wird demgemäß mit einem 24 MBit/s-Bestätigungsframe quittiert, da dies die höchste, zwingend zu implementierende Bitrate darstellt.

Vorteilhaft am DATA-ACK-Verfahren ist dessen einfache Umsetzbarkeit. Datenframes sind die einzige Möglichkeit, Nutzdaten zu übertragen. Sie können daher durch einen einfachen Datentransfer erzeugt werden. Ein Zugriff auf spezielle Treiberschnittstellen ist nicht notwendig. Eine Möglichkeit, diesen Datentransfer zu realisieren, stellt z. B. das bereits erwähnte Programm `ping` dar. Nicht nur die Abfolge von ICMP Echo-Request und Echo-Reply stellt einen TOF-Ablauf dar. Sowohl das Echo-Request als auch das Echo-Reply, welche beide in Datenframes transportiert werden, werden mit einem ACK-Frame auf der 802.11-Sicherungsschicht bestätigt. Aus diesem Grund ist das DATA-ACK-Verfahren mindestens gleich mächtig wie das Ping-Verfahren. Durch `ping` intitiierte DATA-ACK-Sequenzen werden u. a. in [GH05] genutzt. Die DATA-ACK-Methode wird weiterhin in [CBAI07] und [LPLaY00] vorgeschlagen.

### 3.3.3 Das NULL-ACK-Verfahren

Das NULL-ACK-Verfahren ist genau genommen ein Spezialfall des eben diskutierten DATA-ACK-Verfahrens. Bei NULL-Frames handelt es sich um einen speziellen Typ von Datenframes, die keine Nutzlast transportieren. Die Frame-Sequenz unterscheidet sich daher nicht von einer DATA-ACK-Sequenz (siehe Abbildung 3.8).

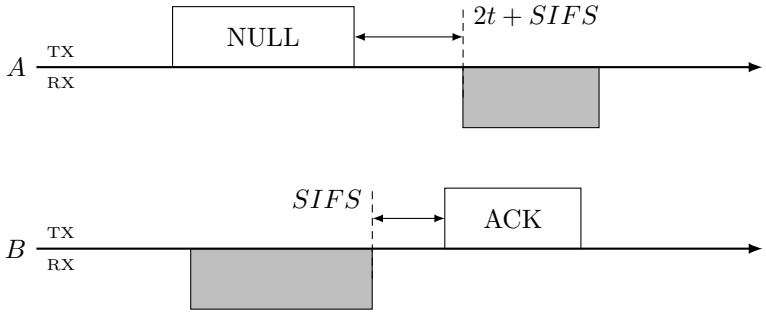



Abbildung 3.8: NULL-ACK-Sequenz

Die einzige Aufgabe von NULL-Frames ist die Übertragung des PM-Bits im FC-Header des MAC-Frames. Dieses Bit zeigt an, dass eine Station beabsichtigt, in den Stromsparmodus zu wechseln, und deshalb ihr WLAN-PHY abschaltet. Andere Stationen werden es daraufhin unterlassen, Pakete an diese Station zu senden, und damit vermeiden, durch aussichtslose Übertragungsversuche das Medium zu belegen. Frames für diese Station werden ggf. beim Sender gepuffert. Der Übergang in den Stromsparmodus ist erfolgreich, wenn auf den NULL-Frame ein Acknowledgement registriert wurde. Natürlich muss das PM-Bit gelöscht sein, sodass die Station $B$ nicht irrtümlich davon ausgeht, dass Station $A$ in den Stromsparmodus wechselt.

NULL-Frames besitzen die Besonderheit, dass ihr „To DS“-Flag im FC-Header immer gelöscht ist. Ihr Ziel ist damit immer eine Station mit einem WLAN-PHY und nie eine hinter dem Distribution System gelegene Station. Wegen der fehlenden Nutzlast kommt für den Frame keine Verschlüsselung zur Anwendung, auch wenn er in einer kryptografisch abgesicherten Zelle übertragen wird. Ein Einfluss des Verschlüsselungsalgorithmus auf die Messwerte kann folglich mit großer Sicherheit ausgeschlossen werden. Ferner ist die Übertragungsdauer $d$ der Frames nur noch vom verwendeten Modulationsverfahren und der Bitrate abhängig. Durch die Nutzlastgröße hervorgerufene Störeinflüsse sind somit ebenfalls nicht zu befürchten.

Dieser Reihe von Vorteilen steht der Fakt gegenüber, dass es mit konventionellen Mitteln nicht so einfach möglich ist, den Austausch von NULL-Frames auszulösen, wie Datenframes durch das Programm `ping` zu erzeugen. Aller Voraussicht nach wird dazu eine spezielle Betriebssystemschnittstelle zum WLAN-Treiber notwendig. Nach Kenntnis des Autors gab es bis dato noch keine Versuche, NULL-Frames zur Laufzeitmessung heranzuziehen.

### 3.3.4 Das RTS-CTS-Verfahren

Das RTS-CTS-Handshake[2] verfolgt den Zweck, das „Hidden Station“-Problem, welches bereits in Abschnitt 2.3.1 kurz thematisiert wurde, weitgehend zu vermeiden. Eine Station $A$, die zu Station $B$ senden will, überträgt vor der eigentlichen Framesequenz einen RTS-Frame. Ist das Medium bei $B$ frei, wird diese nach einer SIFS-Zeit mit einem CTS-Frame antworten (siehe Abbildung 3.9). Hat $A$ bei $B$ eine bereits laufende Übertragung gestört, wird der CTS-Frame ausbleiben. Die gestörte Station wird einen neuen Versuch unternehmen, $A$ hält sich hingegen wegen des ausbleibenden CTS-Frames mit einem erneuten Versuch für eine ausreichend lange Zeit zurück.

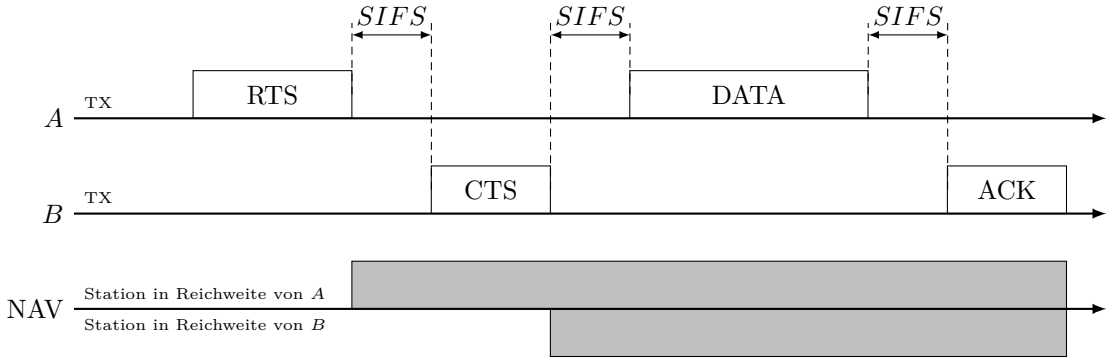



Abbildung 3.9: RTS-CTS-Handshake

Sowohl der RTS-Frame als auch der CTS-Frame führen im Duration-Feld ihres MAC-Headers die Zeit mit sich, für die nach Ende der Aussendung das Medium gesperrt bleiben soll (vgl. Ausführungen zum NAV-Wert in Abschnitt 2.3.1). Nach erfolgreichem RTS-CTS-Handshake ist somit sichergestellt, dass $A$ und $B$ nicht durch dritte Stationen gestört werden, und die eigentliche Datenübertragung folgt nach einer weiteren SIFS-Zeit.

[2] Abk. für „Request to Send“ und „Clear to Send“

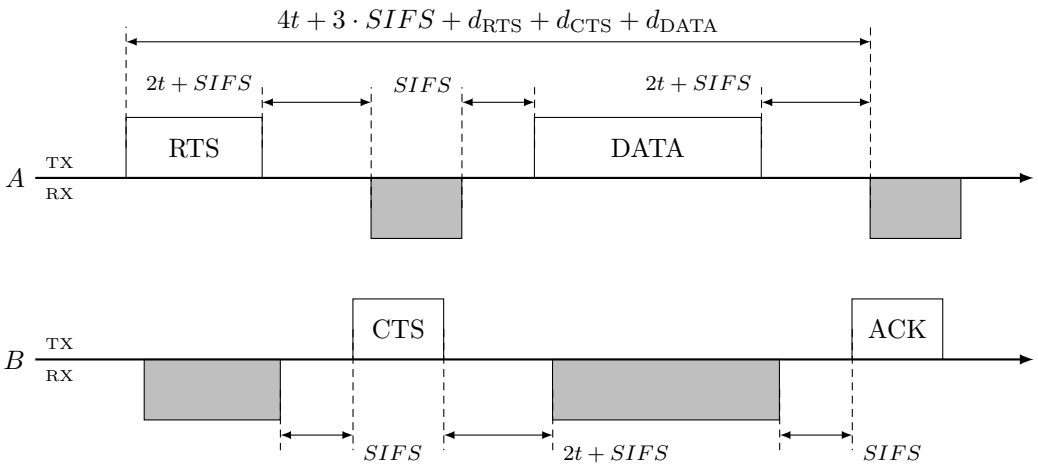



Abbildung 3.10: RTS-CTS-DATA-ACK-Sequenz

Das RTS-CTS-Handshake ist prinzipiell äquivalent zu einer NULL-ACK-Sequenz und verspricht für sich allein genommen keine weiteren Vorteile. Allerdings lässt sich das RTS-CTS-Handshake mit den im Vorfeld beschriebenen Verfahren kombinieren. In Verbindung mit dem DATA-ACK-Verfahren geht die Signallaufzeit $t$ zwischen dem RTS-Frame und dem ACK-Frame insgesamt viermal in den Ausdruck $\Delta$ ein (siehe Abbildung 3.10). Eventuelle Messfehler halbieren sich somit gegenüber dem ursprünglichen Verfahren. Die Bitrate für den CTS-Frame und damit der Wert $d_{\text{CTS}}$ bestimmt sich nach den selben Regeln, wie sie für die Bitrate des ACK-Frames gelten. Eine RTS-CTS-DATA-ACK-Sequenz lässt sich genauso leicht erzeugen wie eine DATA-ACK-Sequenz. Üblicherweise kann über Betriebssystemschnittstellen ein sog. RTS-Threshold festgelegt werden (vgl. Abschnitt 2.4). Überschreitet die Nutzlast eines Datenframes diesen RTS-Schwellwert, wird ein RTS-CTS-Handshake ausgeführt. Es ist möglich, diese Schwelle auf die Größe von 0 Bytes zu setzen, womit stets ein RTS-CTS-Handshake notwendig wird. Das RTS-CTS-Handshake lässt sich ebenfalls mit einer NULL-ACK-Sequenz kombinieren, wobei die Schwierigkeit der Erzeugung einer solchen Sequenz immer noch bestehen bleibt.

Mit der in Abbildung 3.10 beschriebenen Sequenz ist im Übrigen auch $B$ in der Lage, eine Laufzeitmessung vorzunehmen. Der CTS-Frame und der Datenframe stellen aus Sicht von $B$ ein gültiges TOF-Verfahren dar.

Die Möglichkeit, bestehende TOF-Sequenzen durch ein RTS-CTS-Handshake zu ergänzen, wurde unabhängig voneinander in [LPLaY00] und [HW08] vorgeschlagen. Allerdings wurde nur in letztgenannter Quelle eine Implementierung des beschriebenen Verfahrens vorgenommen.

### 3.3.5 IEEE 802.11 Fragmente

Nicht nur das RTS-CTS-Handshake bietet eine Möglichkeit, die Anzahl der Vorkommen von $t$ in $\Delta$ zu vergrößern. Datenframes können in eine Folge von Fragmenten zerlegt werden, die einzeln durch einen ACK-Frame bestätigt werden. Zwischen aufeinanderfolgenden Daten- und ACK-Frames vergeht jeweils eine SIFS-Zeit (siehe Abbildung 3.11).


Abbildung 3.11: Fragmente eines Datenframes

Wann eine Fragmentierung einsetzt, wird durch den Fragmentation-Threshold bestimmt. Überschreitet die Nutzlast diese Größe, wird der Datenframe in Fragmente geteilt, die, mit Ausnahme des letzten Fragments, genau die durch den Schwellwert festgelegte Größe haben. Jedes Fragment wird mit einem eigenen 802.11 MAC-Header versehen. Ist das erste Fragment nicht groß genug, um Schlüsselinformation für kryptografisch gesicherte Zellen aufzunehmen, darf dieses Fragment den Schwellwert überschreiten. Die Fragmentierungsmethode ist prinzipbedingt nur für Datenframes mit vorhandener Nutzlast anwendbar. Es können nur Datenframes fragmentiert werden, die nicht für Broad- oder Multicast-Adressen bestimmt sind. Die Nutzlast eines Fragments soll mindestens 256 Bytes umfassen. Die Zeit $\Delta$ vom Anfang des ersten Fragments bis zum Anfang des letzten Bestätigungsframes berechnet sich durch:

$$\Delta = 2n \cdot t + (2n-1) \cdot SIFS + (n-1) \cdot d_{\text{ACK}} + \sum_{i=0}^{n-1} d_{\text{Fragment } i} \tag{3.20}$$

Mit der RTS-CTS-Methode kann eine weitere TOF-Sequenz in die Frame-Folge aufgenommen werden. Fragmentierung und RTS-CTS-Handshake verfolgen das Ziel, den Faktor von $t$ in $\Delta$ zu vergrößern, und werden deshalb oft gemeinsam diskutiert [LPLaY00, HW08].

### 3.3.6 Einbettung von TX-Zeitstempeln

Das Zeitstempelverfahren nimmt eine Sonderrolle unter den hier beschriebenen Varianten ein. Bisher wurde davon ausgegangen, dass es möglich ist, den Sendezeitpunkt von Frames

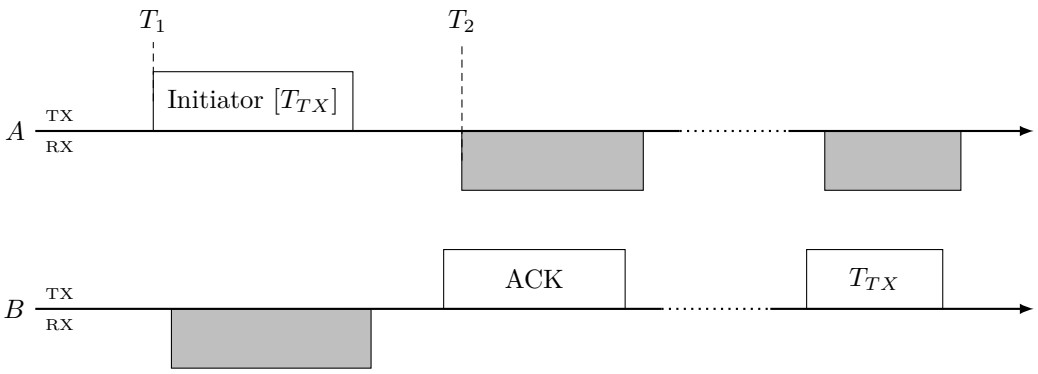



Abbildung 3.12: Austausch von TX-Zeitstempeln

zu bestimmen. Wie sich in Abschnitt 4.2.2 herausstellen wird, ist die Erfassung dieser TX-Zeitstempel mit herkömmlicher Hardware im Allgemeinen nicht möglich. IEEE 802.11 sieht allerdings Frame-Typen vor, in denen der Wert des TSF-Timers zum Zeitpunkt der Aussendung kodiert ist. Gelingt es, wie in Abbildung 3.12 gezeigt, eine TOF-Sequenz auszulösen, in der ein solcher Frame enthalten ist, kann die Signallaufzeit nur durch Empfangszeitstempel bestimmt werden. Dazu wird zunächst der Initiator-Frame versendet, in dem der WLAN-Adpater den Wert des TSF-Zählers vermerkt. Im zweiten Schritt wird der TSF-Zähler beim Eintreffen des Bestätigungsframes festgehalten. Für die Berechnung von $T_2 - T_1$ ist nun aber Wissen über $T_2$ notwendig. Abschnitt 11.1.2 in IEEE 802.11 regelt, wie $T_1$ und $T_{TX}$ zusammenhängen. Letzteren Wert kann $B$ aus dem empfangenen Frame auslesen und in einem dritten Schritt an $A$ übermitteln.

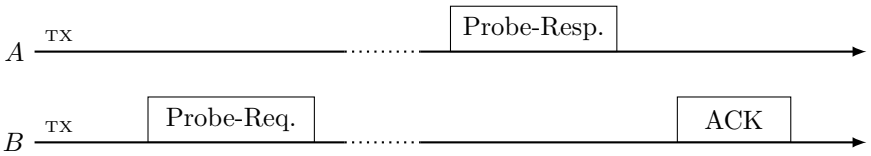



Abbildung 3.13: Probe-Response

Es gibt genau zwei Frames, die einen TSF-Zeitstempel enthalten: Beacon-Frames und Probe-Response-Frames. Beide verbreiten Management-Informationen über die Funkzelle (ESSID, unterstützte Datenraten, etc.) an alle Stationen im Umkreis. Die Beacon-Frames werden in regelmäßigen Zeitabständen (für gewöhnlich etwa alle 102.4 ms) als Broadcast-Aussendung ausgestrahlt. Probe-Response-Frames verteilen die selben Informationen wie Beacon-Frames, werden jedoch durch einen Probe-Request-Frame abgefordert. Probe-Response-Frames werden durch ACK-Frames nach einer SIFS-Zeit bestätigt und eignen sich daher für ein TOF-Verfahren. Die Schwierigkeit dieses Verfahrens besteht jedoch in der Auslösung einer solchen Sequenz, da $B$ ein Probe-Request initiieren muss, damit für $A$ eine nutzbare TOF-Sequenz entsteht. Ggf. lässt sich sich aber der Mechanismus, der die Zeitstempel in den Frames einbettet, auch auf andere Frames anwenden. Allerdings besteht dabei die Gefahr, die Grenzen des Standards IEEE 802.11 zu überschreiten.

Beacon-Frames werden in [WKP09] für ein TDOA-Verfahren genutzt. Allerdings sind die verwendeten Frame-Sequenzen nicht atomar, wodurch die Frame-Abstände so groß werden, dass der Uhrendrift Fehler bis zu 5 m hervorruft. Die Autoren von [VK04] schlagen Beacon-Frames zur Durchführung eines TOA-Verfahrens vor, bezweifeln aber dessen praktische Umsetzbarkeit.

## 3.4 Vergleich der Messverfahren

Vor der Umsetzung der im vergangenen Abschnitt dargelegten Methoden sollen diese durch gegenseitigen Vergleich auf ihre praktische Eignung untersucht werden. Zunächst lassen sich die Inklusionsbeziehungen (3.21) aufstellen. Der Ausdruck $A \sqsubset B$ bedeutet hierbei, dass $A$ ein Spezialfall von $B$ ist und dass alle Einschränkungen für $A$ auch für $B$ gelten. Die Inklusionen ergeben sich durch das Auslassen von Fragmenten, RTS-CTS-Handshakes oder dem Entfernen von Nutzlasten. Das Ping-Verfahren erzeugt stets eine DATA-ACK-Sequenz.

$$\begin{array}{ccccc} \text{RTS-CTS-NULL-ACK} & \sqsubset & \text{RTS-CTS-DATA-ACK} & \sqsubset & \text{RTS-CTS-Frag.} \\ \sqcup & & \sqcup & & \sqcup \\ \text{NULL-ACK} & \sqsubset & \text{DATA-ACK} & \sqsubset & \text{Frag.} \\ & & \sqcap & & \\ & & \text{Ping} & & \end{array} \tag{3.21}$$

Prinzipiell zeichnet sich bereits das NULL-ACK-Verfahren als guter Kandidat für eine Implementierung ab. Zunächst soll jedoch ein differenzierterer Vergleich weitere Unterschiede und Gemeinsamkeiten der Verfahren hervorheben.

**Welche Besonderheiten treten in Infrastrukturnetzen auf?**

In Infrastrukturnetzen ist der Access Point nie Ziel oder Quelle eines Datenframes. Es ist somit stets mindestens eines der Flags „To DS“ und „From DS“ gesetzt. Ein solcher Frameaustausch erfordert immer, dass sich die Station zuvor mit dem Access Point assoziiert hat. NULL-Frames sind hingegen direkt an den Access Point gerichtet und erfordern demnach im Vorfeld keine zeitaufwändige Assoziierung.

Sinn eines Access Point ist es, Datenframes zwischen Drahtlosmedium und Distribution System zu vermitteln. Für das Aussenden von NULL-Frames durch einen Access Point besteht keine Notwendigkeit, weil ein Access Point nicht in den Stromsparmodus wechselt und seine Sendetätigkeit einstellt. Prinzipiell ist eine solche Framesequenz dennoch gültig, allerdings ist das Verhalten der Zielstation nicht spezifiziert. Sofern die MLME eines Access Point keine spezielle Möglichkeit vorsieht, kann das NULL-ACK-Verfahren folglich nicht an diesem AP initiiert werden. Eine Fremdlokalisierung ist also nur mit assoziierten Stationen über das DATA-ACK-Verfahren möglich.

**Wo müssen welche Größen gemessen werden?**

Mit Ausnahme des Ping-Verfahren sind nur Zeitstempelmessungen beim eigentlichen Initiator notwendig. Das Ping-Verfahren erfordert eine Messeinrichtung an der Stelle, an der das Echo-Request empfangen und das Echo-Reply gesendet wird, um den genauen Wert von $g$ zu bestimmen. Dieser Messwert muss dem Initiator anschließend über einen geeigneten Kanal zugestellt werden.

Um den Wert $d$ bestimmen zu können, müssen die zur Anwendung gekommenen Datenraten und Modulationsverfahren der Datenframes bzw. RTS-Frames aufgezeichnet werden. Die Datenraten für ACK- und CTS-Frames ergeben sich aus diesen Messungen.

Bei Verfahren mit DATA- oder NULL-Frames sind sowohl Sende- als auch Empfangszeitpunkte bei $A$ festzuhalten. Beim Zeitstempelverfahren wird der Sendezeitpunkt durch die MLME festgehalten, sodass im weiteren Verlauf nur noch Empfangszeitpunkte protokolliert werden müssen, was von vielen WLAN-Adaptern ohne Weiteres unterstützt wird. Der in den Initialframe eingesetzte Zeitstempel muss vom Empfänger oder einer beliebigen anderen Station in Reichweite

protokolliert werden. Der Ort dieser Station bzw. Signallaufzeiten zu dieser Station haben keinen Einfluss, da nur der Inhalt dieses Frames von Interesse ist, und nicht sein Empfangszeitpunkt.

**Welche Störeinflüsse wirken auf $\Delta$?**

Die Messgröße $\Delta$ setzt sich aus den drei Größen $t$, $d$ und $g$ zusammen. Um möglichst genaue Aussagen über $t$ treffen zu können, ist es wichtig, störende Einflüsse auf $d$ und $g$ weitestgehend zu eliminieren. Das Modulationsverfahren und die verwendeten Bitraten haben Einfluss auf beide Größen. Von daher muss bei der späteren Umsetzung vorrangig eine Möglichkeit gefunden werden, Bitrate und Modulation von Aussendungen gezielt beeinflussen zu können.

Auf die prinzipbedingte Störung des $g$-Werts beim Ping-Verfahren wurde bereits in der entsprechenden Erläuterung eingegangen. Bei allen anderen vorgeschlagenen Verfahren ist die Bitrate die einzige Einflussgröße auf $g$, sieht man von unvermeidlichen Hardwaretoleranzen ab. Diese Verfahren sind deshalb bereits im Vorteil gegenüber dem Ping-Verfahren.

Die Länge von RTS-, CTS- und ACK-Frames ist konstant und führt somit nicht zu einem Schwanken der $d$-Werte. Datenframes sind in ihrer Länge variabel. Nutzt man jedoch das Programm `ping` zur Erzeugung von DATA-ACK-Sequenzen, lässt sich die Länge der Ethernet-Frames und damit auch die Länge der 802.11-MAC-Frames festlegen. Bei der Beobachtung von DATA-ACK-Sequenzen des regulären Datenverkehrs der eigenen Station ist dies im Allgemeinen nicht gegeben. NULL-Frames transportieren keine Nutzlast und haben dadurch immer eine konstante Länge. Die Länge von Probe-Response-Frames des TX-Zeitstempelverfahrens ist von der Konfiguration der Station $B$ abhängig, da diese Frames Datenfelder mit variabler Länge (z. B. die ESSID oder einen Vektor unterstützter Bitraten) enthalten. Frames mit Nutzlast erfahren ggf. eine Verschlüsselung. Die in die Nutzlast eingefügten Schlüsselvektoren wirken sich ebenfalls auf $d$ aus.

Wegen der fehlenden Nutzlast erscheint das (RTS-CTS-)NULL-ACK-Verfahren besonders geeignet, weil hier die meisten Störeinflüsse ausgeschlossen werden können.

**Welche Nebeneffekte entstehen für das Drahtlosmedium?**

Das Drahtlosmedium wird in jedem Fall durch die Übertragung von Frames belastet. Es stellt sicher daher die Frage, welches Verfahren das Medium möglichst effizient ausnutzt. Die auf Datenframes basierenden Verfahren (Ping, DATA-ACK) erzeugen realtiv umfangreiche Datenframes. Die Kodierung von Ethernet-Frames innerhalb der MAC-SDU erfolgt über das IEEE 802.3 LLC-Protokoll. Insgesamt ist damit die MAC-PDU eines leeren Ethernet-Frames bereits mindestens 36 Byte lang. Ein darin transportieres ICMP-Echo-Paket umfasst nochmals 84 Byte. Bei Anwendung des Ping-Verfahrens entstehen jeweils zwei ICMP- und zwei ACK-Frames, wovon jeweils entweder nur beide ICMP-Frames oder ein ICMP-ACK-Paar verwendet werden können.

Bei einem RTS-CTS-Handshake und der NULL-ACK-Sequenz werden nur die für das TOF-Verfahren notwendigen Frames übertragen. Sie sind wegen der fehlenden Nutzlast viel kürzer als Datenframes. Die RTS-CTS-Sequenz stellt die kürzestmögliche TOF-Sequenz dar. Der CTS-Frame enthält nur die obligatorischen Felder des MAC-Headers. Der RTS-Frame enthält zusätzlich zu den obligatorischen Feldern nur die Transmission Address, die aber für die Receiver Address des CTS-Frames zwingend bei Station $B$ vorliegen muss. Ein NULL-Frame ist lediglich 8 Byte größer als ein RTS-Frame.

Bei einem RTS-CTS-Handshake ohne weitere Frame-Sequenzen ist darauf zu achten, dass der Duration-Wert im RTS-Frame so gesetzt wird, dass das Medium nach dem CTS-Frame nicht unnötig lang gesperrt wird und dadurch andere Stationen behindert werden.

Das Fragmentierungsverfahren erscheint zunächst besonders vorteilhaft, da die Größe $t$ mehrfach in $\Delta$ eingeht. Dieser Vorteil wird sich jedoch damit erkauft, dass bei Anwendung von Fragmentierung das Medium für eine sehr lange Zeit belegt wird. Jedes Fragment erfordert die Übertragung von zwei PLCP- und MAC-Header, jeweils einmal für das Datenfragment und ein zweites Mal für den ACK-Frame. Die PLCP-Header werden zudem noch mit der niedrigstmöglichen Bitrate übertragen und erfordern daher besonders viel Zeit. Die niedrigste Fragmentgröße wird durch IEEE 802.11 mit 256 Byte definiert, was bereits die Verwendung sehr langer Datenframes erzwingt. Aus diesen Gründen bricht die Nettodatenrate bei Drittstationen stark ein.

Probe-Response-Frames des Zeitstempelverfahrens sind vom Datenaufkommen her vergleichbar mit Datenframes, da sie diverse Informationen über die Funkzelle verbreiten. In Infrastrukturnetzen sind Framegrößen von 180 Byte keine Seltenheit. Aus diesem Grund empfiehlt es sich eine Möglichkeit zu suchen, den Zeitstempel, der immer in den ersten 8 Byte kodiert ist, in andere Frames einzubetten.

**Welche Nebeneffekte entstehen für das Distribution System?**

Das Distribution System existiert nur in Infrastrukturnetzwerken, da es Access Points untereinander oder einen Access Point mit einem leitungsgebundenen Netzwerk verknüpft. Es wird prinzipiell nur beim Austausch von Datenframes involviert. Eventuelle Spezialprotokolle, die bestimmte Statusinformationen des Access Points über das Distribution System melden, bleiben von dieser Aussage natürlich unberührt. NULL-Frames, Probe-Responses und RTS-CTS-Handshakes erzeugen keine Belastung des Distribution System. Um somit unnötige Last zu vermeiden, sollten daher bevorzugt das NULL-ACK-Verfahren, RTS-CTS-Handshakes oder ein Zeitstempelverfahren zum Einsatz kommen.

Zusammenfassend lässt sich feststellen, dass Nicht-Datenframes für das TOF-Verfahren am geeignetsten erscheinen, da sie die wenigsten Nebeneffekte auf das bestehende Netzwerk haben, keine aufwändigen Managementoperationen erfordern und viele Störeinflüsse ausschließen. Demgegenüber steht die Schwierigkeit, solche TOF-Sequenzen gezielt zu erzeugen.

## 3.5 Das Drei-Teilnehmer-Modell

In den bisherigen Betrachtungen waren jeweils nur zwei Teilnehmer in den Ablauf des TOF-Verfahrens involviert. Wie sich im nächsten Kapitel herausstellen wird, ist es dem Initiator $A$ nicht immer möglich, die Zeit zu bestimmen, zu der er ein Paket versendet. Aus diesem Grund bietet es sich bereits zu diesem Zeitpunkt an, drei Teilnehmer in das TOF-Verfahren zu involvieren, von denen jeder eine der folgenden Funktionen erfüllt.

- $A$ ist der Initiator und sendet ein Paket an $B$ aus.
- $B$ ist der Reflektor und quittiert das Paket von $A$ mit einem Antwortpaket.
- $M$ fungiert als Monitor und beobachtet den Datenaustausch zwischen $A$ und $B$. Der Monitor ist vollkommen passiv, d. h. er versendet keine Daten.

Im allgemeinen Fall befinden sich $A$, $B$ und $M$ an verschiedenen Positionen. Der Abstand zwischen $X$ und $Y$ sei dabei durch $r_{X,Y}$ bezeichnet ($X, Y \in \{A, B, M\}$). Die Signallaufzeiten $t_{X,Y}$ für den Fall, dass die Fresnelzone zwischen $X$ und $Y$ nicht verdeckt ist, ergibt sich dann aus $t_{X,Y} = \frac{r_{X,Y}}{c}$. Für unverdeckte Fresnelzonen gilt weiterhin noch die Dreiecksungleichung

$$\forall X, Y, Z : r_{X,Y} \leq r_{X,Z} + r_{Z,Y}.$$

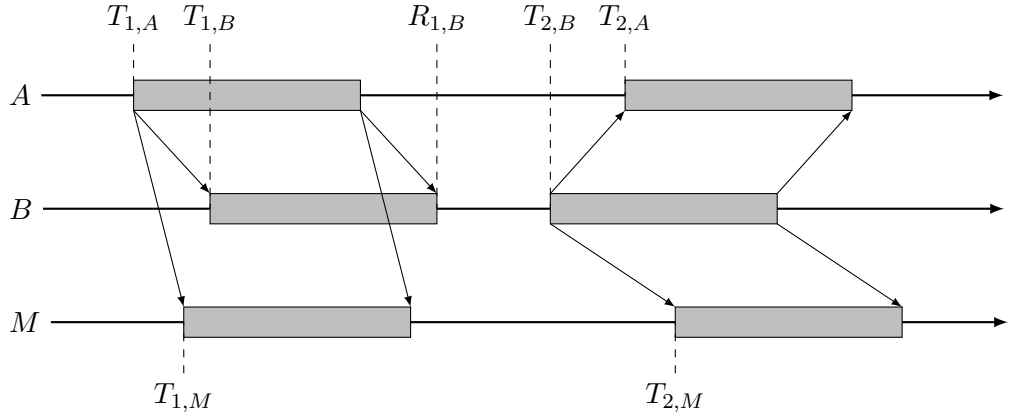



Abbildung 3.14: Ablauf des TOF-Verfahren zwischen Initiator, Reflektor und Monitor

$A$ und $B$ führen nun das in Abschnitt 3.1 beschriebene TOF-Verfahren aus (siehe Abbildung 3.14). Gleichung (3.4) stellt sich dann als

$$T_{2,A} - T_{1,A} = d + g + 2t_{A,B} \tag{3.22}$$

dar. Der Ausdruck $T_{2,M} - T_{1,M}$ entspricht der zeitlichen Differenz der selben Ereignisse wie in Gleichung (3.22), nur von $M$ aus beobachtet.

$$\begin{aligned} T_{2,M} - T_{1,M} &= (T_{2,B} + t_{B,M}) - (T_{1,A} + t_{A,M}) \\ &= ((T_{2,A} - t_{A,B}) + t_{B,M}) - (T_{1,A} + t_{A,M}) \\ &= T_{2,A} - T_{1,A} - t_{A,B} + t_{B,M} - t_{A,M} \\ &= d + g + t_{A,B} + t_{B,M} - t_{A,M} \end{aligned} \tag{3.23}$$

Das zwischen $A$ und $B$ ausgeführte TOF-Verfahren ist für $M$ folglich ein TDOA-Verfahren. Auf genau synchronisierte Uhren zwischen $A$ und $B$ kann verzichtet werden, da der erste Frame der TOF-Sequenz die notwendige Synchronisierung herbeiführt.

Eine Ausbreitung auf Sichtlinie muss aber nicht in jedem Fall gegeben sein, wie aus Abbildung 3.15 deutlich wird. Der Weg zwischen $A$ und $M$ wirkt durch ein Hindernis stark dämpfend. Werden die Signale von $A$ durch ein weit entferntes Objekt zu $M$ reflektiert, wird die Signallaufzeit größer, als sie der Strecke $r_{A,B} + r_{B,M}$ entspricht.

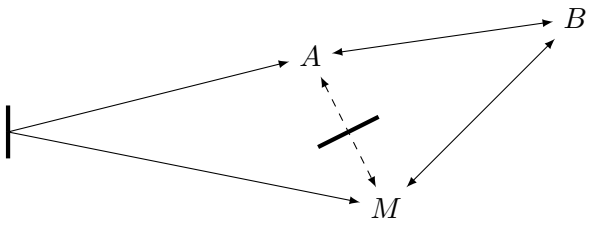



Abbildung 3.15: Indirekte Ausbreitung von Funksignalen

Für die tatsächlichen Signallaufzeiten $t'_{X,Y}$ gilt dann die Ungleichung

$$t'_{X,Y} \geq t_{X,Y}.$$

Unter der Bedingung $r_{B,M} \approx 0$ können wir allerdings davon ausgehen, dass $M$ über den selben – aber nicht notwendigerweise direkten – Weg von $A$ empfängt wie $B$. Gleiches gilt für die Vertauschung von $A$ und $B$ im letzten Satz. Unter Verwendung der obigen Abschätzung geht Gleichung (3.22) in die Ungleichung

$$T_{2,A} - T_{1,A} \geq d + g + 2t_{A,B} \tag{3.24}$$

über. Wegen des negativen Vorzeichens vor $t_{A,M}$ kann für Gleichung (3.23) keine Abschätzung wie im Fall von Gleichung (3.22) vorgenommen werden. Besteht zwischen $A$ und $M$ eine Sichtverbindung, kann $t'_{A,M} = t_{A,M}$ angenommen werden. Nehmen die Signale auf der Strecke $\overline{AB}$ und $\overline{BM}$ „ähnliche Wege“, lässt sich Gleichung (3.23) mit

$$T_{2,M} - T_{1,M} \geq d + g + t_{A,B} + t_{B,M} - t_{A,M} \tag{3.25}$$

abschätzen.

## 3.6 Einsatzszenarien

Bisher wurde nur betrachtet, auf welche Weise ein TOF-Verfahren zwischen den Stationen $A$ und $B$ ausgeführt werden kann, und wie sich dieses aus Sicht von $M$ darstellt. Welche Rollen die Stationen in dem Verfahren letztendlich einnehmen, wurde nicht festgelegt. Prinzipiell gibt es zwei Kategorien von Stationen.

- Mobile Stationen sind Stationen, deren Position unbekannt ist.
- Referenzstationen sind Stationen mit bekannten Koordinaten, die als Navigationsbasis für die Lokalisierung mobiler Stationen dienen können.

In der einfachsten Konfiguration ist $A$ die Mobilstation und $B_i$ stellen Referenzpunkte dar (siehe Abbildung 3.16). $A$ kann durch TOF-Messungen die Signallaufzeit und damit auch die Abstände zu jeder der $B$-Stationen bestimmen. In geometrischer Interpretation liegt $A$ folglich auf einem Kreis um jedes der $B_i$, dessen Radius $r_i$ von der Signallaufzeit abhängt. Liegen ausreichend Messungen vor, ist mittels der in Abschnitt 2.2 dargelegten Gleichungen eine Positionsbestimmung durch Trilateration möglich. Die Zellen der $B_i$ müssen sich jedoch ausreichend überlappen, damit $A$ hinreichend viele Messungen vornehmen kann. In Infrastrukturnetzwerken eignen sich Access Points als feste Referenzpunkte. Prinzipiell erfüllt aber auch jede andere Station – z. B. Stationen in Ad-hoc-Netzwerken – diesen Zweck, sofern ihre Position bekannt ist.

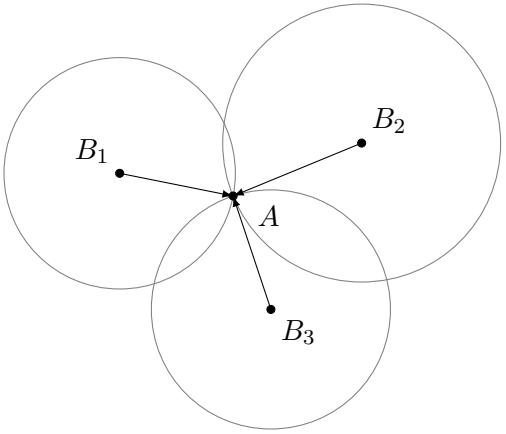



Abbildung 3.16: Trilateration durch Messung der Abstände $r_{A,B_i}$

Der Initiator $A$ und die Reflektoren $B_i$ können gegeneinander ausgetauscht werden. Für den Fall, dass es sich bei den $B_i$ um Access Points handelt, müssen die in diesem Kapitel beschriebenen Messmethoden am Access Point initiierbar sein. Mittels einfacher Trilateration ist somit sowohl die Eigenlokalisierung als auch die Fremdlokalisierung möglich.

Ist die Überdeckung der Funkzellen nicht ausreichend für eine Trilateration, können noch weitere Monitor-Stationen als Referenzpunkte herangezogen werden. Durch ihre Passivität vergrößern sie den Störpegel anderer Stationen nicht und können in beliebiger Dichte platziert werden. Auf diese Weise lässt sich zumindest zwischen $A$ und den Monitor-Stationen eine Sichtverbindung garantieren, indem die $M$-Stationen in regelmäßigen Abständen auf Gängen oder den Ecken von Räumen installiert werden.

Die Stationen $M_i$ werten die in Abschnitt 3.5 hergeleitete Monitor-Gleichung (3.23) aus. In den $\Delta$-Werten kommt nun die Differenz der beiden Unbekannten $t_{A,B}$ und $t_{A,M_i}$ vor.

$$\Delta_i = d + g + t_{A,B} + t_{B,M_i} - t_{A,M_i} \tag{3.26}$$

Der Wert $t_{B,M_i}$ ist konstant, da es sich bei $B$ und den $M_i$ um Referenzstationen handelt. Die Monitor-Gleichung ist genau dann erfüllt, wenn sich $A$ auf der $(B, M_i)$-Hyperbel mit dem Parameter $\Delta_i - (d + g + t_{B,M_i})$ befindet (siehe Abbildung 3.17).

Zur Positionsbestimmung kann weiterhin das in Abschnitt 2.2 beschriebene Gleichungssystem (2.8) genutzt werden, wenn ein weiterer Freiheitsgrad $\tau$ eingeführt wird. Für geeignete Wahl

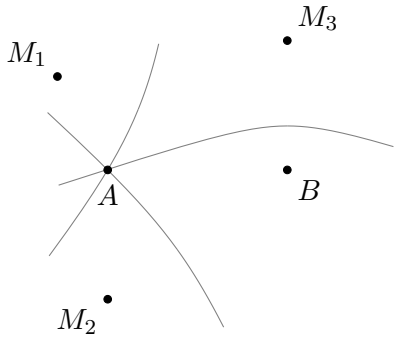



Abbildung 3.17: Hyperbelnavigation mit mehreren Monitoren

von $\tau$ gilt:

$$\tau - \Delta_i = t_{A,M_i} \tag{3.27}$$

Aus dieser Größe kann die Länge der Strecke $\overline{AM_i}$ für eine anschließende Trilateration abgeleitet werden. Zu beachten ist nur, dass gegenüber der Trilateration eine zusätzliche Messung notwendig wird, um eine eindeutige Lösung des Gleichungssystems zu erhalten.

Prinzipbedingt ist diese Art der Hyperbelnavigation nur für die Fremdlokalisierung verwendbar, da die Messwerte an mehreren Stellen erfasst werden müssen. In modifizierter Version ist die Hyperbelnavigation dennoch zur Eigenlokalisierung nutzbar. In diesem Fall sind allerdings mehrere $B$-Stationen als Referenzpunkte notwendig, und der einzige Monitor $M$ stellt das Mobilgerät dar.

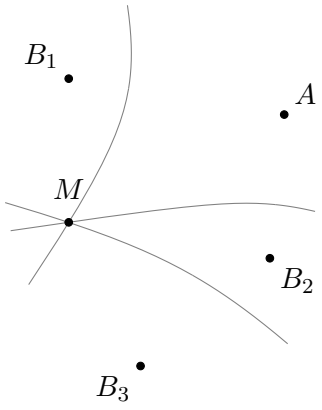



Abbildung 3.18: Hyperbelnavigation mit mehreren TOF-Sequenzen

In der Monitorgleichung ist nun $t_{A,B_i}$ konstant und bekannt. Die Differenz $t_{B_i,M} - t_{A,M}$ ist von der Position von $M$ in Bezug auf die Strecke $\overline{AB_i}$ abhängig. Die Station $M$ kann durch Messung den Parameter ihrer $(A, B_i)$-Hyperbel bestimmen. Analog zum Multi-Monitor-Szenario kann folgender Ansatz zur Lösung der Trilaterationsgleichungen benutzt werden.

$$\Delta_i - \tau = t_{B_i,M} \tag{3.28}$$

Dieses Szenario stellt in gewissem Sinne das Gegenteil zum Multi-Monitor-Szenario dar. Es ist nur für die Eigenlokalisierung verwendbar und erfordert weiterhin überlappende Funkzellen bei der Station $M$, was eine eher unerwünschte Situation in WLAN-Infrastrukturen ist.

# Kapitel 4

# Technische Umsetzung

Die im vorangegangenen Kapitel beschriebenen Methoden können für die praktische Umsetzung in drei einzelne Schritte gegliedert werden, nach denen sich auch die Strukturierung dieses Kapitels richtet.

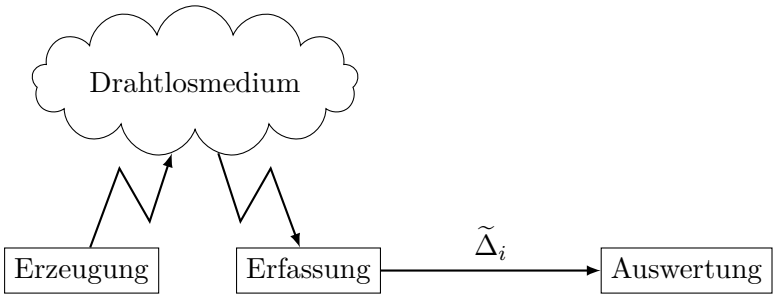



Abbildung 4.1: Basis-Schritte für eine Laufzeitmessung

**Messwerterzeugung** Zuerst muss durch entsprechende technische Möglichkeiten eine TOF-Sequenz, wie sie in Abschnitt 3.3 beschrieben ist, erzeugt werden.

**Messwerterfassung** Gleichzeitig zur Erzeugung dieser TOF-Sequenzen muss die Erfassung der $\Delta$-Werte stattfinden. Der Ort der Messung wird sich je nach Einsatzszenario vom Ort der Messwerterzeugung unterscheiden.

**Messwertauswertung** Die Auswertung der erfassten Messwerte kann zeitlich nach den ersten beiden Schritten erfolgen. Mittels der in Abschnitt 3.2 gemachten Überlegungen kann eine Schätzung für die Signallaufzeit angegeben werden. Sofern möglich sind ggf. Fehlerquellen zu korrigieren, die bei den theoretischen Betrachtungen im vorherigen Kapitel außer Acht gelassen wurden.

## 4.1 Messwerterzeugung

Die wesentlichen Aufgaben der Implementierungen von Messwerterzeugungsverfahren sind:

- die entsprechende Sequenz auf dem Drahtlosmedium auszulösen,
- sicherstellen, dass die Sequenz in der gewünschten Datenrate abgewickelt wird, um störende Effekte auszuschließen und
- ggf. ein RTS-CTS-Handshake zu veranlassen.

Aufgrund der großen Ähnlichkeit zwischen Ping- und DATA-ACK-Verfahren werden beide gemeinsam in einem Abschnitt abgehandelt. Bei den Verfahren, wo ein RTS-CTS-Handshake oder Fragmentierung möglich ist, wird im entsprechenden Abschnitt des Basisverfahrens darauf eingegangen.

### 4.1.1 Das Ping- und DATA-ACK-Verfahren

Das Ping-Verfahren und damit auch ein spezielles DATA-ACK-Verfahren kann recht einfach mit üblichen Bordmitteln eines Linux-Systems umgesetzt werden. Zunächst muss das WLAN-Interface aktiviert und einer Funkzelle beigetreten werden:

```
# Infrastruktur-Netzwerke
$ ip link set ⟨Interface⟩ down
$ iw dev ⟨Interface⟩ set type managed
$ ip link set ⟨Interface⟩ up
$ iw dev ⟨Interface⟩ set channel ⟨Kanal⟩
$ iw dev ⟨Interface⟩ connect ⟨SSID⟩

# Ad-hoc-Netzwerke
$ ip link set ⟨Interface⟩ down
$ iw dev ⟨Interface⟩ set type ibss
$ ip link set ⟨Interface⟩ up
$ iw dev ⟨Interface⟩ ibss join ⟨SSID⟩ ⟨Frequenz⟩
```

Anschließend muss eine IP-Adresse für das Interface konfiguriert werden, da das ICMP-Protokoll auf dem IP-Protokoll aufsetzt. Die Konfiguration kann entweder manuell oder automatisiert durch das DHCP-Protokoll erfolgen.

Bevor die TOF-Sequenzen erzeugt werden, muss dem Rate-Control-Algorithmus noch eine Bitmaske erlaubter Bitraten übergeben werden. Die Masken werden für jedes Frequenzband der IEEE 802.11abg-PHYs getrennt gesetzt.

```
# 2.4 GHz-Band
$ iw dev ⟨Interface⟩ set bitrates legacy-2.4 ⟨Bitrate⟩

# 5 GHz-Band
$ iw dev ⟨Interface⟩ set bitrates legacy-5   ⟨Bitrate⟩
```

Das Einstellen von Bitraten für 802.11n-PHYs ist derzeit noch nicht möglich.[1] Beim Umgang mit Bitraten-Masken sind jedoch einige Nebeneffekte zu beachten. Die Bitraten-Einstellung wirken sich nicht nur auf die Frames der TOF-Sequenz aus, sondern auch auf den restlichen

[1] http://wireless.kernel.org/en/users/Documentation/iw#Modifying_transmit_bitrates

Datenverkehr, der über das Interface abgewickelt wird. Es greifen ferner nicht alle Treiber auf den Rate-Control-Algorithmus des MAC80211-Subsystems zurück. Der Treiber `iwlagn` implementiert seinen eigenen Algorithmus. Die Firmware der Geräte, die über den Treiber `ath9k_htc` angesprochen werden, implementieren die Auswahl der Bitrate in Hardware und bieten keine Möglichkeit für eine softwareseitige Modifikation.

Nach diesen Vorbereitungsmaßnahmen können mit dem Programm `ping` eine Reihe von DATA-ACK-Sequenzen ausgelöst werden.

```
# ohne Wartezeit zwischen den einzelnen Sequenzen
$ ping -I ⟨Interface⟩ -f            -c ⟨Anzahl Messungen⟩ ⟨Zielhost⟩

# mit Wartezeit zwischen den einzelnen Sequenzen
$ ping -I ⟨Interface⟩ -i ⟨Wartezeit⟩ -c ⟨Anzahl Messungen⟩ ⟨Zielhost⟩
```

Der Zielhost muss auf alle Fälle erreichbar sein, da sonst seine IP-Adresse nicht in eine MAC-Adresse aufgelöst werden kann und keine Ping-Sequenzen übertragen werden. Die Routing-Einstellungen des lokalen Hosts sind so zu konfigurieren, dass der Zielhost über das angegebene Interface erreichbar ist.

Nach Ende der Sequenzen können die Bitraten-Masken gelöscht werden, um eine Beeinträchtigung des regulären Datenverkehrs zu vermeiden.

```
$ iw dev ⟨Interface⟩ set bitrates
```

Den Datenframes lässt sich ein RTS-CTS-Handshake voranstellen, indem man den RTS-Threshold auf 0 setzt. Auch hier ist zu beachten, dass diese Einstellung für alle anderen Frames ein RTS-CTS-Handshake veranlasst und die Nettodatenrate entsprechend beschränkt. Diese Einstellung gilt ferner für alle virtuellen Schnittstellen (VIF) eines PHYs. Zunächst muss der Identifikator der PHYs für ein VIF ermittelt werden

```
$ iw dev
phy#0
        Interface wlan0
                ifindex 5
                type managed
```

Im Beispiel ist `phy0` das PHY von Interface `wlan0`. Der RTS-Threshold kann anschließend über folgende Kommandos gesetzt werden.

```
$ iw phy ⟨PHY⟩ set rts 0      # RTS-CTS-Handshake aktivieren
$ iw phy ⟨PHY⟩ set rts off    # RTS-CTS-Handshake deaktivieren
```

Auf die gleiche Weise kann eine Fragmentierung der Frames herbeigeführt werden. Auch diese wirkt auf alle Frames, die das entsprechende PHY verlassen. Die minimale Fragmentgröße beträgt 256 Byte und kann über folgende Kommandos eingestellt werden.

```
$ iw phy ⟨PHY⟩ set frag 256   # Fragmentierung aktivieren
$ iw phy ⟨PHY⟩ set frag off   # Fragmentierung deaktivieren
```

Um eine tatsächliche Fragmentierung zu erreichen, müssen alle ICMP-Echo-Requests eine entsprechende Größe besitzen. In das erste Fragment gehen 20 Byte für den IPv4-Header und 8 weitere Byte für den Kopf des ICMP-Pakets ein. Für jedes weitere Fragment muss das jeweils nächste Vielfache von 256 Byte überschritten werden. Die Anzahl der Bytes, die nach dem ICMP-Header folgen, kann dem `ping`-Kommando mit dem `s`-Parameter übergeben werden. Folgendes Kommando erzeugt eine Frame-Sequenz aus $n$ Fragmenten.

`$ ping -I` ⟨*Interface*⟩ `-s` ⟨$n \cdot 256 - 27$⟩ `-f -c` ⟨*Anzahl Messungen*⟩ ⟨*Zielhost*⟩

Auf Grund der beschränkten Größe der MAC-PDUs und der minimalen Fragmengröße kann eine MAC-SDU maximal in neun Fragmente aufgeteilt werden.

### 4.1.2 Das NULL-ACK-Verfahren

Wie bereits im vorhergehenden Kapitel angemerkt wurde, gibt es in einem Linux-System kein übliches Verfahren, eine ausreichende Menge NULL-Frames zu erzeugen. Der Linux-Kern bietet allerdings die Möglichkeit, beliebige IEEE 802.11 Frames zu versenden. Diese Technik wird als „Frame-Injection“ bezeichnet, und ihre Funktionsweise wird in der Kernel-Dokumentation beschrieben.[2] Den Frames muss ein Radiotap-Header vorangestellt werden. Ein Parser innerhalb des Kernels wertet alle für das Versenden von Frames notwendigen Radiotap-Felder (siehe Tabelle 2.2) aus und gibt diese Informationen an den WLAN-Treiber weiter. Diese Frames müssen über einen Socket, der an ein Monitor-VIF gebunden ist, in den Kern eingeschleust werden.

Über den Radiotap-Header sollen im Weitern folgende Informationen an das MAC80211-Subsystem übermittelt werden können.

1. Der Kern soll die Berechnung der Frame Check Sequence übernehmen.
2. Der Kern soll den Rate-Control-Algorithmus umgehen und eine spezifizierte Bitrate zur Übertragung verwenden.
3. Der Kern soll im Falle von 802.11b-Frame die Short Preamble verwenden.
4. Der Kern soll der eigentlichen Framesequenz ein RTS-CTS-Handshake vornehmen.

Das Parsen eines eingeschleusten Radiotap-Headers geschieht innerhalb der Funktion **static** `bool __ieee80211_parse_tx_radiotap()` in der Datei `net/mac80211/tx.c`. Die Unterstützung der definierten Radiotap-Felder ist jedoch hochgradig unvollständig. In der aktuellen Version 2.6.37 wird nur das Flags-Feld ausgewertet, was lediglich Punkt 1 der oben stehenden Liste erfasst. Zur Festlegung der Bitraten existieren Zusatzpatches.[3,4] Der Patch aus Fußnote 3 unterstützt dabei nur Bitraten nach IEEE 802.11abg. Der Patch in Fußnote 4 enthält auch die Unterstützung für die MCS-Indizes nach IEEE 802.11n. Im Rahmen dieser Arbeit kam der letztgenannte Patch zur Anwendung. Das von diesem Patch definierte MCS-Format im Radiotap-Header wurde zwischenzeitlich allerdings zu Gunsten eines allgemeineren Formats verworfen.[5] Ferner stellte sich heraus, dass der Patch für konventionelle Datenraten größer als

[2] `http://www.kernel.org/doc/Documentation/networking/mac80211-injection.txt`
[3] `http://thread.gmane.org/gmane.linux.kernel.wireless.general/47441`
[4] `https://patchwork.kernel.org/patch/118564/`
[5] vgl. `http://www.radiotap.org/rejected-fields/MCS`

1 MBit/s fehlerhaft war und ein Korrekturpatch erforderlich wurde (siehe Anhang B.1.1). Es ist jedoch zu beachten, dass diese Patches nur den Rate-Control-Algorithmus des MAC80211-Subsystems umgehen. Eine treibereigene Bitratenberechnung muss durch individuelle Patches umgangen werden, wie es z. B. beim `iwlagn`-Treiber notwendig ist (siehe Anhang B.1.2). Für die Punkte 3 und 4 mussten komplett eigene Patches entwickelt werden (siehe Anhang B.1.3 und B.1.4).

```
# Radiotap-Header
0x00:  0x00 0x00 0xc0 0x00     # Vorspann
0x04:  0x06 0x80 0x00 0x00     # Present-Flags
0x08:  0x10                    # Flags
0x09:  ⟨Bitrate⟩               # Rate
0x0a:  0x04 0x00               # TX-Flags

# 802.11-Frame
0x0c:  0x48 0x00               # NULL-Frame, keine Flags
0x0e:  0x00 0x00               # Duration-Wert
0x10:  ⟨DA⟩                    # Destination Address
0x16:  ⟨SA⟩                    # Source Address
0x1c:  ⟨BSSID⟩                 # BSSID
0x22:  0x00 0x00               # Fragment- und Sequenznummer
0x24:  0x00 0x00 0x00 0x00     # Platz fuer FCS
```

Abbildung 4.2: Speicherabbild eines NULL-Frames mit Radiotap-Header

Abbildung 4.2 zeigt das Speicherabbild eines NULL-Frames, wie er in den Kern eingeschleust wird. Im Flags-Feld wird die Berechnung der FCS angefordert, die im eingeschleusten Frame ausgenullt ist. Als Bitrate ist ein Ganzzahlwert einzusetzen, der der Bitrate in MBit/s, multipliziert mit dem Faktor 2, entspricht. Das gesetzte Bit im TX-Flags-Feld fordert ein RTS-CTS-Handshake an. Wird das Handshake nicht benötigt, kann das Feld ausgelassen werden. Die Präsenz-Bits und die Länge des Radiotap-Headers im Vorspann sind dann entsprechend anzupassen. Der Duration-Wert kann bei Null belassen werden. Er wird vom MAC80211-Subsystem berechnet und eingesetzt. Im NULL-Frame sind das „To DS“- und das „From DS“-Flag gelöscht. Die ersten beiden Adressfelder entsprechen somit der MAC-Adresse des Ziels (DA) und der MAC-Adresse des sendenden Interfaces (SA). Im NULL-Frame muss zusätzlich noch das dritte Adressfeld ausgefüllt werden, welches als Zellenidentifikator gilt. In Infrastrukturnetzwerken ist dieses Feld gleich der DA. In Ad-hoc-Netzwerken wird die BSSID durch die erste Station der Zelle nach einem stochastischem Verfahren bestimmt.

Vor dem Einschleusen des Frames muss zunächst ein Monitor-Interface angelegt werden.

```
$ ip link set ⟨Interface⟩ up
$ iw dev ⟨Interface⟩ set channel ⟨Kanal⟩
$ iw dev ⟨Interface⟩ interface add inject_if type monitor
$ ip link set inject_if up
```

Die Übergabe des Frames an das Interface kann durch ein kleines C-Programm erfolgen. Die PCAP-Bibliothek [PCA] eignet sich besonders gut für diese Zwecke. Listing 4.1 zeigt

Beispiel-C-Code für das Einschleusen von `cnt` Frames mittels libpcap über das Interface mit dem Namen `iface`. Der Parameter `frame` zeigt dabei auf den Beginn des Speicherabbilds des `len` Byte langen Frames. Zwischen den einzelnen Frames wird ca. `delay` Mikrosekunden gewartet.

Listing 4.1: Code zum Einschleusen eines 802.11-Frames

```
1  void inject(char *iface, unsigned int cnt, uint8_t *frame, unsigned
       int len, unsigned long delay)
2  {
3    char errbuf[PCAP_ERRBUF_SIZE];
4    pcap_t *pcap;
5    int result;
6    unsigned int i;
7
8    strcpy(errbuf, "");
9    pcap = pcap_open_live(iface, 800, 1, 20, errbuf);
10   if(pcap == NULL)
11     error(-1, 0, "ERROR: %s\n", errbuf);
12   if(strlen(errbuf))
13     fprintf(stderr, "WARNING: %s\n", errbuf);
14
15   for(i = 0; i < cnt; i++)
16   {
17     result = pcap_inject(pcap, frame, len);
18     if(result < 0)
19     {
20       fprintf(stderr, "ERROR: %s\n", pcap_geterr(pcap));
21       break;
22     }
23
24     usleep(delay);
25   }
26
27   pcap_close(pcap);
28 }
```

Für weitere Versuche wurde der obige Code in einem Programm mit der Bezeichnung `nullinject` implementiert (siehe Anhang C.1).

```
$ nullinject
Usage: nullinject
  -i <interface>
 [-n <number of packets>]
 [-b <bssid>]
 [-s <src mac>]
  -d <dst mac>
 [-r <bitrate>]
 [-m <mcs index>]
 [-a <antenna>]
 [-t <delay>]
 [-p]                          # use RTS-CTS
```

```
 [-o]                              # use short preamble
 [-k]                              # don't wait for ACK
 [-x]                              # insert TSF timestamp

Bitrates:
  0 => let interface decide
  1 =>   1   MBit/s         2 =>   2   MBit/s
  3 =>   5.5 MBit/s         4 =>  11   MBit/s
  5 =>   6   MBit/s         6 =>   9   MBit/s
  7 =>  12   MBit/s         8 =>  18   MBit/s
  9 =>  24   MBit/s        10 =>  36   MBit/s
 11 =>  48   MBit/s        12 =>  54   MBit/s
```

Dieses Programm besitzt einige Fähigkeiten, die über die oben diskutierten Erfordernisse hinausgehen, aber sich im späteren Verlauf dieser Arbeit als nützlich erweisen.

- Mit dem `m`-Parameter kann eine 802.11n-Bitrate festgelegt werden.
- Der `a`-Parameter erlaubt, den Index der Sendeantenne festzulegen. Damit sich der Parameter tatsächlich auswirkt, muss der Treiber natürlich diese Funktion unterstützen.
- Mit dem `o`-Parameter kann eine „Short Preamble“ für 802.11b-Frames verwendet werden.
- Der `k`-Parameter weist das MAC80211-Subsystem an, nicht auf einen ACK-Frame für die NULL-Frames zu warten, sondern die „erfolgreiche“ Übertragung des NULL-Frames sofort nach Ende der Aussendung zu bestätigen.
- Mit dem `x`-Parameter wird dem Treiber mitgeteilt, dass in den ersten 8 Bytes der TSF-Zeitstempel zum Sendezeitpunkt eingebettet wird. Dieser Parameter hat nur dann einen Effekt, wenn der WLAN-Treiber eine entsprechende Unterstützung vorsieht.

Die eingangs gezeigte Frame-Sequenz kann mittels `nullinject` wie folgt erzeugt werden.

```
# ohne RTS-CTS-Handshake
$ nullinject -i inject_if -d ⟨DA⟩ -b ⟨BSSID⟩ -r ⟨Bitrate⟩

# mit RTS-CTS-Handshake
$ nullinject -i inject_if -d ⟨DA⟩ -b ⟨BSSID⟩ -r ⟨Bitrate⟩ -p
```

Für die Bitrate ist entsprechend der Hilfeausschrift von `nullinject` ein Index zwischen 0 und 12 zu wählen. Die Quelladresse wird durch `nullinject` automatisch bestimmt.

Nach Abschluss der Messwerterzeugung kann die Monitor-Schnittstelle wieder entfernt werden.

```
$ ip link set inject_if down
$ iw dev inject_if del
```

### 4.1.3 Das Zeitstempelverfahren

Die prinzipiellen Probleme bei der Einbettung von TX-Zeitstempeln in IEEE 802.11 Frames wurden bereits in Abschnitt 3.3.6 diskutiert. Der einzig geeignete Frame ist ein Probe-Response-Frame. Er selbst stellt eine Antwort auf einen Probe-Request-Frame dar, der laut IEEE 802.11 von $B$ stammen muss. Dies könnte dadurch umgangen werden, indem eine dritte Station $C$ einen gefälschten Probe-Request-Frame aussendet, dessen Absendeadresse $B$ anstatt $C$ lautet.

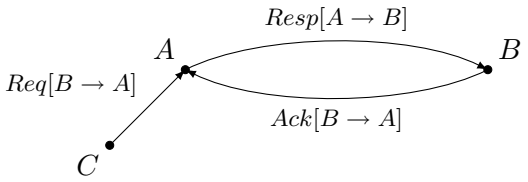



Abbildung 4.3: Einschleusen „gefälschter“ Probe-Requests

Allerdings kann nicht vorhergesagt werden, wie sich die Station $B$ auf ein Probe-Response verhält, ohne dass sie ein Probe-Request ausgesandt hat. Das Problem bleibt auch bestehen, wenn $A$ über die Frame-Injection-Technik direkt das Probe-Response an $B$ verschickt. Zudem müsste $A$ in diesem Fall einen Mechanismus implementieren, der die TSF-Zeitstempel in den Frame einbettet, was durch Frame-Injection nicht vorgesehen ist.

Besteht die Möglichkeit der Zeitstempeleinbettung, kann diese unter Umständen auch auf andere Frame-Typen angewandt werden. Aus diesem Grund wird im folgenden die Kombination des NULL-ACK-Verfahrens mit der Zeitstempelmethode vorgeschlagen. NULL-Frames transportieren keine Datenlast. Bei Einschleusen von Frames ist es indes ohne Weiteres möglich, dem NULL-Frame eine 8 Byte lange Nutzlast anzuhängen. Der Frame wird in diesem Fall auch korrekt übertragen. Gelingt es nun, diese Nutzlast durch einen Zeitstempel zu überschreiben, sind alle Anforderungen des Zeitstempelverfahrens erfüllt. Allerdings verletzt diese TOF-Sequenz den IEEE-802.11-Standard, der keine NULL-Frames mit Nutzlast vorsieht. Der einzige Zweck von NULL-Frames ist die Übermittlung des PM-Bits. Demnach ist die Wahrscheinlichkeit hoch, dass die Nutzlast beim Empfänger ohne weitere Auswertung ignoriert wird. Eine Anwendung dieser Modifikation auf Datenframes scheidet aus, da der Anfang jeder Nutzlast als IEEE 802.3 LLC-Header interpretiert und ein beliebiger Zeitstempel einen ungültigen bzw. unsinnigen LLC-Header darstellt.

Die Einbettung von TX-Zeitstempeln kann beim `iwlagn`-Treiber sehr leicht vorgenommen werden. Bei der Übermittlung des Frames an die Firmware muss lediglich ein Bit in einem Flag-Feld der entsprechenden Kontrollstruktur gesetzt werden (siehe Anhang B.1.5). Um die Frames zu kennzeichnen, die mit einem TX-Zeitstempel versehen werden sollen, wurde der Radiotap-Header um ein entsprechendes Flag „TX-Flags“-Feld erweitert (siehe Anhang B.1.6). Der Vollständigkeit halber soll erwähnt werden, dass dieser Patch zudem die Funktionalität für ein weiteres TX-Flag im Radiotap-Header realisiert, welches das MAC80211-Subsystem anweist, mit der Sendebestätigung nicht bis zum Eintreffen von ACK-Frames zu warten. Tabelle 4.1 fasst alle definierten TX-Flags nochmals zusammen.

Abbildung 4.4 zeigt das Speicherabbild eines modifizierten NULL-Frames für das Zeitstempelverfahren. Nach dem MAC-Header folgen 8 Bytes Nutzlast, die später durch den TSF-

| Flags | Beschreibung |
|---|---|
| 0x0001 | Nach Abschluss der Übertragung wird dem Monitor-Interface ein Bestätigungsframe zugestellt. Ist dieses Bit gesetzt, wurde die Übertragung wegen zu vieler Wiederholungsversuche aufgegeben. |
| 0x0002 | Dieses Flag zeigt an, dass vor einem eingeschleusten Frame ein „CTS-to-self“-Verfahren ausgeführt werden soll. |
| 0x0004 | Dieses Flag zeigt an, dass vor einem eingeschleusten Frame ein RTS-CTS-Handshake ausgeführt werden soll. |
| 0x0008 | Dieses Flag zeigt an, dass das MAC80211-Subsystem keinen ACK-Frame auf den eingeschleusten Frame erwarten soll. |
| 0x0010 | Dieses Flag zeigt an, dass das MAC80211-Subsystem keine Modifikationen am SEQ-Feld im MAC-Header vornehmen soll. |
| 0x0020 | Dieses Flag zeigt an, dass der WLAN-Treiber einen TSF-Zeitstempel in die ersten 8 Bytes der Nutzlast einfügen soll. |

Tabelle 4.1: Die im TX-Flags-Feld definierten Flags

Zeitstempel überschrieben werden. Mittels der TX-Flags wird die Zeitstempeleinbettung angefordert und ferner eine Modifikation des SEQ-Felds verhindert, womit sichergestellt ist, dass die Nummerierung aller NULL-Frames vom Kern nicht verändert wird. Diese Modifikation ist notwendig, um die Frames später unterscheiden zu können. Beim `iwlagn`-Treiber wird der Zeitstempel direkt durch die Firmware in die Nutzlast eingesetzt. Der Kern erhält keine Information über den Wert des Zeitstempels. Es ist demzufolge notwendig, die NULL-Frames an einer anderen Stelle aufzunehmen und die TSF-Zeitstempel zu sichern. Diese Funktion kann z. B. die Station $B$ erfüllen. Um nun die Sendezeitstempel in den NULL-Frames den Emp-

```
# Radiotap-Header
0x00:  0x00 0x00 0xc0 0x00     # Vorspann
0x04:  0x06 0x80 0x00 0x00     # Present-Flags
0x08:  0x10                    # Flags
0x09:  ⟨Bitrate⟩               # Rate
0x0a:  0x38 0x00               # TX-Flags

# 802.11-Frame
0x0c:  0x48 0x00               # NULL-Frame, keine Flags
0x0e:  0x00 0x00               # Duration-Wert
0x10:  ⟨DA⟩                    # Destination Address
0x16:  ⟨SA⟩                    # Source Address
0x1c:  ⟨BSSID⟩                 # BSSID
0x22:  ⟨Nummer⟩                # Fragment- und Sequenznummer
0x24:  0x00 0x00 0x00 0x00     # Platz fuer TSF
       0x00 0x00 0x00 0x00
0x2c:  0x00 0x00 0x00 0x00     # Platz fuer FCS
```

Abbildung 4.4: Speicherabbild eines NULL-Frames für das Zeitstempelverfahren

fangszeitstempeln aus dem Radiotap-Header der ACK-Frames bei $A$ zuordnen zu können, wird das SEQ-Feld als Zuordnungsmerkmal herangezogen.

Das Einschleusen von Frames kann mit dem bereits im letzten Abschnitt beschriebenen Programm `nullinject` erfolgen.

`$ nullinject -i inject_if -d` ⟨*DA*⟩ `-b` ⟨*BSSID*⟩ `-r` ⟨*Bitrate*⟩ `-x -k`

## 4.2 Messwerterfassung

### 4.2.1 Nutzbare Timer

Zur Erfassung von Messwerten werden Uhren benötigt, an denen die Zeitstempel von Sende- und Empfangsereignissen abgelesen werden können. Für diese Zwecke kommen mehrere Zeitquellen in Betracht, die im Verlaufe dieses Abschnitts eingehender betrachtet werden sollen.

**TSF** Auf die Bedeutung der TSF wurde bereits in Abschnitt 2.3.1 eingegangen. Es handelt sich um eine Zeitquelle mit vergleichsweise niedriger Auflösung (1 $\mu$s), die aber bei jedem WLAN-Gerät vorhanden sein muss. Durch die hardwarenahe Realisierung verspricht der TSF-Timer dennoch ein geeigneter Kandidat für die Laufzeitmessung zu sein, da sich Jitter-Effekte weitestgehend ausschließen lassen.

**TSC** Der „time stamp counter“ ist ein prozessorinterner Zähler von Intel Pentium Prozessoren, dessen Zeitbasis sich aus dem Prozessortakt ableitet [Int07]. Aufgrund hoher Taktraten besitzt er eine sehr feine Auflösung. Auf seinen Wert kann mit dem speziellen Maschinenbefehl `rdtsc` zugegriffen werden. Aus diesem Grund ist der Timer nur innerhalb von Treiber-, Betriebssystem- oder Anwendungscode auslesbar. Den frühestmöglichen Zeitpunkt für die Erfassung von TSC-Zeitstempeln stellt somit die durch die WLAN-Hardware aufgerufene Interrupt Service Routine dar.

Der TSC-Timer erfordert im Vorfeld der Messung eine Kalibrierung, da seine Basisfrequenz vom Prozessortakt abhängig ist. Problematisch erweisen sich Stromsparfunktionen, die die Takfrequenz herabsetzen und somit auch die Zeitbasis verändern. Diese Funktion wird allerdings durch das Betriebssystem reguliert, sodass die Möglichkeit besteht, eine erneute Kalibrierung beim Wechsel der Taktfrequenz vorzunehmen [KQ06].

Der TSC ist nur auf Intel-kompatiblen Architekturen vorhanden. Dennoch implementieren auch andere Hardwareplattformen ähnliche Mechanismen. Der Linux-Kern stellt die architekturunabhängige Funktion `cycles_t get_cycles(void)` zum Auslesen solcher Taktzyklenzähler bereit.

**HPET** Die „high precission event timer“ sind prozessorunabhängige Timer, die heute auf vielen PC-Architekturen zu finden sind [Int04]. Im Gegensatz zu TSC-Timern wird ihre Taktfrequenz nicht während der Laufzeit verändert. Dennoch ist die Basisfrequenz von System zu System unterschiedlich. Die Mindesttaktfrequenz beträgt 10 MHz. Die Taktrate kann über ein spezielles Register ausgelesen werden. Eine Kalibrierung entfällt folglich. Der Zugriff auf HPET-Timer kann ebenfalls erst frühestens innerhalb der Interrupt Service Routine erfolgen und verursacht mehr Overhead als das Auslesen des TSC-Zählers, da für HPET-Bausteine ein IO-Zugriff erforderlich ist.

Der Linux-Kern besitzt ein HPET-Subsystem. Ein HPET-Register kann über die Funktion **unsigned int** `hpet_readl(`**unsigned int** `a)` ausgelesen werden, wobei `a` einen Registerindex bezeichnet. Im Register mit dem Index `HPET_PERIOD` ist die Auflösung des Zählers in Femtosekunden hinterlegt, und `HPET_COUNTER` beinhaltet den Zählerwert.

**PCAP-Zeitstempel** Werden Netzwerkpakete mittels libpcap [PCA] abgegriffen, hält die PCAP-Bibliothek für jeden Frame den Empfangszeitpunkt fest. Der genaue Zeitpunkt wird jedoch durch den Anwendungscode ermittelt und besitzt nur eine Auflösung von 1 $\mu$s. Da ein Frame verschiedene Kernsubsysteme (Paket-Filter, Traffic-Shaper, etc.) durchläuft, ist davon auszugehen, dass die Messwerte stark gestreut werden. Ein Gewinn gegenüber dem TSF, der die selbe Auflösung besitzt, kann somit aller Voraussicht nach nicht erzielt werden.

Um diese Zeitquellen zugänglich zu machen, mussten am Linux-Kern einige Erweiterungen vorgenommen werden. Zunächst wurden die Datenstrukturen und das MAC80211-Subsytem um weitere Felder für die entsprechenden Zeitstempel erweitert. Gleichzeitig werden nun die von den WLAN-Treibern zu liefernden Werte für diese Felder in den Sende- und Empfangswegen festgehalten (Anhang B.1.7). Um die Zeitstempel der Anwendung zugänglich zu machen, wurde das Radiotap-Protokoll um die Datenfelder in Tabelle 4.2 erweitert. Diese Erweiterung sah ursprünglich auch den im Interrupt-Handler erfassten Wert des TSF-Zählers vor, der eine Abschätzung der Streuung der TSC- und HPET-Werte vom eigentlichen Empfangszeitpunkt ermöglichen sollte. Es stellte sich jedoch heraus, dass der Zugriff auf den TSF über die Treiber-Funktionen nicht aus dem Tasklet-Kontext möglich ist, in dem die MAC80211-Subroutinen abgearbeitet werden.

| Bit | Größe / Alignm. | Name | Bedeutung |
|---|---|---|---|
| 27 | 8/8 Byte | ISR TSF | Wert des TSF-Zählers bei Eintritt in den Interrupt-Handler. |
| 28 | 8/8 Byte | ISR TSC | Wert des TSC-Zählers bei Eintritt in den Interrupt-Handler. |
| 29 | 8/8 Byte | ISR HPET | Wert des HPET-Zählers bei Eintritt in den Interrupt-Handler. |

Tabelle 4.2: Neu definierte Radiotap-Felder

Die Patches in den Anhängen B.1.8 bis B.1.10 erfassen alle benötigten Zeitstempel in den Interrupt-Handlern und liefern sie an das MAC80211-Subsystem aus. Für den Zugriff auf die HPET-Register musste das Symbol `hpet_readl()` exportiert werden (Anhang B.1.11). Der `iwlagn`-Treiber musste ebenfalls gering modifiziert werden, um den von der Firmware gelieferten TSF-Wert an das MAC80211-Subsystem weiterzugeben (Anhang B.1.12).

### 4.2.2 Methoden zur Messwerterfassung

Möglichkeiten zur Messwerterfassung müssen zwei Aufgaben erfüllen:

- die Feststellung des Zeitpunkts, zu dem ein WLAN-Frame empfangen wurde und
- die Feststellung des Zeitpunkts, zu dem ein WLAN-Frame gesendet wurde.

Trotz der starken Ähnlichkeit dieser Aufgaben gibt es signifikante Unterschiede, die eine getrennte Betrachtung beider Problemfälle notwendig macht. Das Eintreffen von WLAN-Frames geschieht asynchron zu den sonstigen Prozessabläufen beim Empfänger und kann jederzeit der Fall sein.

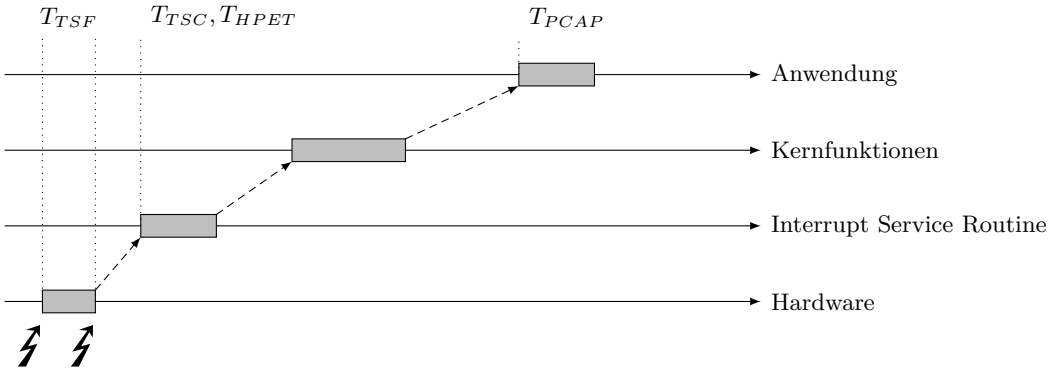



Abbildung 4.5: Ablauf beim Empfangen eines Frames

Abbildung 4.5 zeigt schematisch die Abläufe, welche beim Empfangen von WLAN-Frames über die Monitorschnittstelle stattfinden. Auf Hardwareebene wird firmwareseitig der TSF-Zeitstempel bestimmt. Laut Kernel-Quellen[6] erfüllt der an das MAC80211-Subsystem übergebene Wert folgende Definition

> „Value in microseconds of the 64-bit Time Synchronization Function (TSF) timer when the first data symbol (MPDU) arrived at the hardware."

Allerdings wurde bei den im Rahmen dieser Arbeit durchgeführten Messungen festgestellt, dass manche WLAN-Adapter den TSF-Zähler erst bei Ende der MAC-PDU fixieren, wogegen andere den TSF-Zähler im Anschluss an den PLCP-Header festhalten, was der obigen Definition entspricht. Eine genauere Betrachtung dieser Effekte folgt im nächsten Abschnitt.

Der Kern wird per Interrupt über eingetroffene Frames informiert. Die Interrupt Service Routine kann evt. durch andere Interrupt-Handler, die während des Empfangs aktiv sind, verzögert werden. Bei USB-WLAN-Adaptern ist ferner mit einer zusätzlichen Verzögerung durch das USB-Protokoll zu rechnen. Zu Beginn des Interrupt-Handlers werden die TSC- und HPET-Zeitstempel erfasst. Um die Ausführungszeit des Handlers gering zu halten, wird ein Arbeitsobjekt angelegt und die eigentliche Verarbeitung der Frames an sog. Tasklets delegiert. Aus diesem Takslet heraus wird der Frame in den Anwendungsspeicher kopiert und der Anwendungsprozess deblockiert. Tasklet und Anwendungsprozess können ebenfalls durch Interrupts oder Scheduling-Effekte weiter verzögert werden. Die Erfassung des PCAP-Zeitstempels erfolgt innerhalb des Anwendungscodes.

Für ausgehende WLAN-Frames hat die Anwendung nur einen geringen Einfluss auf den Sendezeitpunkt. Der Beginn der Aussendung kann durch die Zugriffsverfahren auf das Drahtlosmedium sehr lang verzögert werden. Dennoch besteht eine Möglichkeit, den Sendezeitpunkt abzuschätzen. Der WLAN-Adapter signalisiert mittels eines Interrupts, dass die Übertra-

[6] vgl. `http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git;a=blob;f=include/net/mac80211.h`

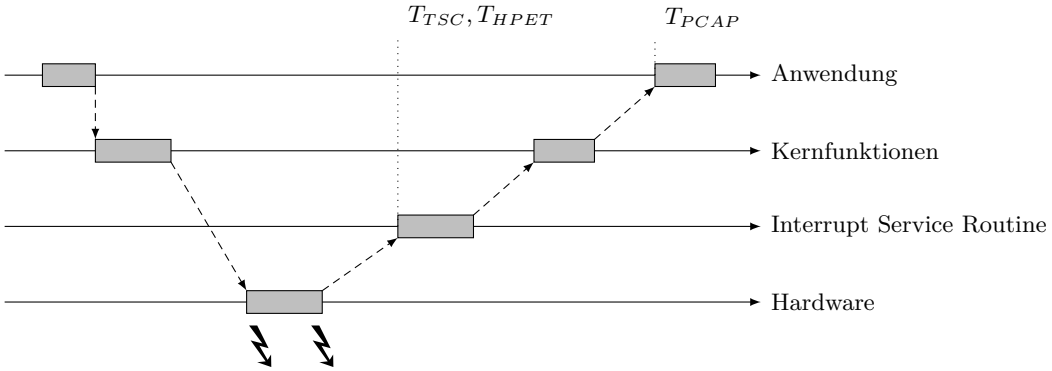



Abbildung 4.6: Ablauf beim Senden eines Frames

gung erfolgt ist. In diesem Interrupt-Handler können die selben Zeitstempel wie im Empfangsfall erfasst werden. Weiterhin löst dieser Interrupt den TX-Response-Mechanismus des MAC80211-Subsystems aus, welcher daraufhin eine Kopie des gesendeten Frames über die Monitor-Schnittstellen an die Anwendung weiterreicht.

Obwohl WLAN-Hardware zur Erzeugung von Beacon-Frames eine Möglichkeit vorsehen muss, den TSF-Zeitstempel für zu sendende Frames festzustellen, wird dieser Wert meist nicht dem WLAN-Treiber zur Verfügung gestellt. Keine der im Rahmen dieser Arbeit zur Verfügung stehenden WLAN-Geräte boten hierzu eine Möglichkeit. In [HW08] wurden weitere Hardware-Produkte untersucht, die die gleiche Einschränkung aufweisen. Ein weiteres Problem wird bei der Betrachtung der Umstände deutlich, unter denen der Bestätigungsinterrupt ausgelöst wird. Die Bestätigung der Übertragung von Frames erfolgt in der Regel erst dann, wenn die Firmware einen entsprechenden ACK-Frame empfangen hat. ACK-Frame und Sendebestätigung wurden meist sogar durch den selben Interrupt an das Betriebssystem weitergereicht. Die Interrupt-Zeitstempel sind folglich ebenfalls ungeeignet für eine Laufzeitmessung.

Will man nicht die Firmware der WLAN-Geräte modifizieren, schafft ein zusätzlicher Monitor-Adapter Abhilfe. Wird dieser in der Nähe der Station $A$ platziert, können die Signallaufzeiten zwischen $A$ und dem Monitor vernachlässigt werden. Jeder Frame, der von $A$ gesendet wird, stellt am Monitor ein Empfangsereignis dar. Der gesamte TOF-Ablauf wurde somit auf eine Reihe von Empfangsereignissen am Monitor reduziert, für die die oben erwähnten Zeitquellen herangezogen werden können. Der TSF-Zähler eines Monitor-Adapters ist ferner unabhängig vom TSF-Zähler in $A$, die Annahmen in Abschnitt 3.2 können also als erfüllt angenommen werden.

Die eigentliche Messwerterfassung kann mit einem beliebigen Paket-Sniffer wie z. B. Wireshark [WS] erfolgen. Zunächst muss wieder ein Monitor-VIF angelegt werden. Der Bezeichner ⟨*Interface*⟩ stehe dabei für ein VIF des Monitor-Adapters.

```
$ ip link set ⟨Interface⟩ down
$ ip link set ⟨Interface⟩ up
$ iw dev ⟨Interface⟩ set channel ⟨Kanal⟩
$ iw dev ⟨Interface⟩ interface add monitor_if type monitor
$ ip link set monitor_if up
```

Anschließend können die im Radiotap-Header kodierten Messwerte mit dem Sniffer erfasst werden. Die gesamten Frames werden dabei hier beispielhaft in der Datei `messung.pcap` aufgezeichnet.

```
tshark -i monitor_if -w messung.pcap
```

Es empfiehlt sich, zusätzlich noch einen Filter festzulegen, um nur wirklich notwendige Pakete zu speichern. Folgende Kommandozeile stellt einen Filter für das RTS-CTS-NULL-ACK-Verfahren bzw. alle auf NULL-Frames basierenden Verfahren dar.

```
$ tshark -i mon -w mon.pcap -f \
  "(wlan addr1 ⟨B⟩ and wlan addr2 ⟨A⟩ and subtype rts) or \
   (wlan addr1 ⟨A⟩ and subtype cts) or \
   (wlan addr1 ⟨B⟩ and wlan addr2 ⟨A⟩ and wlan addr3 ⟨BSS⟩ and \
       subtype null) or \
   (wlan addr1 ⟨A⟩ and subtype ack)"
```

Für $\langle A \rangle$ und $\langle B \rangle$ sind jeweils die MAC-Adressen der Stationen $A$ und $B$ einzusetzen. Der Bezeichner $\langle BSS \rangle$ steht in Ad-hoc-Netzwerken für die zufällig erzeugte ID der Funkzelle. In Infrastrukturnetzwerken entspricht die BSSID der MAC-Adresse des jeweiligen Access Points.

### 4.2.3 Vergleich der Timer

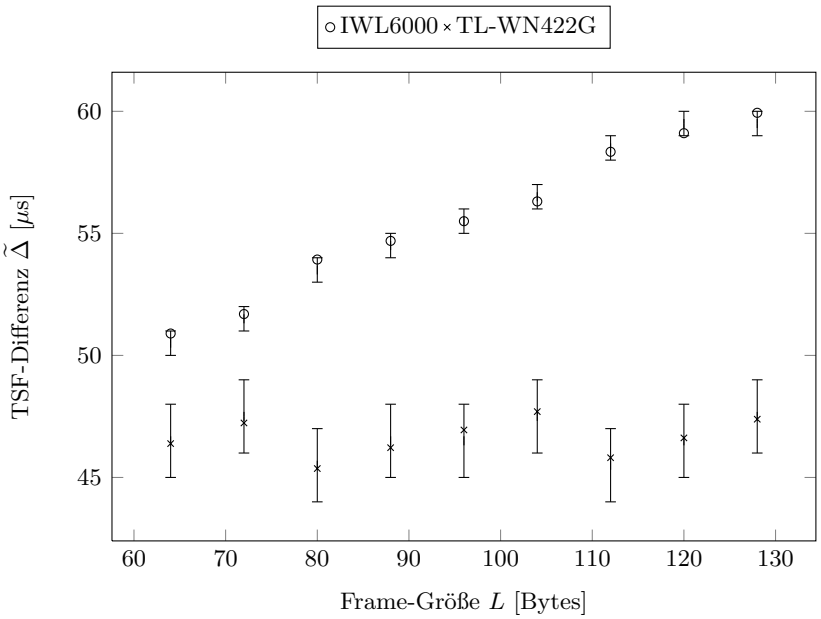



Abbildung 4.7: Bestimmung der Messzeitpunkte durch Messung von $\widetilde{\Delta}$

Zur Bestimmung des genauen Messzeitpunkts des TSF-Zählers wurde ein DATA-ACK-Verfahren durchgeführt. Die Länge der Datenframes wurde durch das Programm `ping` festgelegt. Anschließend wurde die Differenz der TSF-Zeistempel von ACK- und DATA-Frame gebildet. In Diagramm 4.7 sind diese Differenzen für zwei verschiedene WLAN-Adapter aufgetragen.

Die Messungen fanden für 54 MBit/s statt, und es wurden je ca. 1000 Messungen gemittelt. Die Balken stellen die Schwankungsbreite der Messwerte dar. Wird der TSF-Zeitstempel am Ende einer Übertragung erfasst, bleibt die Differenz konstant, da nur die Übertragungsdauer des ACK-Frames in die Differenz eingeht. Im umgekehrten Fall steigt die Übertragungsdauer mit der Länge des Datenframes an. Man erkennt, dass der Atheros-Adapter den TSF-Zeitstempel vermutlich am Ende des Frames fixiert, wogegen der Intel-Adapter den TSF-Zeitstempel zu Beginn des Frames aufnimmt. Nach den Gleichungen (2.20) sollte sich für die Frame-Größen 80, 88, 96 und 102 Bytes die selbe Differenz ergeben, da die Anzahl der OFDM-Symbole in allen Fällen gleich ist. Es erscheint allerdings so, als wäre dieser Differenz noch eine Größe überlagert, die linear vom Füllgrad des letzten OFDM-Symbols abhängt.

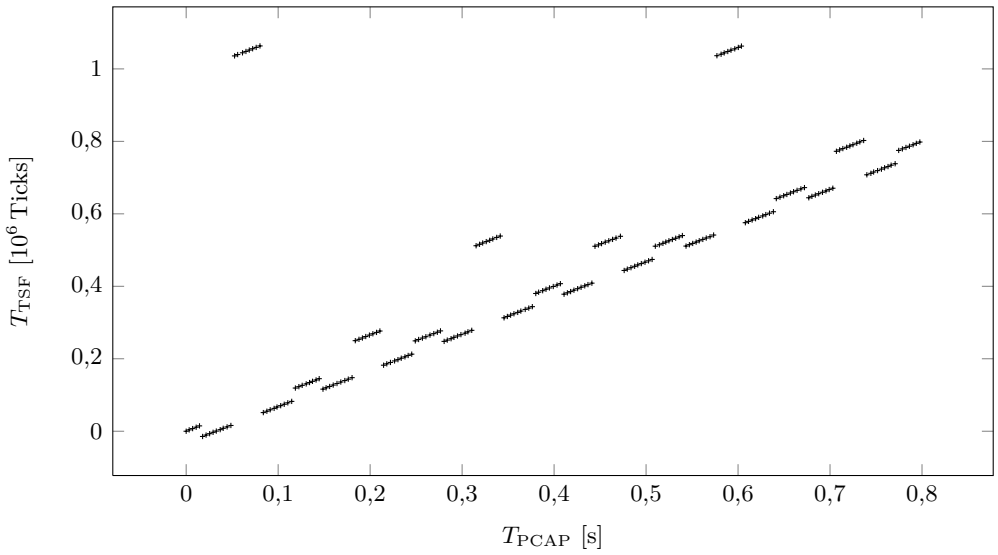



Abbildung 4.8: Atheros-Anomalie

Bei Auswertung der Atheros-TSF-Zeitstempel trat die in Abbildung 4.8 dargestellte Anomalie zu Tage. Jeder Datenpunkt entspricht einem empfangenen Frame, für den die PCAP- und TSF-Zeitstempel gegeneinander abgetragen wurden. Es kann davon ausgegangen werden, dass die PCAP-Uhr konsistent ist. Die TSF-Zeitstempel sind folglich nicht konsistent. In regelmäßigen Abständen treten Sprünge auf, die stets sehr eng an Zweierpotenzen liegen. Die Sprünge sind in jedem Fall mindestens $2^{15}$ TSF-Ticks weit. Vermutlich setzt sich die bitweise Darstellung des TSF-Zeitstempels aus zwei Registern zusammen, die von der Firmware nicht-atomar ausgelesen werden. Bei einer späteren Auswertung der Messdaten müssen demnach ggf. Ausreiser gefiltert werden, die entstehen, wenn innerhalb der TOF-Sequenz ein Sprung auftritt.

Um den durch Verzögerungseffekte hervorgerufenen Jitter der Interrupt-Zeitstempel abschätzen zu können, wurde der Datenverkehr innerhalb einer WLAN-Zelle für eine kurze Zeit aufgezeichnet. Anschließend wurden die Interrupt-Zeitstempel mit den TSF-Zeitstempeln verglichen, welche keinen Jitter aufweisen. Bei ungestörten Interrupt-Zeitstempeln besteht ein linearer Zusammenhang zwischen diesen und den TSF-Zeitstempeln. Der Anstieg dieser linearen Funktion entspricht genau dem Verhältnis der Geschwindigkeiten beider Uhren. Um die Abweichung dieser Zeitstempel von der linearen Funktion besser darstellen zu können, wurden jeweils die Differenzen der Zeitstempel zweier konsekutiver Frames berechnet und

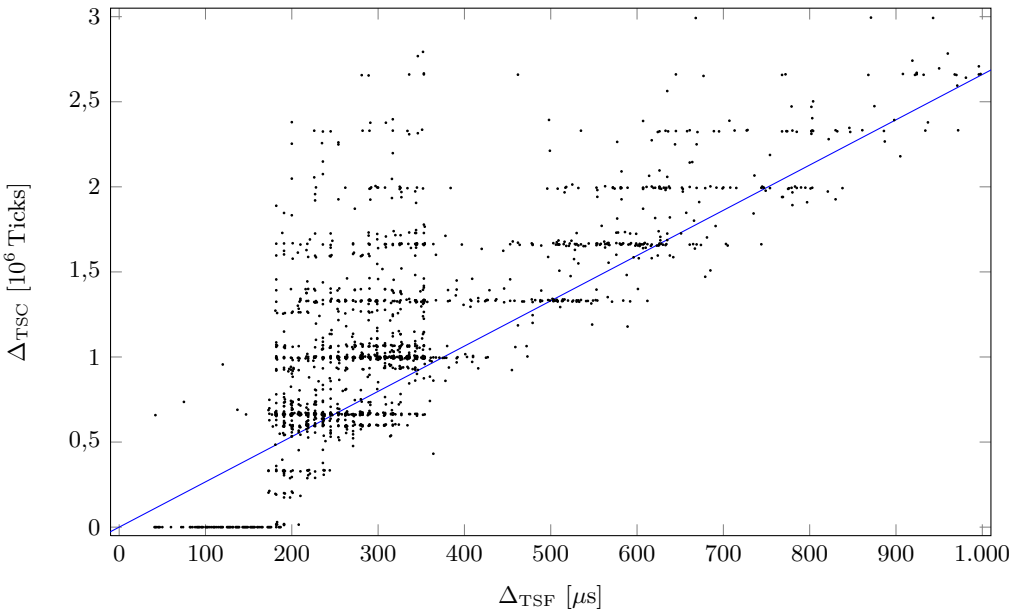



Abbildung 4.9: Vergleich der Timer (TL-WN422G)

im Diagramm 4.9 dargestellt. Die Daten wurden von einem Atheros-Adapter erfasst. Die Datenübertragung fand hauptsächlich mit höheren OFDM-Bitraten statt.

Bei geringem Jitter müssten alle Datenpunkte wiederum auf der Ursprungsgerade liegen, deren Anstieg dem Verhältnis der Geschwindigkeiten entspricht. Bei den kleineren TSF-Differenzen, wie sie bei den TOF-Verfahren auftreten, fällt zunächst auf, dass die TSC-Differenzen hauptsächlich Null sind. Beide Frames der TOF-Sequenz werden also im selben Interrupt-Handler verarbeitet. Die Interrupt-Zeitstempel, und damit auch die PCAP-Zeitstempel, sind demnach für weitere Messungen wertlos. Weiterhin konzentrieren sich die TSC-Differenzen in Abständen von 125 $\mu$s (eine Mikrosekunde entspricht auf dem Referenzsystem ca. 2660 TSC-Ticks). Es kann angenommen werden, dass diese Effekte durch das USB-Protokoll hervorgerufen werden. Das kleinste Abfrageintervall im Interrupt-Modus beträgt für High-Speed-USB-Geräte genau 125 $\mu$s.

Für den IWL6000-Adapter stellt sich die Situation nur wenig besser dar (siehe Abbildung 4.10). Für diese Messreihe wurden die Daten unter vergleichbaren Bedingungen wie bei Abbildung 4.9 erfasst. Bedingt durch den Sampling-Zeitpunkt für die TSF-Zeitstempel beim Intel-Adapter ist $\Delta_{\mathrm{TSF}}$ nicht für jede DATA-ACK-Sequenz konstant. Deshalb wurden zusätzlich eine Reihe von NULL-ACK-Sequenzen erzeugt. Wegen der fehlenden Nutzlast ist für sie $\Delta_{\mathrm{TSF}}$ konstant und entspricht dem linken Ausschlag in Abbildung 4.10. In diesem Bereich kurzer Frame-Zwischenzeiten streuen die TSC-Differenzen erheblich. Es bleibt fraglich, ob sich durch Filterung nutzbare Daten aus den TSC-Zeitstempeln gewinnen lassen. Außerdem treten auch hier Fälle auf, in denen mehrere Frames im selben Interrupt-Handler-Aufruf abgearbeitet werden.

Sowohl für den Atheros- und den Intel-Adapter ergaben sich bei Auswertung der HPET- und PCAP-Zeitstempel ähnliche Diagramme.

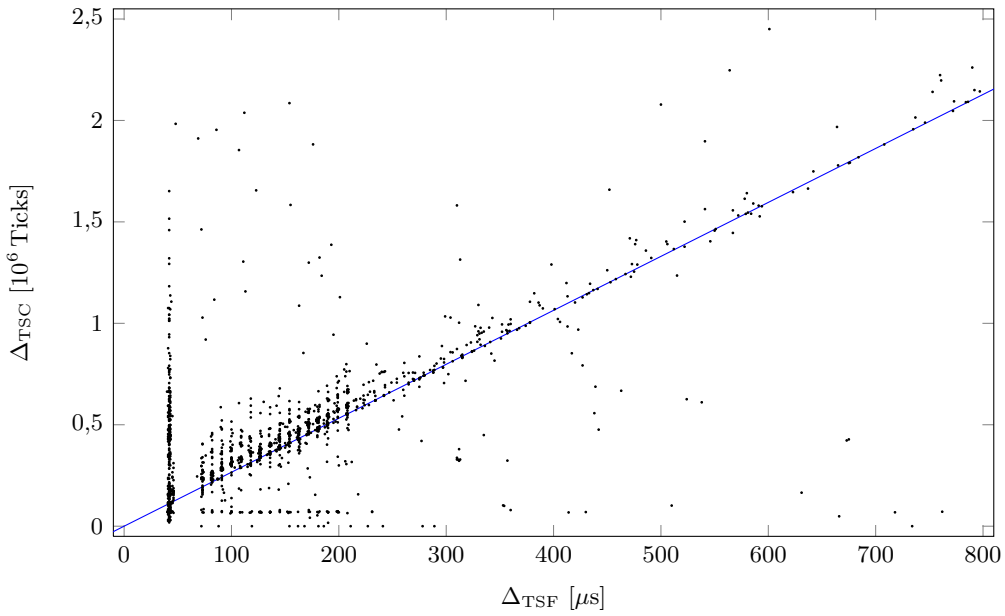



Abbildung 4.10: Vergleich der Timer (IWL6000)

## 4.3 Ein Framework zur Laufzeitmessung

Aufgrund der starken Verzahnung von Messwerterzeugung und -erfassung bietet es sich an, beide Aufgaben in einem gemeinsamen Framework zu implementieren. Es fasst die in den letzten beiden Abschnitten beschriebenen Kommandosequenzen zusammen und führt sie automatisiert aus. Die Implementierung wurde in der Skriptsprache Lua [IFC06] vorgenommen. Lua ist sehr minimalistisch aufgebaut, kann aber trotzdem auf einfache Weise komplexe Datenstrukturen darstellen und ermöglicht sogar die Umsetzung objektorientierter Paradigmen [Ier06]. Ausschlaggebend für die Entscheidung war, dass Lua eine „Klebesprache“ darstellt und sich einfach in andere Sprachen integrieren lässt. Es ist sowohl möglich, die entwickelten Lua-Methoden zur Durchführung der Laufzeitmessung aus C-Code heraus aufzurufen, als auch Low-Level-Funktionen fremder Bibliotheken für das Framework verfügbar zu machen.

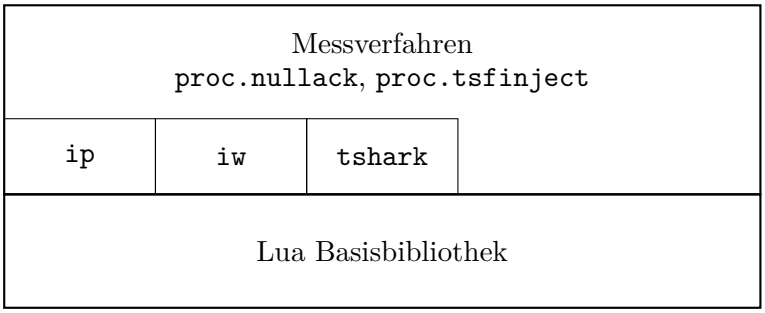



Abbildung 4.11: Struktur des Messframeworks

Abbildung 4.11 zeigt den grundlegenden Aufbau des Frameworks. Die Module `ip` und `iw` schaffen eine Schnittstelle zur Konfiguration der Netzwerkgeräte. Die definierten Methoden werden

in Aufrufe der Programme `ip` und `iw` umgesetzt. Das Modul `tshark` dient der Steuerung des Netzwerksniffers Wireshark. Die einzelnen Methoden dieser Module sind in Anhang D.1 erläutert.

Die Messverfahren werden mittels eines objektorientierten Ansatzes implementiert. Die Klassen `proc.nullack` und `proc.tsfinject` realisieren das (RTS-CTS-)NULL-ACK- und das Zeitstempelverfahren. Für jede Klasse wurden zwei Unterklassen definiert, welche jeweils die Aufgaben des Initiators und des Monitors wahrnehmen (siehe Tabelle 4.3).

| Klasse | Funktion |
|---|---|
| `proc.nullack.monitor` | Erfassung von (RTS-CTS-)NULL-ACK-Verfahren |
| `proc.nullack.initiator` | Erzeugung von (RTS-CTS-)NULL-ACK-Verfahren |
| `proc.nullack` | Erzeugung und Erfassung von (RTS-CTS-)NULL-ACK-Verfahren |
| `proc.tsfinject.monitor` | Erfassung von Zeitstempelverfahren |
| `proc.tsfinject.initiator` | Erzeugung von Zeitstempelverfahren |
| `proc.tsfinject` | Erzeugung und Erfassung von Zeistempelverfahren |

Tabelle 4.3: Klassen für die Abwicklung der Messverfahren

Für diese Klassen sind folgende Methoden definiert:

**`proc.nullack.monitor:prepare(`**⟨monifs⟩**`,` ⟨chan⟩`)`**

erzeugt ein Monitorobjekt. Die Methode stellt folglich den Konstruktor dar. Von der Funktion wird ein Handler auf das Objekt zurückgegeben. Der Parameter ⟨monifs⟩ muss ein Array von Strings sein, welches alle WLAN-Schnittstellen angibt, über die Messwerte erfasst werden sollen. Die Kanalnummer wird durch ⟨chan⟩ angegeben.

**`proc.nullack.monitor:start(`**⟨prefix⟩**`,` ⟨src⟩**`,` ⟨dst⟩**`,` ⟨bssid⟩**`)`**

startet die Messwerterfassung für das Monitorobjekt. Mit den Parametern ⟨src⟩, ⟨dst⟩ und ⟨bssid⟩ wird der Paketfilter für die Messwerterfassung festgelegt. Alle Messwerte werden in einer Datei gespeichert, deren Name sich wie folgt zusammensetzt:

⟨prefix⟩`-`⟨monif⟩`-`⟨dst⟩`-`⟨Zeitstempel⟩`.pcap`

Für jedes Interface wird eine eigene Datei angelegt, wobei ⟨monif⟩ durch dessen Bezeichner ersetzt wird. Das Präfix ⟨prefix⟩ kann eine beliebige Verzeichnisangabe enthalten. Für ⟨Zeitstempel⟩ wird der UNIX-Zeitstempel zum Beginn der Messung eingesetzt. Die Funktion kehrt nach Beginn der Messung zurück.

**`proc.nullack.monitor:stop()`**

beendet alle laufenden Messinstanzen.

**`proc.nullack.monitor:cleanup()`**

gibt das Messobjekt frei. Alle Monitorschnittstellen werden heruntergefahren.

**`proc.nullack.initiator:prepare(`**⟨fromif⟩**`,` ⟨chan⟩**`)`**

ist der Konstruktor für ein Initiatorobjekt. Der Parameter ⟨fromif⟩ gibt das WLAN-Interface an, über welche die NULL-ACK-Sequenzen erzeugt werden sollen. Die Kanalnummer wird durch ⟨chan⟩ festgelegt.

`proc.nullack.initiator:initiate(`⟨dst⟩`,` ⟨bssid⟩`,` ⟨n⟩`,` ⟨rate⟩`,` ⟨dt⟩`,` ⟨rts⟩`,` ⟨ant⟩`)`

löst NULL-ACK-Sequenzen aus. Die Parameter ⟨dst⟩ und ⟨bssid⟩ geben die MAC-Adresse der Station $B$ und die Bezeichnung der Funkzelle an. Es werden insgesamt ⟨n⟩ Sequenzen initiiert. Ist der Parameter ⟨rate⟩ verschieden von `nil`, werden die NULL-Frames mit der festgelegten Bitrate übertragen. Ist der Parameter ⟨dt⟩ gegeben, wird er als eine Zeitangabe in Mikrosekunden interpretiert, die zwischen den einzelnen NULL-ACK-Sequenzen zu warten ist. Wird ⟨rts⟩ als wahr ausgewertet, wird ein RTS-CTS-NULL-ACK-Verfahren ausgeführt. Ist ⟨ant⟩ gegeben, wird der Wert als Index der Sendeantenne an das Programm `nullinject` übergeben. Die Funktion kehrt nach Beendigung des Programms `nullinject` zurück.

`proc.nullack.initiator:cleanup()`

gibt das Initiatorobjekt frei. Das vom Konstruktor angelegte Monitor-VIF wird gelöscht.

`proc.nullack:prepare(`⟨fromif⟩`,` ⟨monifs⟩`,` ⟨chan⟩`)`

erzeugt ein Messobjekt, welches Erfassung und Erzeugung des NULL-ACK-Verfahrens in einem Schritt vornimmt. Die Parameter werden an die Konstruktoren der Klassen `proc.nullack.monitor` und `proc.nullack.initiator` durchgereicht.

`proc.nullack:measure(`⟨prefix⟩`,` ⟨dst⟩`,` ⟨bssid⟩`,` ⟨n⟩`,` ⟨rate⟩`,` ⟨dt⟩`,` ⟨rts⟩`,` ⟨ant⟩`)`

nimmt die Erzeugung und Erfassung von NULL-ACK-Sequenzen vor. Die Parameter werden an die Funktionen `proc.nullack.monitor:start` und `proc.nullack.initiator:initiate` weitergegeben. Zuerst wird die Messwerterfassung gestartet. Anschließend folgt die Erzeugung der NULL-ACK-Sequenzen. Vor dem Beenden der Messung wird für eine von ⟨n⟩, ⟨rate⟩, ⟨dt⟩ und ⟨rts⟩ abhängige Zeit gewartet, um sicherzustellen, dass alle NULL-ACK-Sequenzen erfasst werden.

`proc.nullack:cleanup()`

ruft die Destruktoren des Monitor- und Initiatorobjekts auf.

Für das Zeitstempelverfahren sind gleichlautende Methoden implementiert. Im Folgenden wird lediglich auf die Unterschiede zu den `proc.nullack`-Klassen eingegangen.

- An die Stelle des Parameters ⟨monifs⟩ tritt der Parameter ⟨monif⟩. Er stellt kein Array von Bezeichnern dar, sondern entspricht direkt einer Bezeichnung für ein Monitor-Interface.
- Die Methode `proc.tsfinject.initiator:initiate` muss ebenfalls Pakete aufzeichnen und erhält deshalb einen ⟨prefix⟩-Parameter. Um die Mitschnitte der Initiator-Methode von den Mitschnitten der Monitor-Methode unterscheiden zu können, werden die Dateinamen nach folgendem Muster gebildet:
  - ⟨prefix⟩`-mon-`⟨monif⟩`-`⟨dst⟩`-`⟨Zeitstempel⟩`.pcap` für den Monitor
  - ⟨prefix⟩`-ini-`⟨fromif⟩`-`⟨dst⟩`-`⟨Zeitstempel⟩`.pcap` für den Initiator
- Ein RTS-CTS-Handshake ist für das Zeitstempelverfahren überflüssig. Der ⟨rts⟩-Parameter entfällt damit.
- Nach diesen Modifikationen ergeben sich somit folgende neue Signaturen:

- **proc.tsfinject.initiator:initiate(**⟨prefix⟩**,** ⟨dst⟩**,** ⟨bssid⟩**,** ⟨n⟩**,** ⟨rate⟩**,** ⟨dt⟩**,** ⟨ant⟩**)**
- **proc.tsfinject:measure(**⟨prefix⟩**,** ⟨dst⟩**,** ⟨bssid⟩**,** ⟨n⟩**,** ⟨rate⟩**,** ⟨dt⟩**,** ⟨ant⟩**)**

Die Nutzung des Frameworks erfolgt durch den Aufruf der einzelnen Funktionen über den interaktiven Lua-Interpreter. Für die Ausführung des Frameworks sind Root-Rechte notwendig, da Monitor-Schnittstellen angelegt werden müssen und der WLAN-Sniffer Zugriff auf einen RAW-Socket erfordert. Am besten wird die Benutzung des Frameworks durch Betrachten einer beispielhaften Laufzeitmessung deutlich. Im Beispielszenario verfügt die Station $A$, die ihren Abstand zur Station $B$ messen will, über drei WLAN-Schnittstellen. Die Schnittstellen `wlan0` und `wlan1` sollen die Monitor-Funktion erfüllen. Über die Schnittstelle `wlan2` wird ein RTS-CTS-NULL-ACK-Verfahren initiiert. Vor Laden des Frameworks ist ggf. noch die BSSID der Funkzelle zu bestimmen.

```
$ iw wlan2 scan
BSS 0a:af:33:2c:84:0c (on wlan2)
        TSF: 1109035735 usec (0d, 00:18:29)
        freq: 2472
        beacon interval: 100
        capability: IBSS (0x0002)
        signal: -35.00 dBm
        last seen: 1424 ms ago
        SSID: testwlan
        Supported rates: 1.0* 2.0* 5.5 11.0 6.0 9.0 12.0 18.0
        DS Parameter set: channel 13
        Extended supported rates: 24.0 36.0 48.0 54.0
```

Im Beispiel ist auf Kanal 13 die Ad-hoc-Funkzelle `testwlan` mit der BSSID `0a:af:33:2c:84:0c` aktiv. Die MAC-Adresse der $B$-Station lautet im Beispiel `00:1e:e5:e2:b7:80`. Liegen alle MAC-Adressen vor, kann das Framework geladen werden

```
$ cd messtool
$ lua -i tool.lua
Lua 5.1.4  Copyright (C) 1994-2008 Lua.org, PUC-Rio
>
```

Vor der Messung muss ein Messobjekt angelegt werden. Dazu sind die Interface-Bezeichner und die Kanalnummer an den Konstruktor zu übergeben.

```
> mo = proc.nullack:prepare("wlan2", { "wlan0", "wlan1" }, 13)
```

Durch diesen Aufruf werden die benötigten Monitor-VIFs erzeugt. Der Messmethoden werden die oben aufgeführten Parameter übergeben, die anschließend das Programm `nullinject` aufruft.

```
> mo:measure("/tmp/messung", "00:1e:e5:e2:b7:80",
   "0a:af:33:2c:84:0c", 10000, 54, 1000, true)
10000 packets 00:c0:ca:35:1b:e2 -> 00:1e:e5:e2:b7:80 via na_ini
   (bssid  0a:af:33:2c:84:0c)
10000 packets transmitted
```

Listing 4.2: Automatisierte Erfassung einer Messreihe mit dem Messframework

```
local ms =
{
  { rate = "54", k = 5, n = 10000, delta = "1000", rts = true },
  { rate = "11", k = 5, n = 10000, delta = "1000", rts = true },
  { rate =  "2", k = 5, n = 10000, delta = "1000", rts = true },
}

for _, m in ipairs(ms) do
  for i = 1, m.k do
    local myprefix

    myprefix = string.format("/tmp/messreihe-%smbps-%sus-%d", m.
      rate,
                             m.delta, i)
    mo:measure(myprefix, dst, bssid, m.n, m.rate, m.delta, m.rts)
  end
end
```

Nach der Messung kann der Destruktor aufgerufen werden. Das Framework wird durch Verlassen des Interpreters beendet werden.

```
> mo:cleanup()
> ^D
$
```

Die erfassten TOF-Sequenzen wurden durch das Framework in folgenden beiden Dateien im `/tmp`-Verzeichnis abgelegt, wo sie bis zur anschließenden Auswertung verbleiben können.

```
/tmp/messung-wlan0-00:1e:e5:e2:b7:80-20110219162235.pcap
/tmp/messung-wlan4-00:1e:e5:e2:b7:80-20110219162235.pcap
```

Für die automatisierte Durchführung einer Messreihe sind keine Modifikationen am Framework notwendig. Der Lua-Intereter stellt bereits alle notwendigen Bausteine wie Schleifen und komplexe Datenstrukturen bereit. Das Lua-Skript in Listing 4.2 führt eine Messreihe für die Bitraten 2, 11 und 54 MBit/s durch. Für jede Bitrate werden fünf Messungen zu je 10000 RTS-CTS-NULL-ACK-Sequenzen vorgenommen.

## 4.4 Auswertung der Messwerte

Die Messwerte liegen nach der Erfassung als eine durch Wireshark aufgezeichnete Folge von WLAN-Frames vor. Zur Anwendung der in Abschnitt 3.2 beschriebenen Modelle müssen TOF-Sequenzen in diesem Frame-Strom identifiziert und Zeitstempeldifferenzen berechnet werden. Die Extraktion der Werte aus dem Mitschnitt kann ebenfalls mittels Wireshark erfolgen. Durch den folgenden einfachen Kommandozeilenaufruf können die mit der `e`-Option angegebenen Datenfelder aus den Frames in eine Ausgabe im CSV-Format konvertiert werden.

```
$ tshark -r mon.pcap -T fields -E header=n -E separator=, \
  -e frame.number            -e frame.len             \
  -e radiotap.mactime        -e radiotap.length       \
  -e radiotap.datarate       -e radiotap.channel      \
  -e radiotap.flags          -e wlan.fc.type_subtype  \
  -e wlan.sa                 -e wlan.da               \
  -e wlan.ta                 -e wlan.bssid            \
  ...
```

Die Liste aller definierten Felder ist Bestandteil der Wireshark-Dokumentation.[7] Der Radiotap-Protokolldekoder von Wireshark unterstützt nicht alle definierten Datenfelder. Die in Abschnitt 4.2.1 neu definierten Radiotap-Felder mussten ebenfalls nachgerüstet werden. Der entsprechende Patch, der diese beiden Modifikationen vornimmt, kann in Anhang B.2.1 nachgeschlagen werden. Tabelle 4.4 führt die Zuordnung zwischen den in dieser Arbeit eingeführten Radiotap-Feldern und deren Wireshark-Datenfeldbezeichnern auf. Zudem wurden die Bits der Präsenzmaske ebenfalls durch eigene Wireshark-Datenfelder zugänglich gemacht.

| Radiotap-Feld | Wireshark-Feld | Present-Flag |
|---|---|---|
| `ISR TSF` | `radiotap.isrtsf` | `radiotap.present.isrtsf` |
| `ISR TSC` | `radiotap.isrtsc` | `radiotap.present.isrtsc` |
| `ISR HPET` | `radiotap.isrhpet` | `radiotap.present.isrhpet` |

Tabelle 4.4: Wireshark-Feldbezeichner für die neu definierten Radiotap-Felder

Für das Zeitstempelverfahren wurden die TSF-Zeitstempel als Nutzlast an einen NULL-Frame angehängt. Da NULL-Frames laut IEEE 802.11 keine Nutzlast enthalten, bietet Wireshark folglich auch keine Möglichkeit, diese Daten zu dekodieren. Die in Anhang B.2.2 beschriebene Modifikation ruft den Data-Dissector von Wireshark für diese Nutzlast auf. Dieses Modul macht den Zeitstempel über das `data`-Feld als Hexadezimalcode verfügbar.

Zur Auswertung eines NULL-ACK-Verfahren müssen zunächst alle NULL-ACK-Sequenzen im Frame-Strom ausfindig gemacht werden. Eine Frame-Folge $(f_1, f_2)$ ist genau dann eine NULL-ACK-Sequenz, wenn

- die Frames unmittelbar aufeinanderfolgend aufgenommen wurden,
- $f_1$ ein NULL-Frame ist,
- $f_2$ ein ACK-Frame ist,
- die Transmitting Addr. des NULL-Frames mit der MAC-Adresse von $A$ übereinstimmt,
- die Receiving Addr. des NULL-Frames mit der MAC-Adresse von $B$ übereinstimmt und
- die Receiving Addr. des ACK-Frames mit der MAC-Adresse von $A$ übereinstimmt.

Für das RTS-CTS-NULL-ACK-Verfahren müssen die obigen Bedingungen um die vorgelagerten RTS- und CTS-Frames ergänzt werden. Für die Adressen des RTS-Frames gelten die selben Regelungen wie für die Adressen des NULL-Frames. Gleiches trifft auf CTS- und ACK-Frames zu.

Algorithmus 4.1 beschreibt die Auswertung der NULL-ACK-Messwerte formal. Es wird ein Fenster der Länge 2 über den Frame-Strom geschoben und geprüft, ob dieses Fenster über

[7] vgl. `http://www.wireshark.org/docs/dfref/`

**Algorithmus 4.1** Auswertung der $\widetilde{\Delta}_i$ aus dem NULL-ACK-Verfahren

**procedure** ANALYZE($(f_i), A, B$) ▷ $(f_i)$ ... Folge von Frames
▷ $A, B$ ... Adressen von $A$ und $B$

$L \leftarrow ()$ ▷ Liste aller gemessenen Laufzeitdifferenzen
**for** $i = 1, \ldots, \text{len}(f) - 1$ **do**
**if** $(f_i, f_{i+1})$ ist eine gültige NULL-ACK-Sequenz zwischen $A$ und $B$ **then**
$\widetilde{\Delta} \leftarrow \text{rxtime}(f_{i+1}) - \text{rxtime}(f_i)$
Füge $\widetilde{\Delta}$ zu $L$ hinzu.
**end if**
**end for**
Bereinige $L$ von vermutlichen Messfehlern.
Bestimme $\widehat{\Delta}$ als arithmetisches Mittel über $L$.
Bestimme die Entfernung durch den Vergleich von $\widehat{\Delta}$ mit den Resultaten einer Referenzmessung.
**end procedure**

einer gültigen NULL-ACK-Sequenz liegt. Ist dies der Fall, wird die Differenz der Empfangszeitstempel berechnet und gespeichert. Aus den in Abschnitt 4.2.3 genannten Gründen wird eine Bereinigung der Messwerte von Messfehlern notwendig sein. Auf ein mögliches Filterverfahren wird weiter unten in diesem Abschnitt eingegangen.

Die Auswertung eines Zeitstempelverfahrens stellt sich komplexer dar und ist in Algorithmus 4.2 dargestellt. Das Problem besteht darin, dass sich die TSF-Zeitstempel auf zwei Frame-Ströme verteilen. Im Frame-Strom des Initiators muss mittels der Fenstertechnik zunächst eine NULL-ACK-Sequenz gefunden werden. Anschließend muss im Monitor-Strom der selbe NULL-Frame identifiziert werden. Um sicher zu gehen, dass es sich wirklich um die selben NULL-Frames handelt, werden die Sequenzfelder der MAC-Header miteinander verglichen. Wurde festgestellt, dass die NULL-Frames identisch sind, kann die Differenz zwischen dem TX-Zeistempel des am Monitor erfassten NULL-Frames und dem RX-Zeitstempel des ACK-Frames beim Initiator berechnet werden. Eine Filterung des Messwerte wie beim NULL-ACK-Verfahren bietet sich an.

Ein mögliches Filterverfahren stellt das $\kappa$-$\sigma$-Clipping dar. Die Bezeichner $\kappa$ und $\sigma$ sind zugleich die Parameter des Filters. Aus einer Folge von Messwerten werden diejenigen Realisierungen entfernt, deren Wert nicht in einem bestimmten Intervall liegt. Die Intervallgrenzen hängen dabei vom empirischen Mittelwert $m$ und der empirischen Varianz $v$ der Messreihe ab. Das Intervall berechnet sich anhand der Filterparameter und der empirischen Kenngrößen durch $[\, m - \sigma \cdot \sqrt{v}, m + \sigma \cdot \sqrt{v}\,]$. Dieser Schritt wird bis zu $\kappa$ mal wiederholt. Das $\kappa$-$\sigma$-Clipping ist ein adaptives Verfahren, benötigt also kein weiteres Wissen über die Verteilung der Messwerte. Je kleiner der Parameter $\sigma$ ist, um so aggressiver beschneidet das Verfahren die Messwerte.

Für Zweipunktverteilungen, wie sie laut Abschnitt 3.2 zu erwarten sind, besitzt das Verfahren allerdings eine Schwäche. Konzentrieren sich die Messwerte hauptsächlich auf eine der beiden möglichen Ausprägungen, wird die Varianz sehr klein. Es besteht die Gefahr, dass das Clipping-Intervall zu eng wird und alle anderen Ausprägungen verwirft. Aus diesem Grund wird hier das $\kappa$-$\sigma$-Verfahren um einen dritten Parameter $\mu$ erweitert, der die Mindestbreite jeder Intervallhälfte festlegt (siehe Algorithmus 4.3). Für $\mu$ sollte die Differenz zwischen den beiden zu erwartenden Ausprägungswerten eingesetzt werden. Es wird damit sichergestellt, dass auch bei aggressivem $\sigma$ die Zweipunktverteilung in jedem Fall erhalten bleibt.

**Algorithmus 4.2** Auswertung der $\widetilde{\Delta}_i$ aus dem Zeitstempelverfahren

**procedure** ANALYZE($(f_i^I), (f_i^M), A, B$) ▷ $(f_i^I)$ ... Folge von Frames am Initiator-Interface
▷ $(f_i^M)$ ... Folge von Frames am Monitor-Interface
▷ $A, B$ ... Adressen von $A$ und $B$

$j \leftarrow 1$
$L \leftarrow ()$ ▷ Liste aller gemessenen Laufzeitdifferenzen
**for** $i = 1, \ldots, \text{len}(f^I) - 1$ **do**
**if** $(f_i^I, f_{i+1}^I)$ ist keine gültige NULL-ACK-Sequenz zwischen $A$ und $B$ **then**
**continue**
**end if**
▷ Den Monitor-Frame zu $f_i^I$ suchen, der den TX-Zeistempel enthält.
**while** $j < \text{len}(f^M) \wedge \text{txmactime}(f_j^M) < \text{rxmactime}(f_{i+1}^I)$ **do**
$j \leftarrow j + 1$
**end while**
$j \leftarrow j - 1$
▷ $f_i^I$ und $f_j^M$ sind vermutlich der selbe Frame.
**if** $\text{seq}(f_i^I) \neq \text{seq}(f_j^M)$ **then**
**continue**
**end if**
▷ $f_i^I$ und $f_j^M$ sind der selbe Frame.
$\widetilde{\Delta} \leftarrow \text{txmactime}(f_j^M) - \text{rxmactime}(f_{i+1}^I)$
Füge $\widetilde{\Delta}$ zu $L$ hinzu.
**end for**
Bereinige $L$ von vermutlichen Messfehlern.
Bestimme $\widehat{\Delta}$ als arithmetisches Mittel über $L$.
Bestimme die Entfernung durch den Vergleich von $\widehat{\Delta}$ mit den Resultaten einer Referenzmessung.
**end procedure**

**Algorithmus 4.3** Modifiziertes $\kappa$-$\sigma$-Clipping

**function** $\kappa$-$\sigma$-CLIPPING($(x_i), \kappa, \sigma, \mu$)
$L \leftarrow (x_i)$
**for** $i = 1, \ldots, \kappa$ **do**
$L' \leftarrow ()$
$m \leftarrow$ empirischer Mittelwert über $L$
$v \leftarrow$ empirische Varianz über $L$
$d \leftarrow \sigma \cdot \sqrt{var}$
**if** $d < \mu$ **then**
$d \leftarrow \mu$
**end if**
**for** $x \in L$ **do**
**if** $x \in [\, m - d,\, m + d]$ **then**
Füge $x$ zu $L'$ hinzu.
**end if**
**end for**
$L \leftarrow L'$
**end for**
**return** $L$
**end function**

## 4.5 Ein Framework zur Messwertauswertung

Das Auswerteframework ist in der Handhabung ähnlich zum Messframework. Es besteht aus einer Menge von Lua-Funktionen, die über einen interaktiven Interpreter aufgerufen werden können. Die Funktionen sind in Anhang D.2 beschrieben und gliedern sich in vier Module.

**import** definiert Funktionen zum Umwandeln von PCAP- in CSV-Dateien und deren anschließenden Import in Lua-Datenstrukturen.

**frame** wertet die durch das Importmodul geladenen Frames aus.

**list** vereinfacht den Zugriff auf Datenstrukturen, die die diversen TOF-Analysatoren liefern.

**util** stellt eine Reihe von Statistikfunktionen zur eigentlichen Auswertung der Messreihen bereit.

Die TOF-Analysatoren haben die Aufgabe, einen Frame-Strom nach TOF-Sequenzen abzusuchen, ggf. Laufzeitdifferenzen zu berechnen und diese zurückzugeben. Die Messwerte werden dabei in einer speziell strukturierten Lua-Datenstruktur an den Aufrufer zurückgegeben. Da ein TOF-Ablauf mit verschiedenen Uhren gemessen werden kann, müssen für jede Sequenz mehrere Messwerte speicherbar sein. Weiterhin kann es hilfreich sein, jede Sequenz später wieder einem Frame zuordnen zu können. Dies ist unter anderem dann erwünscht, wenn die Laufzeitmessungen mit anderen Informationen wie z. B. Empfangsantennen, Empfangsfeldstärken etc. korreliert werden sollen. Die in Abbildung 4.12 dargestellte Datenstruktur leistet diese Fähigkeiten.

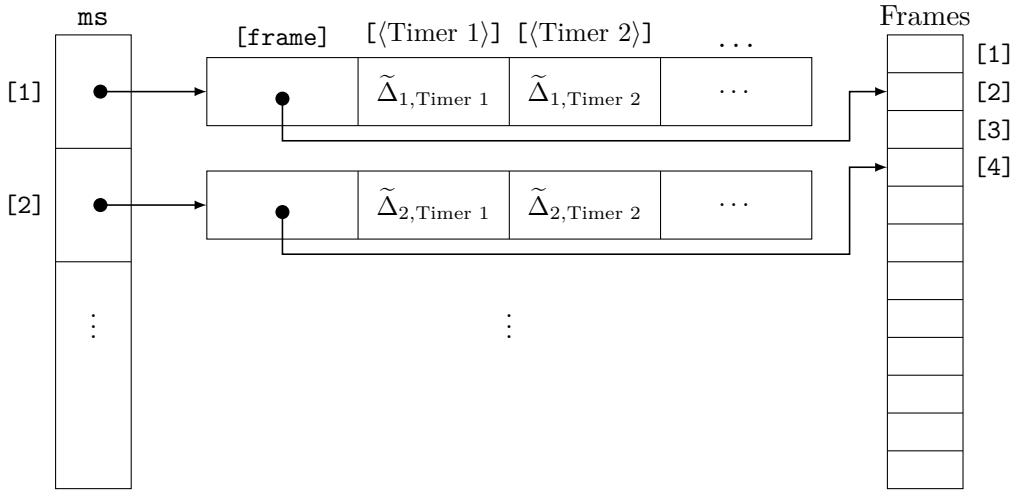



Abbildung 4.12: Analysator-Rückgabedatenstruktur

Die Messwerte werden in Form einer Lua-Tabelle `ms` zurückgegeben, die hier als Array interpretiert kann. Für eine detaillierte Beschreibung, wie komplexe Datenstrukturen durch Lua-Tabellen dargestellt werden können, sei auf Abschnitt 2.5.7 von [IFC06] oder Kapitel 11 von [Ier06] verwiesen. Jedes Element des Rückgabearrays stellt einen Verweis auf eine weitere Lua-Tabelle dar, die einen Verbunddatentyp realisiert. Das Element `frame` dieses Verbunds ist ein Verweis auf den ersten Frame der TOF-Sequenz. Für jeden Timer, der zur Auswertung

herangezogen wurde, existiert ein weiteres Feld in diesem Verbund, das den numerischen Wert der Zeitstempeldifferenz speichert.

Für die Zwecke dieser Arbeit wurden fünf Analysatoren implementiert.

**`analyze.dataack(`**⟨fs⟩**`,`** ⟨fields⟩**`,`** ⟨amac⟩**`,`** ⟨bmac⟩**`,`** ⟨rate⟩**`,`** ⟨len⟩**`)`**

untersucht den Frame-Strom ⟨fs⟩ auf DATA-ACK-Sequenzen zwischen den Stationen mit den MAC-Adressen ⟨amac⟩ und ⟨bmac⟩. Der Datenframe muss mit der Bitrate ⟨rate⟩ übertragen worden sein und inklusive MAC-Header eine Länge von ⟨len⟩ Bytes haben. Im Array ⟨fields⟩ werden die Bezeichner der Timer angegeben, die durch die Analyseroutine ausgewertet werden sollen. Tabelle 4.5 listet die Bezeichner aller implementierten Timer auf. Wird nur der Parameter ⟨fs⟩ an die Funktion übergeben, gibt die Funktion eine Übersicht aller möglichen Belegungen von ⟨amac⟩, ⟨bmac⟩, ⟨rate⟩ und ⟨len⟩ aus, für die TOF-Sequenzen in ⟨fs⟩ existieren.

Der DATA-ACK-Analysator wird nicht zur Laufzeitmessungen benötigt. Er diente lediglich zur Berechnung der Daten für das Diagramm in Abbildung 4.7.

**`analyze.nullack(`**⟨fs⟩**`,`** ⟨fields⟩**`,`** ⟨amac⟩**`,`** ⟨bmac⟩**`,`** ⟨rate⟩**`)`**

wertet analog zum DATA-ACK-Analysator NULL-ACK-Sequenzen in ⟨fs⟩ aus. Die Bedeutung der Parameter unterscheidet sich nicht. Lediglich der ⟨len⟩-Parameter wird nicht benötigt, da NULL-Frames keine Nutzlast transportieren. Auch der NULL-ACK-Analysator gibt einer Übersicht über die Wahlmöglichkeiten der restlichen Parameter aus, wenn nur der Frame-Strom übergeben wird.

**`analyze.rtsctsnullack(`**⟨fs⟩**`,`** ⟨fields⟩**`,`** ⟨amac⟩**`,`** ⟨bmac⟩**`,`** ⟨rtsrate⟩**`,`** ⟨nullrate⟩**`)`**

wertet RTS-CTS-NULL-ACK-Sequenzen aus. Die Datenrate für die RTS-Frames und die NULL-Frames können getrennt voneinander spezifiziert werden. Sonst arbeitet die Analyseroutine genau wie `analyze.nullack()`.

**`analyze.rtscts(`**⟨fs⟩**`,`** ⟨fields⟩**`,`** ⟨amac⟩**`,`** ⟨bmac⟩**`,`** ⟨rate⟩**`)`**

wertet RTS-CTS-Sequenzen aus. Der Parameter ⟨rate⟩ gibt die Bitrate der RTS-Frames an.

**`analyze.tsfinject(`**⟨fsini⟩**`,`** ⟨fsmon⟩**`,`** ⟨amac⟩**`,`** ⟨bmac⟩**`,`** ⟨rate⟩**`)`**

wertet ein auf NULL-Frames basierendes Zeitstempelverfahren aus. In ⟨fsini⟩ werden die Frames des Initiator-Devices übergeben. Der Parameter ⟨fsmon⟩ enthält die Parameter des Monitor-Devices.

| Bezeichner | Timer |
|---|---|
| `mactime` | TSF-Zeitstempel |
| `tsctime` | TSC-Zeitstempel |
| `hpettime` | HPET-Zeitstempel |
| `pcaptime` | PCAP-Zeitstempel |

Tabelle 4.5: Timer-Bezeichner für die Analyseroutinen

Am ehesten wird die Benutzung des Frameworks durch Betrachtung eines Beispiels deutlich. Das Auswerteframework wird analog zum Messframework gestartet.

```
# cd auswertetool
$ lua -i tool.lua
Lua 5.1.4  Copyright (C) 1994-2008 Lua.org, PUC-Rio
>
```

Im betrachteten Beispiel sollen alle NULL-ACK-Sequenzen in der Datei `/tmp/mon.pcap` analysiert werden. Zunächst werden die Frames aus der betreffenden Datei in eine Lua-Datenstruktur überführt.

```
> fs = import.import("/tmp/mon.pcap")
> print(#fs)
36088
> for k, v in pairs(fs[3]) do print(k, v) end
num             3
len             28
bitrate         54
preamble        0
ofdm            1
mactime         3080431935
tsctime         3840873292
hpettime        3954861103
pcaptime        0.000007000
type            2
subtype         4
duration        172
seq             0
da              00:12:f0:87:2d:96
sa              00:c0:ca:35:1b:e2
bssid           02:12:f0:1a:f7:4a
```

Es wurden insgesamt 36088 Frames importiert. Der letzte Teil des Listings soll noch einmal die Darstellung eines Frames als Lua-Tabelle verdeutlichen. Ein Frame besteht aus einer Liste von Schlüssel-Wert-Paaren. Im Beispiel ist ein 28 Byte langer NULL-Frame (Typ 2, Subtyp 4) gezeigt, der mit 54 MBit/s (OFDM-Modulation) von Station `00:c0:ca:35:1b:e2` zu Station `00:12:f0:87:2d:96` übertragen wurde.

```
> analyze.nullack(fs)
8623    00:c0:ca:35:1b:e2 --> 00:12:f0:87:2d:96 @ 54 MBit/s
```

Eine erste Voranalyse zeigt, dass unter den 36088 Frames insgesamt 8623 verwertbare NULL-ACK-Sequenzen zwischen den Stationen `00:c0:ca:35:1b:e2` und `00:12:f0:87:2d:96` existieren.

```
> ms = analyze.nullack(fs, { "mactime", "tsctime" },
         "00:c0:ca:35:1b:e2", "00:12:f0:87:2d:96", 54)
```

```
> print(#ms)
8623
> for k, v in pairs(ms[1]) do print(k, v) end
frame           table: 0x979310
mactime         49
tsctime         0
> print(fs[3])
table: 0x979310
```

Eine Analyse für die TSF- und TSC-Zeitstempel liefert eine Liste mit 8623 Messwerten. Durch die letzten beiden Anweisungen wird nochmals der in Abbildung 4.12 gezeigte Aufbau der Rückgabedatenstruktur verdeutlicht. Der Frame-Zeiger des ersten Messwerts verweist auf Frame 3, der der Startframe der ersten NULL-ACK-Sequenz ist.

```
> ms_mac = list.proj(ms, "mactime")
```

Aus den Messwerten können nun mittels `list.proj` alle TSF-Zeitstempeldifferenzen in ein Array übertragen werden.

```
> hist = util.hist(ms_mac)
> for k, v in pairs(hist) do print(k, v) end
-491471         1
-229327         3
-98255          3
-32719          4
48              33
49              8320
50              249
392             1
1253            1
1361            1
1718            1
2275            1
2315            1
98353           1
229425          2
491569          1
```

Ein erstes Histogramm zeigt noch eine Reihe von Messfehlern. Diese Messfehler können durch modifiziertes $\kappa$-$\sigma$-Clipping beseitigt werden ($\kappa = 10; \sigma = 3; \mu = 1{,}1$).

```
> ms_mac = util.kappasigma(ms_mac, 10, 3, 1.1)
> print(#ms_mac)
8602
> hist = util.hist(ms_mac)
> for k, v in pairs(hist) do print(k, v) end
49      8320
48      33
50      249
```

Nach dem Clippen bleiben 8602 verwertbare Messwerte übrig. Das Histogramm ist nun wesentlich schmaler. Anschließend kann der Schätzer (3.9) zur Anwendung gebracht werden, der die mittlere Zeitstempeldifferenz zu 49.03 $\mu$s bestimmt.

```
> mean, var = util.meanvar(ms_mac)
> print(mean, var)
49.025110439433          0.032156277767994
```

Durch die Verwendung von Schleifen (siehe Listing 4.3) kann mit einfachen Mitteln die Auswertung einer ganzen Messreihe bewerkstelligt werden, ohne das Framework erweitern zu müssen. Die Ergebnisse können dabei zur weiteren Verarbeitung direkt in eine Datei geschrieben werden.

Listing 4.3: Automatisierte Auswertung einer Messreihe mit dem Auswerteframework

```
-- Eingabedateien
files =
{
  [3] =
  {
    "/tmp/messung-3m-1.pcap",
    "/tmp/messung-3m-2.pcap",
    "/tmp/messung-3m-3.pcap",
    "/tmp/messung-3m-4.pcap",
    "/tmp/messung-3m-5.pcap"
  },
  [6] =
  {
    "/tmp/messung-6m-1.pcap",
    "/tmp/messung-6m-2.pcap",
    "/tmp/messung-6m-3.pcap",
    "/tmp/messung-6m-4.pcap",
    "/tmp/messung-6m-5.pcap"
  },
  ...
}

-- Reihenauswertung vornehmen
io.output("messung.dat")
for dist, file_list in pairs(files) do
  for _, file in ipairs(file_list) do
    fs     = import.import(file)
    ms     = analyze.nullack(fs, { "mactime" }, ...)
    ms_mac = list.proj(ms, "mactime")
    ms_mac = util.kappasigma(ms_mac, ...)
    mean   = util.meanvar(ms_mac)
    io.write(string.format("%d\t%f\n", dist, mean))
  end
end
io.flush()
```

# Kapitel 5

# Referenzmessungen

Mit dem im letzten Kapitel beschriebenen Framework wurden zwei Referenzversuche durchgeführt. Der erste Versuch sollte die beschriebenen Verfahren in Ad-hoc-Netzen anwenden. Dazu kamen zwei Notebooks zum Einsatz, die eine Ad-hoc-Zelle bildeten. In einem zweiten Versuch wurden die beschriebenen Verfahren in einem Infrastrukturnetzwerk auf dem Campus der Universität zur Anwendung gebracht. Für beide Versuche wurde ein RTS-CTS-NULL-ACK-Verfahren durchgeführt. Es kam dazu ein Adapter vom Typ AWUS036EH (RTL8187-Chipsatz) als Initiator zum Einsatz, da dieser keinen eigenen Rate-Control-Algorithmus implementiert. Das TOF-Verfahren wurde von zwei Monitor-Geräten vom Typ IWL6000 bzw. TL-WN422G aufgezeichnet. Das beobachtete RTS-CTS-NULL-ACK-Verfahren ermöglicht durch Verwerfen des RTS-CTS-Handshakes auch die Auswertung eines NULL-ACK-Verfahrens. Für jede Referenzentfernung wurde eine Messreihe für die Bitraten 2 MBit/s, 11 MBit/s und 54 MBit/s erfasst, um eine Aussage über den Einfluss der verschiedenen Modulationsverfahren treffen zu können.

## 5.1 Versuch 1 (Ad-hoc-Modus)

Der Versuch wurde auf einem langen geraden Gang im Hauptgebäude der TU Chemnitz durchgeführt. Im Abstand von 3 bis 30 Meter wurde jeweils in 3 m-Schritten eine Messung vorgenommen. Als Reflektorstation kam ein IBM Thinkpad vom Typ R52 zum Einsatz, welches auf einem Tisch am Ende des Ganges platziert wurde. Das Modell R52 verfügt an der oberen und der rechten Seite des Bildschirmrahmens über je eine WLAN-Flächenantenne.[1] Diese Antennen weisen bei senkrecht aufgestelltem Bildschirm jeweils eine Richtcharakterisitik in Blickrichtung aufwärts und in Blickrichtung rechts auf. Das Reflektor-Thinkpad wurde im eingeklappten Zustand so positioniert, dass die obere Antenne Sichtverbindung zur Initiatorstation hat. Als Initiatorstation kam ein Thinkpad vom Typ T410 zum Einsatz. Dieses verfügt ebenfalls über drei Flächenantennen im oberen Bildschirmrahmen mit einer ähnlichen Richtcharakteristik.[2] Das Display des Initiators befand sich zur Durchführung im aufgeklappten Zustand. Die externen Antennen der USB-WLAN-Sticks waren senkrecht ausgerichtet und

[1] vgl. Abschnitt 2030 in [TP08]
[2] vgl. Abschnitt 2060 in [TP10]

direkt vor dem Display-Rahmen des T410 befestigt. Die Fresnelzone zwischen den Antennen war frei von Hindernissen. Pro Bitrate und Abstand wurden je fünf Messungen zu 10000 RTS-CTS-NULL-ACK-Sequenzen durchgeführt. Zwischen dem Einschleusen der Sequenzen in das MAC80211-Subsystem verging jeweils etwa eine Millisekunde.

| IWL | | | | ATH | | | |
|---|---|---|---|---|---|---|---|
| $\widetilde{\Delta}$ | $n$ | $\widetilde{\Delta}$ | $n$ | $\widetilde{\Delta}$ | $n$ | $\widetilde{\Delta}$ | $n$ |
| 48 | 4866 | 1068 | 1 | -32720 | 1 | 32817 | 5 |
| 49 | 2179 | 1131 | 1 | -32719 | 7 | 98353 | 1 |
| 311 | 2 | 1171 | 1 | -30407 | 1 | 229425 | 1 |
| 331 | 3 | 1231 | 1 | 48 | 1412 | 4161585 | 1 |
| 352 | 1 | 1449 | 1 | 49 | 7339 | | |
| 531 | 1 | 1697 | 1 | 267 | 1 | | |
| 611 | 1 | 2154 | 1 | 1092 | 1 | | |
| 952 | 1 | 2653 | 1 | 1182 | 1 | | |
| 1014 | 1 | 2933 | 1 | 1384 | 1 | | |
| 1020 | 1 | 2134 | 1 | 1652 | 1 | | |

(alle Angaben für $\widetilde{\Delta}$ in $\mu$s)

Tabelle 5.1: Verteilung der TSF-$\widetilde{\Delta}$

Zunächst soll die statistische Verteilung der $\widetilde{\Delta}$ untersucht werden. Es werden ausschließlich der TSF-Timer und der TSC-Timer betrachtet. Die HPET-Zeitstempel sind vergleichbar mit den TSC-Zeitstempeln, da beide im selben Augenblick erfasst werden. Die PCAP-Zeitstempel versprechen bzgl. Auflösung und Jitter keinen Gewinn gegenüber den TSC-Zeitstempeln bzw. den TSF-Zeitstempeln. Tabelle 5.1 zeigt die Verteilung der TSF-Zeitstempeldifferenzen bei einer 3 m-Messung für 54 MBit/s. Der überwiegende Teil der Messwerte verteilt sich in der Nähe der zu erwartenden Größe $\Delta$. Vereinzelt treten Messfehler auf, wenn Frames nicht korrekt empfangen werden konnten und dadurch ein NULL-Frame mit einem ACK-Frame der Folgesequenz verglichen wurde. Die großen Abweichungen beim Atheros-Adapter erklären sich durch die TSF-Anomalie. Nach Anwendung des $\kappa$-$\sigma$-Verfahrens ($\kappa = 2; \sigma = 2$) bleiben nur noch die Größen 48 $\mu$s und 49 $\mu$s übrig.

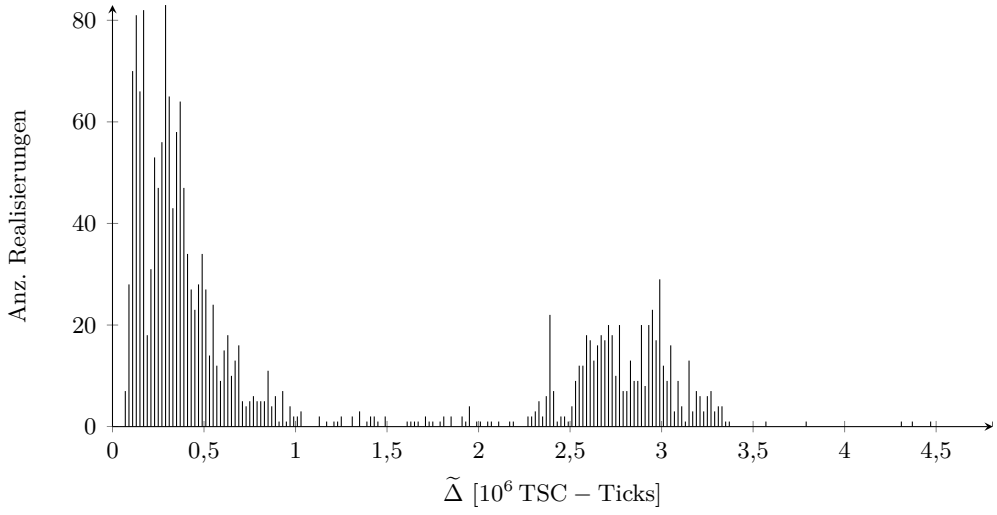



Abbildung 5.1: Verteilung der TSC-$\widetilde{\Delta}$ des Intel-Treiber (54 MBit/s)

Für die selbe Messung lieferten die TSC-Zeitstempel des Atheros-Treiber keine verwertbaren Messwerte. In 8754 von 8773 Fällen war die TSC-Differenz 0. Die übrigen 19 Fälle können als Messfehler angesehen werden. Für den Intel-Adapter ergaben 5115 von 7066 Realisierungen (=72,4%) Null als Differenz. Die restlichen Realisierungen sind im Histogramm in Abbildung 5.1 dargestellt. Negative Realisierungen, die durch einen Überlauf des TSC entstanden, wurden verworfen. Für die Grafik wurden jeweils 20000 benachbarte TSC-Werte zu einem Balken zusammengefasst.

Im linken Teil des Histogramms streuen die Werte um über 350 $\mu$s (eine Mikrosekunde entspricht in etwa 2660 TSC-Ticks). Im Bereich von 48 $\mu$s ist der erste Ausschlag mit mehr als 80 Realisierungen zu verzeichnen. Dennoch ist diese Ausbeute bei 7000 Messungen insgesamt noch sehr gering. Die Ursache für die Verteilung auf der rechten Seite des Histrogramms ist unklar. Auffällig ist jedoch, dass sie die gleiche Breite wie die linke Verteilung besitzt, und um ca. eine Millisekunde zu dieser versetzt ist.

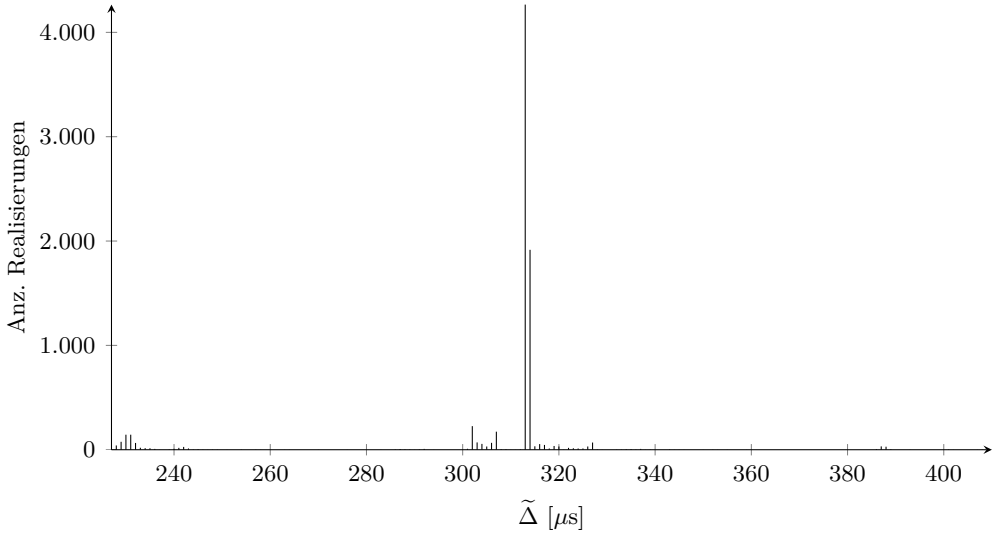
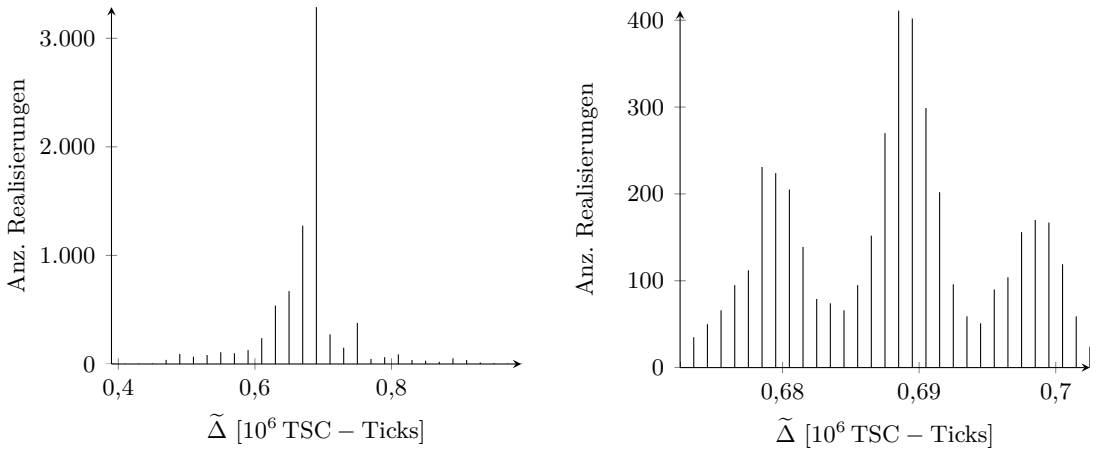



Abbildung 5.2: Verteilung der TSF-$\widetilde{\Delta}$ des Intel-Treiber (2 MBit/s)

Ein auffälliges Verhalten zeigen die TSF-Zeitstempel des Intel-Treibers bei den DSSS-Bitraten (siehe Abbildung 5.2). Die Zeitstempel nehmen geringere Werte an als überhaupt möglich ist. Dieser Effekt kann nicht durch Empfangsfehler erklärt werden, da diese die $\widetilde{\Delta}$-Werte nur vergrößern. Für den selben Versuch konnte der Atheros-Adapter jedenfalls keine TSF-Differenzen geringer als den Sollwert messen. Der Fehler ist demnach auf den Intel-Adapter einzugrenzen. Die genaue Ursache konnte nicht ermittelt werden. Da die Verteilung der Messwerte jedoch nicht all zu stark gestört ist, können diese Messfehler problemlos durch Clipping beseitigt werden.

Die TSC-Zeitstempel des Intel-Treiber streuen bei niedrigen Bitraten deutlich weniger als bei 54 MBit/s (siehe Abbildung 5.3). Die Daten für Abbildung 5.3b wurden wieder durch $\kappa$-$\sigma$-Clipping gewonnen ($\kappa = 10; \sigma = 2$). Dennoch ist unter den gefilterten Messwerten keine eindeutige Häufung auszumachen, und es bleibt fraglich, ob die TSC-Zeitstempel zur Laufzeitmessung geeignet sind. Durch die verhältnismäßig lange Übertragungszeit ist nun auch beim Atheros-Treiber nur eine geringe Anzahl der TSC-Zeitstempeldifferenzen Null. Aller-




*(20000 akkumulierte Ticks pro Balken)*

(a) ungefiltert

*(1000 akkumulierte Ticks pro Balken)*

(b) gefiltert

Abbildung 5.3: Verteilung der TSC-$\widetilde{\Delta}$ des Intel-Treiber (2 MBit/s)

dings streuen diese Werte in einem Bereich von $10^6$ Ticks und besitzen mehrere weit auseinander liegende Häufungspunkte, was sie für die weitere Auswertung unbrauchbar macht. Bei beiden Hardware-Modellen ist das Verhalten für 2 MBit/s und 11 MBit/s ähnlich, weswegen auf eine genauere Betrachtung für letztere Bitrate verzichtet wird.

Mit zunehmender Entfernung ist eine Verschiebung des Histogramms zu den höheren Realisierungswerten zu erwarten. Abbildung 5.4 stellt die Histogramme für alle betrachteten Entfernungen in einem Diagramm dar. Der erwartete Effekt ist deutlich erkennbar.

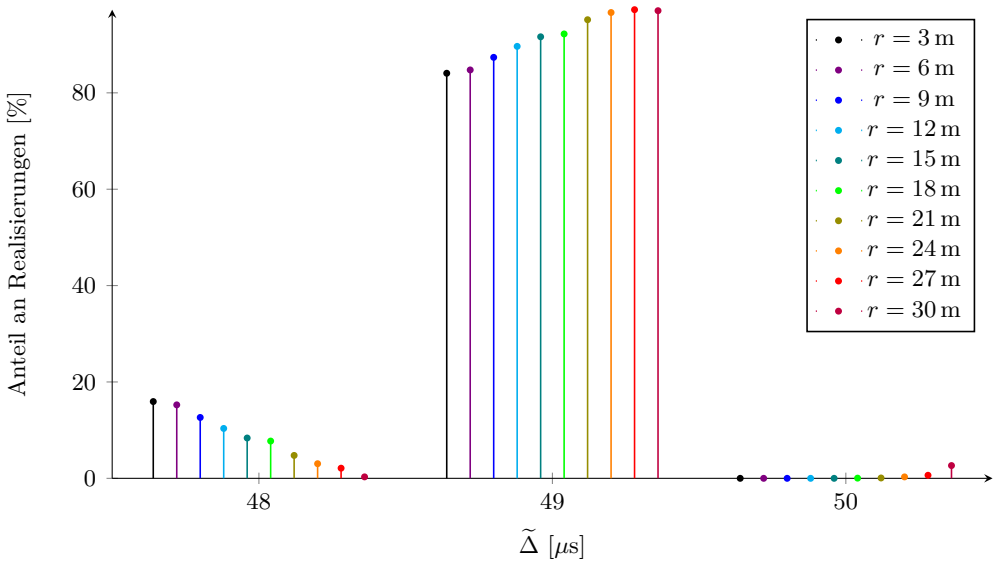



Abbildung 5.4: Verteilung der TSF-$\widetilde{\Delta}$ in Abhängigkeit von der Entfernung (Atheros, NULL-ACK-Verfahren, 54 MBit/s)

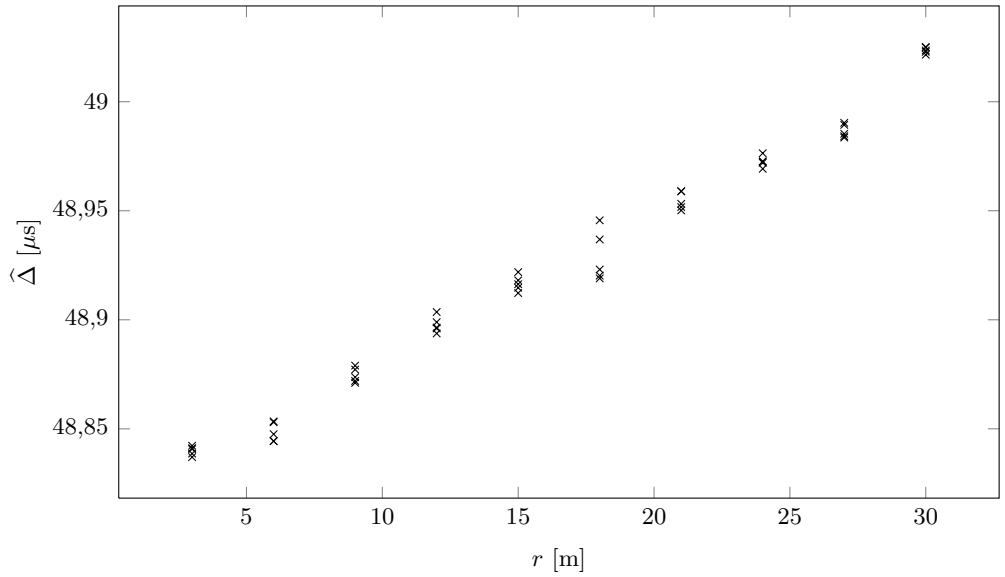



Abbildung 5.5: Korrelation zwischen Entfernung und $\widehat{\Delta}$ (TSF, Atheros, NULL-ACK-Verfahren, 54 MBit/s)

Auf diese Verteilungen kann nun der Schätzer (3.9) angewendet werden. Für jede Referenzentfernung stellt Abbildung 5.5 den Schätzwert $\widehat{\Delta}$ aller fünf Messungen dar. Der Korrelationskoeffizient zwischen $\widehat{\Delta}$ und $r$ beträgt 0,993. Die Abhängigkeit ist damit fast perfekt positiv linear. Die zu Grunde liegenden Messwerte wurden durch modifiziertes $\kappa$-$\sigma$-Clipping gefiltert ($\kappa = 10; \sigma = 3; \mu = 1{,}1$). Substituiert man den NULL-ACK-Ansatz

$$\Delta = 2t + d_{\text{ACK}} + g \tag{5.1}$$

mit Hilfe von Gleichung (2.1), erhält man die lineare Funktion

$$\Delta(r) = \frac{2}{c} \cdot r + \underbrace{d_{\text{ACK}} + g}_{\Delta_0}. \tag{5.2}$$

Eine bestmögliche Anpassung der Parameter $c$ und $\Delta_0$ an die Daten aus Tabelle 5.2 ergibt

$$c = 300.786.255\,\text{m/s} \quad \text{und} \quad \Delta_0 = 48{,}81\,\mu\text{s}.$$

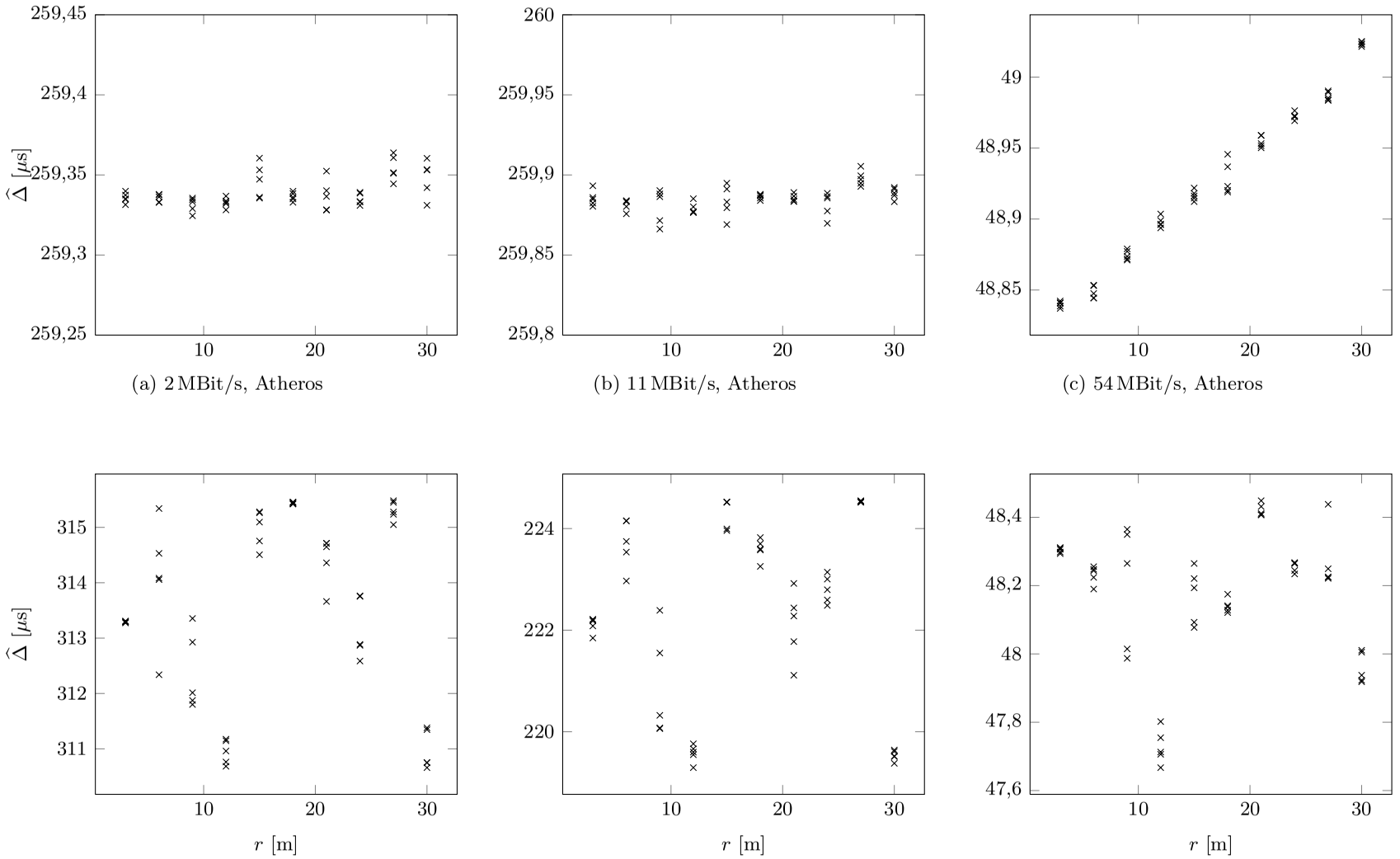
Der Wert von $c$ weicht lediglich um 0,36% von der Lichtgeschwindigkeit in Luft ab. Der lineare Anstieg der gemessenen Laufzeit ist folglich mit an Sicherheit grenzender Wahrscheinlichkeit durch die zunehmende Entfernung zu erklären. Entsprechend der Gleichungen (2.20) wären für $\Delta_0$ 44 $\mu$s zu erwarten. Der um über 4 $\mu$s größere Wert kann aber durch die bereits in Abbildung 4.7 aufgedeckten Effekte erklärt werden. Der genaue Wert von $\Delta_0$ ist für die praktische Anwendung von geringerer Bedeutung. Da er im Vorfeld durch Messung bestimmt werden kann und nicht von der Entfernung abhängt, ist eine entsprechende Kalibrierung der $\Delta$-Schätzungen möglich.

Solch deutliche Korrelationen wie oben konnten jedoch nicht für jede Variante der Verfahrensparameter erzielt werden, wie die Abbildungen 5.6 bis 5.8 zeigen. Unter den TSC-Zeitstempeln konnte in keinem Fall eine Korrelation ausfindig gemacht werden. Auf die Darstellung der entsprechenden Diagramme wird deshalb verzichtet.

| $r$ [m] | $n$ | $\widehat{\Delta}$ [$\mu$s] | $r$ [m] | $n$ | $\widehat{\Delta}$ [$\mu$s] | $r$ [m] | $n$ | $\widehat{\Delta}$ [$\mu$s] |
|---|---|---|---|---|---|---|---|---|
| 3 | 8329 | 48,8407 | 15 | 9409 | 48,9219 | 27 | 8365 | 48,9903 |
| | 8215 | 48,8411 | | 8845 | 48,9123 | | 9313 | 48,9853 |
| | 8178 | 48,8423 | | 8688 | 48,9147 | | 9429 | 48,9836 |
| | 7652 | 48,8370 | | 8963 | 48,9163 | | 8536 | 48,9842 |
| | 8751 | 48,8386 | | 9193 | 48,9181 | | 9136 | 48,9894 |
| 6 | 8235 | 48,8443 | 18 | 9749 | 48,9231 | 30 | 8602 | 49,0251 |
| | 7504 | 48,8475 | | 9856 | 48,9456 | | 8679 | 49,0215 |
| | 8060 | 48,8444 | | 9839 | 48,9190 | | 8644 | 49,0235 |
| | 7709 | 48,8534 | | 9800 | 48,9368 | | 8651 | 49,0249 |
| | 8192 | 48,8530 | | 9756 | 48,9203 | | 8560 | 49,0229 |
| 9 | 8256 | 48,8720 | 21 | 9794 | 48,9517 | | | |
| | 8209 | 48,8737 | | 9677 | 48,9589 | | | |
| | 8338 | 48,8790 | | 9771 | 48,9502 | | | |
| | 8175 | 48,8772 | | 9784 | 48,9532 | | | |
| | 8285 | 48,8711 | | 9757 | 48,9590 | | | |
| 12 | 8914 | 48,8989 | 24 | 9609 | 48,9720 | | | |
| | 8686 | 48,8960 | | 9592 | 48,9692 | | | |
| | 8577 | 48,8965 | | 9489 | 48,9726 | | | |
| | 8348 | 48,9036 | | 9535 | 48,9726 | | | |
| | 8235 | 48,8937 | | 9479 | 48,9764 | | | |

($n$ ... Anzahl verwendbarer Messwerte nach Filterung)

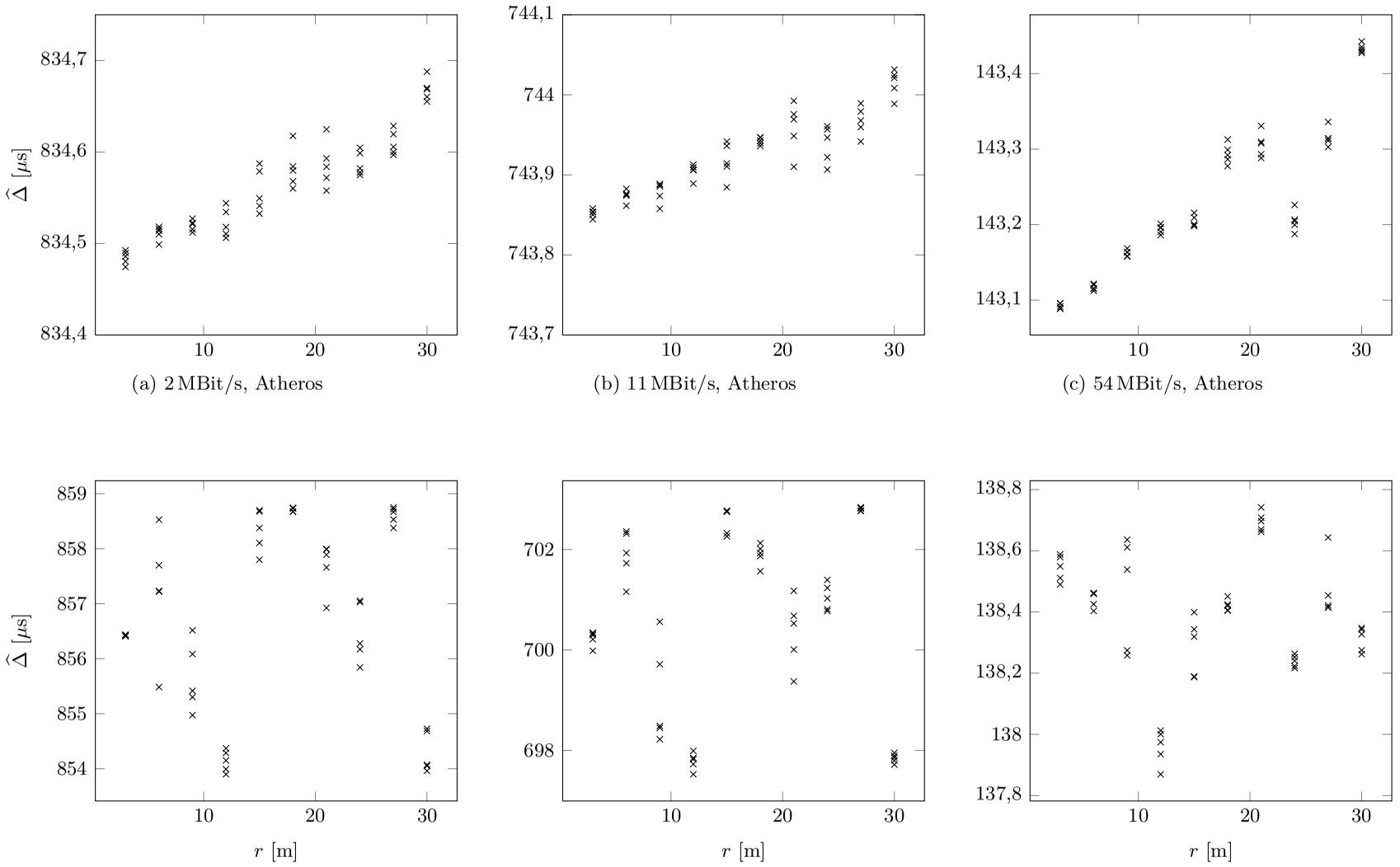
Tabelle 5.2: Messwerte zu Abbildung 5.5




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

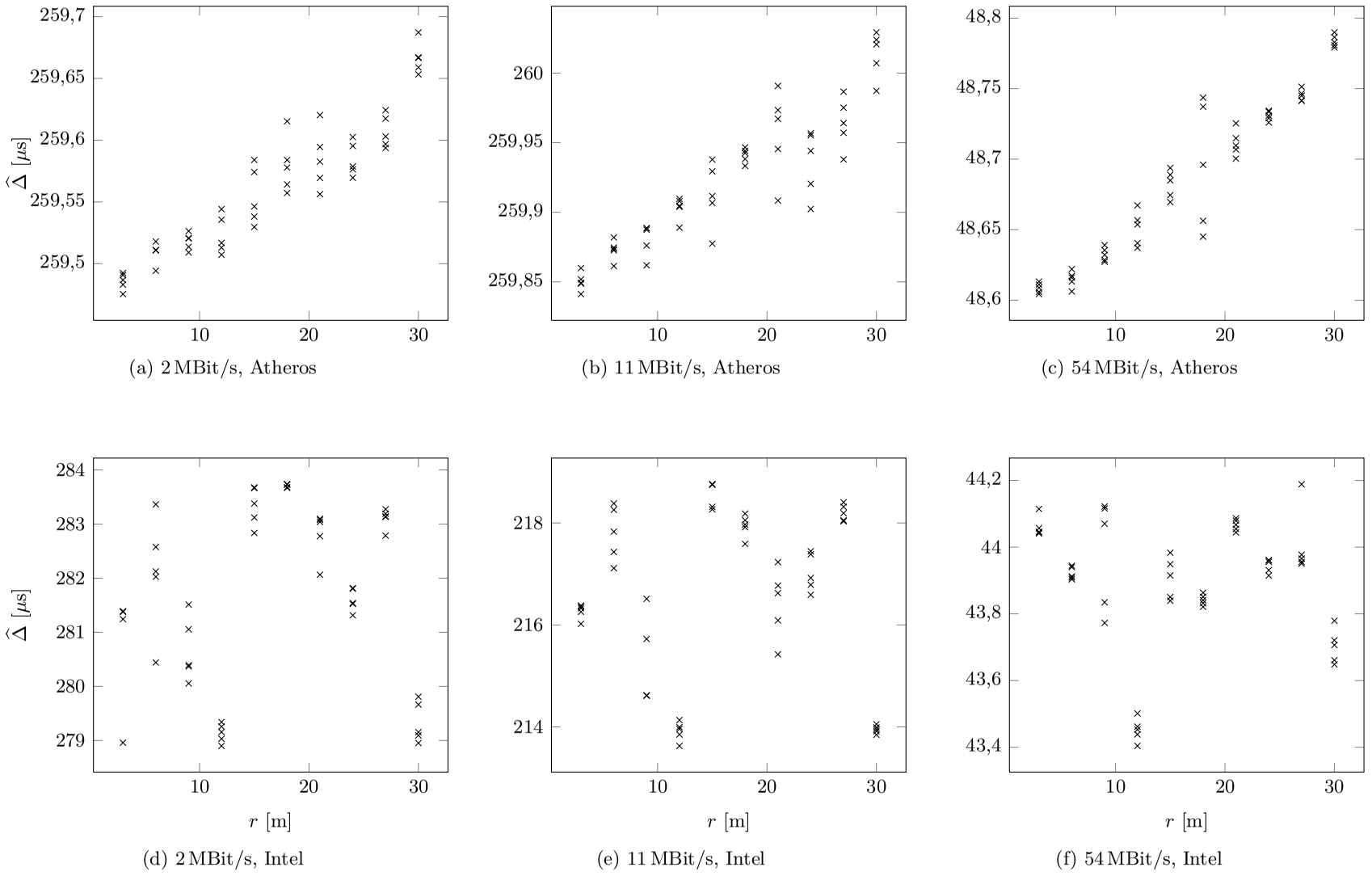
Abbildung 5.6: Korrelation $\widehat{\Delta}$ vs. $r$ für das NULL-ACK-Verfahren (TSF)




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

Abbildung 5.7: Korrelation $\widehat{\Delta}$ vs. $r$ für das RTS-CTS-NULL-ACK-Verfahren (TSF)




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

Abbildung 5.8: Korrelation $\widehat{\Delta}$ vs. $r$ für das RTS-CTS-Verfahren (TSF)

| Hardware | Verfahren | Bitrate | Korr.-Koeff. | $c$ [m/s] | $\Delta_0$ [$\mu$s] |
|---|---|---|---|---|---|
| Atheros | NULL-ACK | 54 MBit/s | 0,993 | $300.786.255 \pm 1{,}69\%$ | 48,81 |
| Atheros | NULL-ACK | 11 MBit/s | 0,398 | — | — |
| Atheros | NULL-ACK | 2 MBit/s | 0,482 | — | — |
| Atheros | RTS-CTS-NULL-ACK | 54 MBit/s | 0,896 | $388.296.736 \pm 7{,}15\%$ | 143,06 |
| Atheros | RTS-CTS-NULL-ACK | 11 MBit/s | 0,917 | $759.159.131 \pm 6{,}29\%$ | 743,84 |
| Atheros | RTS-CTS-NULL-ACK | 2 MBit/s | 0,933 | $682.998.294 \pm 5{,}57\%$ | 834,47 |
| Atheros | RTS-CTS | 54 MBit/s | 0,963 | $311.889.866 \pm 4{,}06\%$ | 48,58 |
| Atheros | RTS-CTS | 11 MBit/s | 0,913 | $384.196.463 \pm 6{,}44\%$ | 259,84 |
| Atheros | RTS-CTS | 2 MBit/s | 0,931 | $346.636.671 \pm 6{,}65\%$ | 259,47 |
| Intel | NULL-ACK | 54 MBit/s | −0,043 | — | — |
| Intel | NULL-ACK | 11 MBit/s | −0,009 | — | — |
| Intel | NULL-ACK | 2 MBit/s | 0,027 | — | — |
| Intel | RTS-CTS-NULL-ACK | 54 MBit/s | −0,058 | — | — |
| Intel | RTS-CTS-NULL-ACK | 11 MBit/s | 0,017 | — | — |
| Intel | RTS-CTS-NULL-ACK | 2 MBit/s | 0,060 | — | — |
| Intel | RTS-CTS | 54 MBit/s | −0,095 | — | — |
| Intel | RTS-CTS | 11 MBit/s | −0,022 | — | — |
| Intel | RTS-CTS | 2 MBit/s | 0,098 | — | — |
| *nach manueller Entfernung von Messfehlern* | | | | | |
| Atheros | RTS-CTS-NULL-ACK | 54 MBit/s | 0,969 | $345.309.829 \pm 3{,}88\%$ | 143,05 |
| Atheros | RTS-CTS | 54 MBit/s | 0,987 | $311.914.186 \pm 2{,}42\%$ | 48,58 |

Tabelle 5.3: Kenngrößen zu den Abbildungen 5.6 bis 5.8

Auf den ersten Blick ist erkennbar, dass sich beim Intel-Adapter keine Korrelation einstellt. Dennoch verhalten sich die Messwerte für unterschiedliche Bitraten bei gleichem Abstand ähnlich, wie besonders aus einem Vergleich zwischen den Ergebnissen für 2 MBit/s und 11 MBit/s ersichtlich wird. In dieser Hinsicht ist interessant, dass die zu Grunde liegenden Messreihen unabhängig voneinander erfasst wurden. Folglich scheinen die Messfehler systematischer Natur zu sein. Zur Bestimmung der genauen Ursachen sind allerdings weiterführende Messungen notwendig. Denkbar wäre, dass die starken Streuungen durch die mehrfach ausgelegten Empfangsantennen am Intel-Adapter hervorgerufen werden. Die HF-Eingangsstufen unterliegen mit Sicherheit bauteilbedingten Toleranzen, die sich auf die zeitliche Korrelation zwischen dem Symbolstrom auf dem Drahtlosmedium und dem dekodierten Datenstrom am Ausgang des Demodulators auswirken. Aufgrund der kurzen Entfernung zwischen Initiator-Interface und Monitor-Interface kann es auch zu einer Übersteuerung der HF-Komponenten gekommen sein, die sich ebenfalls negativ auf die Symbolkorrelation auswirken könnte.

Beim Atheros-Adapter lässt sich in den meisten Fällen eine Korrelation feststellen. Lediglich bei den DSSS-Bitraten liefert das NULL-ACK-Verfahren keine verwendbaren Messwerte. Ob der Grund hierfür direkt im NULL-ACK-Verfahren zu suchen ist, kann durch die vorliegenden Messwerte nicht festgestellt werden. Denkbar wäre, dass das vorausgehende RTS-CTS-Handshake den IPW2200-Adapter in der Reflektorstation so beeinflusst, dass der Sendezeitpunkt des ACK-Frames vom Empfangszeitpunkt des RTS-Frames abhängt. Diese Vermutung lässt sich überprüfen, indem der Versuch ausschließlich mit NULL-ACK-Sequenzen wiederholt wird, anstatt die NULL-ACK-Daten aus einem RTS-CTS-NULL-ACK-Verfahren zu gewinnen. Ist dann wieder eine Korrelation feststellbar, liegt vermutlich eine zeitliche Abhängigkeit zwischen RTS- und ACK-Frame beim IPW2200-Adapter vor. Wegen des fehlenden zweiten TOF-Verfahrens in den RTS-CTS-NULL-ACK-Sequenzen sind die $c$-Werte der entsprechenden RTS-CTS-NULL-ACK-Auswertung um das Doppelte überschätzt.

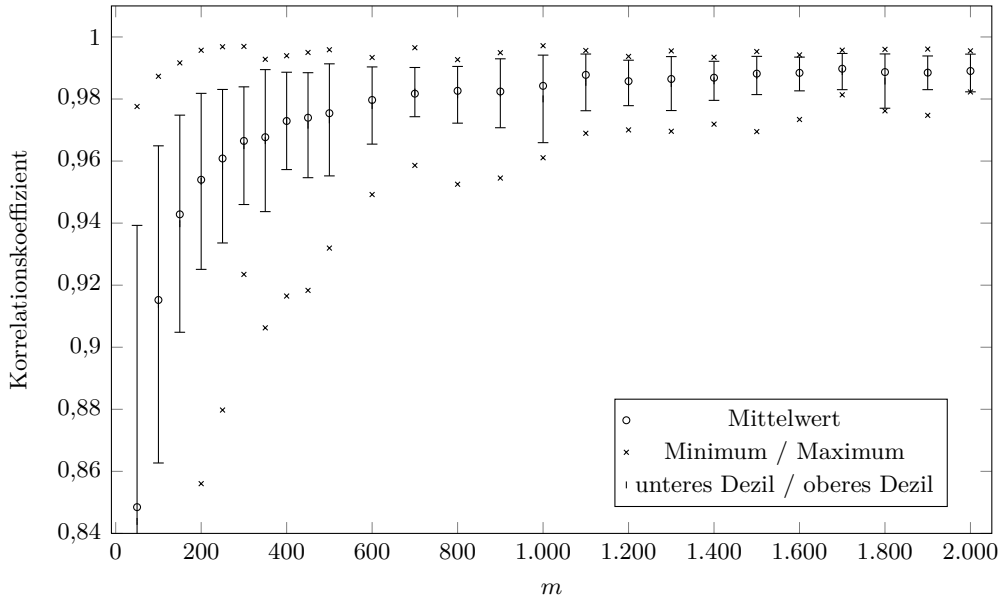



Abbildung 5.9: Abhängigkeit des Korrelationskoeffizienten von der Messreihengröße

Bei der obigen Auswertung der Messreihen gingen jeweils alle $\widetilde{\Delta}$-Werte in die $\Delta$-Schätzung ein. Es stellt sich nun die Frage, ob sich auch bereits mit einer geringeren Messreihengröße eine ausreichende Korrelation zwischen Schätzwert und Entfernung feststellen lässt. Zur Beantwortung dieser Frage werden die der Abbildung 5.5 zu Grunde liegenden Messreihen herangezogen und eine Analyse analog zu Abbildung 10 in [GH05] vorgenommen. Aus den ca. 7000 bis 9000 Einzelmessungen werden für jede Referenzentfernung jeweils $m$ Messwerte zufällig ausgewählt. Für jede Referenzentfernung wird nun anhand der Auswahl eine Schätzung berechnet. Aus den Entfernungen und Laufzeitschätzungen kann letztendlich der Korrelationskoeffizient bestimmt werden. Enthält die Ausgangsmessreihe $n$ Einzelmessungen, können bis zu $\left\lfloor \frac{n}{m} \right\rfloor$ solcher Korrelationskoeffizienten berechnet werden, ohne dass eine Einzelmessung mehrfach in die Korrelationsrechnung eingeht. Abbildung 5.9 zeigt die Verteilung dieser Korrelationskoeffizienten in Abhängigkeit von $m$. Man erkennt, dass bereits bei 200 Einzelmessungen ein Korrelationskoeffizient größer als 0,95 zu erwarten ist. Mit 90%-iger Wahrscheinlichkeit lag ab 400 Einzelmessungen eine Korrelation besser als 0,95 vor. Ab 800 Messungen waren alle Koeffizienten größer als 0,95. Für mehr als 1000 Messungen ist keine weitere Verbesserung erkennbar. Dies deckt sich mit den Erkenntnissen in [GH05], in dem die Autoren ebenfalls 1000 Einzelmessungen vorschlagen. Bei einem Abstand von einer Millisekunde zwischen den Messungen wäre folglich die Entfernung zweier WLAN-Stationen innerhalb einer Sekunde bestimmbar.

Abschließend soll das Vertrauensintervall einer $\Delta$-Schätzung betrachtet werden. Die Breite des Vertrauensintervalls wird durch die Standardabweichung $\sigma$ festgelegt, die sich nach Gleichung (3.11) aus

$$\sigma = P \cdot \sqrt{\frac{\left\langle \frac{\Delta}{P} \right\rangle \cdot \left(1 - \left\langle \frac{\Delta}{P} \right\rangle\right)}{m}} \tag{5.3}$$

ergibt, wobei hier im Nenner die Messreihengröße $m$ an Stelle von $n$ einzusetzen ist. Der

Wert von $P$ beträgt für den TSF-Zähler 1 $\mu$s. Einen Einfluss auf die Genauigkeit haben somit nur die Größen $\Delta$ und $m$. Nimmt man $\widehat{\Delta}$ als normalverteilt an, kann mittels $\sigma$ die Größe des Vertrauensintervalls durch die Quantile der Normalverteilung abgeschätzt werden. Der Wert $\widehat{\Delta}$ liegt demnach mit einer 68%-igen Wahrscheinlichkeit im Intervall $[\,\widehat{\Delta} - \sigma,\, \widehat{\Delta} + \sigma]$. Für ein 90%-Niveau erhöht sich die Schwankung auf $1{,}64\sigma$. Ein 50%-Niveau ist bereits bei $0{,}67\sigma$ erreicht. Die Annahme der Normalverteilung ist ferner gerechtfertigt. Entsprechend Abschnitt 3.2 basiert $\widehat{\Delta}$ auf einer Binomialverteilung. Nach dem Satz von Moivre-Laplace geht diese Verteilung für

$$m \cdot \left\langle \frac{\Delta}{P} \right\rangle \cdot \left(1 - \left\langle \frac{\Delta}{P} \right\rangle\right) \geq 9$$

in eine sehr gute Nährung einer Normalverteilung über. Bei 1000 Einzelmessungen darf sich $\left\langle \frac{\Delta}{P} \right\rangle$ somit fast im gesamten Intervall $[\,0, 1\,[$ bewegen, ohne die Annahme zu verletzen.

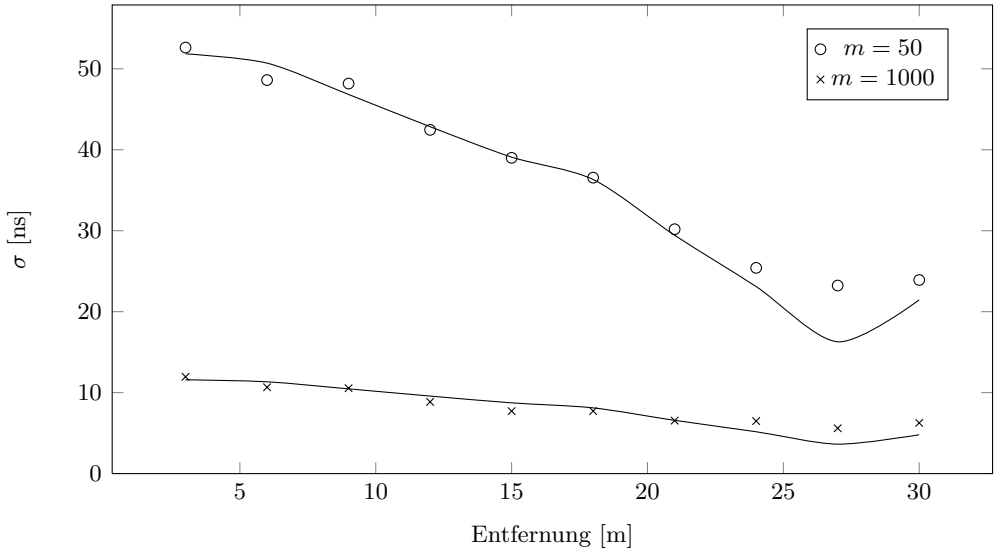


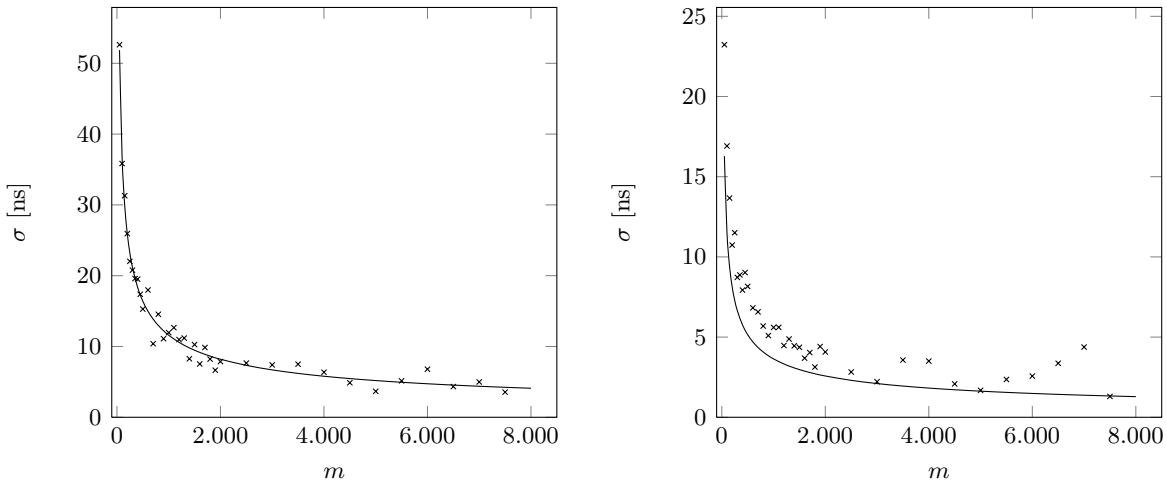

Abbildung 5.10: Einfluss von $\Delta$ auf das Vertrauensintervall

Abbildung 5.10 zeigt die empirische Standardabweichung verschiedener Entfernungen, wie sie sich aus den $\widehat{\Delta}$-Werten in Tabelle 5.1 ergeben. Die Linien im Diagramm zeigen dabei die laut Gleichung (5.3) zu erwartenden Werte an. In Abbildung 5.11 ist noch ergänzend die Abhängigkeit der Standardabweichung von $m$ dargestellt.

Die empirischen Werte liegen zum Großteil sehr nah an den zu erwartenden Werten. Dies ist ein Indiz dafür, dass die Laufzeitmessung die Genauigkeit erreicht, die entsprechend dem Modell in Abschnitt 3.2 möglich ist. Nicht modellierte Störeinflüsse können folglich ausgeschlossen werden. Anhand der Ergebnisse wird deutlich, dass bei 3000 Einzelmessungen Genauigkeiten von bis zu 3,6 ns für ein 90%-Niveau erreicht werden konnten. Dies entspricht einer Positionierungsgenauigkeit von 1,2 m. Bei 5000 Einzelmessungen verbessert sich die Genauigkeit auf ca. 90 cm.

| $\sigma$[ns] | $r$ [m] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| $m =$ 50 | 52,6 | 48,6 | 48,2 | 42,5 | 39,0 | 36,6 | 30,2 | 25,4 | 23,2 | 23,9 |
| 100 | 35,8 | 35,4 | 34,6 | 29,0 | 29,1 | 24,8 | 20,9 | 18,9 | 16,9 | 17,2 |
| 500 | 15,3 | 13,9 | 14,9 | 13,1 | 11,9 | 11,3 | 8,8 | 8,6 | 8,2 | 7,2 |
| 1000 | 12,0 | 10,7 | 10,6 | 8,9 | 7,7 | 7,7 | 6,5 | 6,5 | 5,6 | 6,3 |
| 2000 | 7,9 | 6,9 | 6,3 | 9,2 | 6,2 | 5,5 | 5,4 | 5,0 | 4,1 | 3,0 |
| 3000 | 7,4 | 9,2 | 6,6 | 6,0 | 4,2 | 4,7 | 3,5 | 3,6 | 2,2 | 4,2 |
| 4000 | 6,4 | 6,3 | 5,1 | 4,0 | 4,4 | 3,3 | 1,6 | 2,3 | 3,5 | 2,4 |
| 5000 | 3,7 | 6,4 | 6,0 | 5,3 | 3,4 | 2,2 | 3,0 | 3,0 | 1,7 | 3,3 |
| 7500 | 3,6 | 3,1 | 3,5 | 4,0 | 5,6 | 2,8 | 2,5 | 2,1 | 1,3 | 2,1 |

Tabelle 5.4: Daten zu den Abbildungen 5.10 und 5.11 (Auszug)




(a) $r = 3\,\mathrm{m}$ (b) $r = 27\,\mathrm{m}$

Abbildung 5.11: Einfluss von $m$ auf das Vertrauensintervall

## 5.2 Versuch 2 (Infrastrukturmodus)

Der zweite Versuch fand im Obergeschoss des zentralen Hörsaal- und Seminargebäudes der TU Chemnitz statt, dessen Grundriss in Abbildung 5.12 dargestellt ist. Ziel des Versuchs war es, Signallaufzeiten in einem Infrastrukturnetzwerk zu messen. Der Access Point ist im Bauch einer Skulptur vor dem Hörsaal N112 installiert. Da sich diese Figur in zufälligen Zeitabständen automatisch bewegt, konnte eine gleichbleibende Ausrichtung der Antennen während des Versuchs nicht sichergestellt werden. Die Initiatorstation hatte eine identische Konfiguration wie im ersten Versuch. Sie wurde an den mit 1 bis 5 gekennzeichneten Punkten im Gang links neben den Hörsälen N114 und N115 platziert. Die bauliche Situation erschwerte die Messung zusätzlich. Zwischen Initiatorstation und Access Point bestand keine Sichtverbindung. Die Verbindungslinie schneidet die aus bewehrtem Beton bestehende Seiten- und Bodenwand des Hörsaals N114. Zwischen den Planquadraten F4 und F5, sowie C6 bis D8 ist ein gusseinsernes Geländer installiert, was die Ausbreitung der elektromagnetischen Wellen zusätzlich behindert (vgl. Abschnitt 2.1). Die Abstände zwischen den Stationen wurden mit Hilfe des maßstäblichen Grundrisses abgeschätzt.

Die Messreihen wurden mit den selben Parametern erfasst wie in Versuch 1. Die Anzahl der Messreihen pro Entfernung wurde indes von fünf auf zehn Durchgänge vergrößert. Im Anschluss an die RTS-CTS-NULL-ACK-Verfahren wurden zusätzlich je zehn Durchläufe eines Zeitstempelverfahrens für die Bitraten 2 MBit/s, 11 MBit/s und 54 MBit/s durchgeführt.

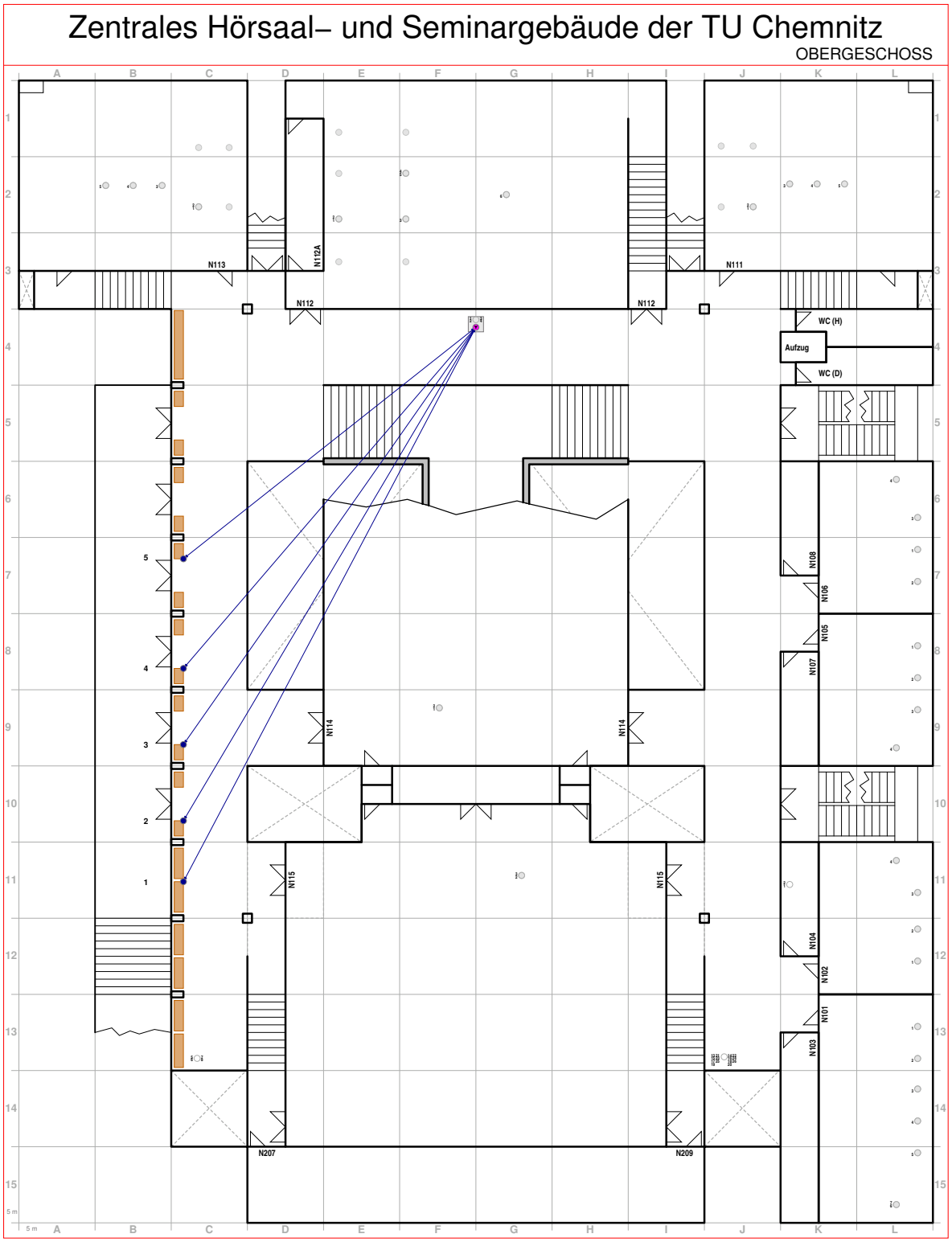
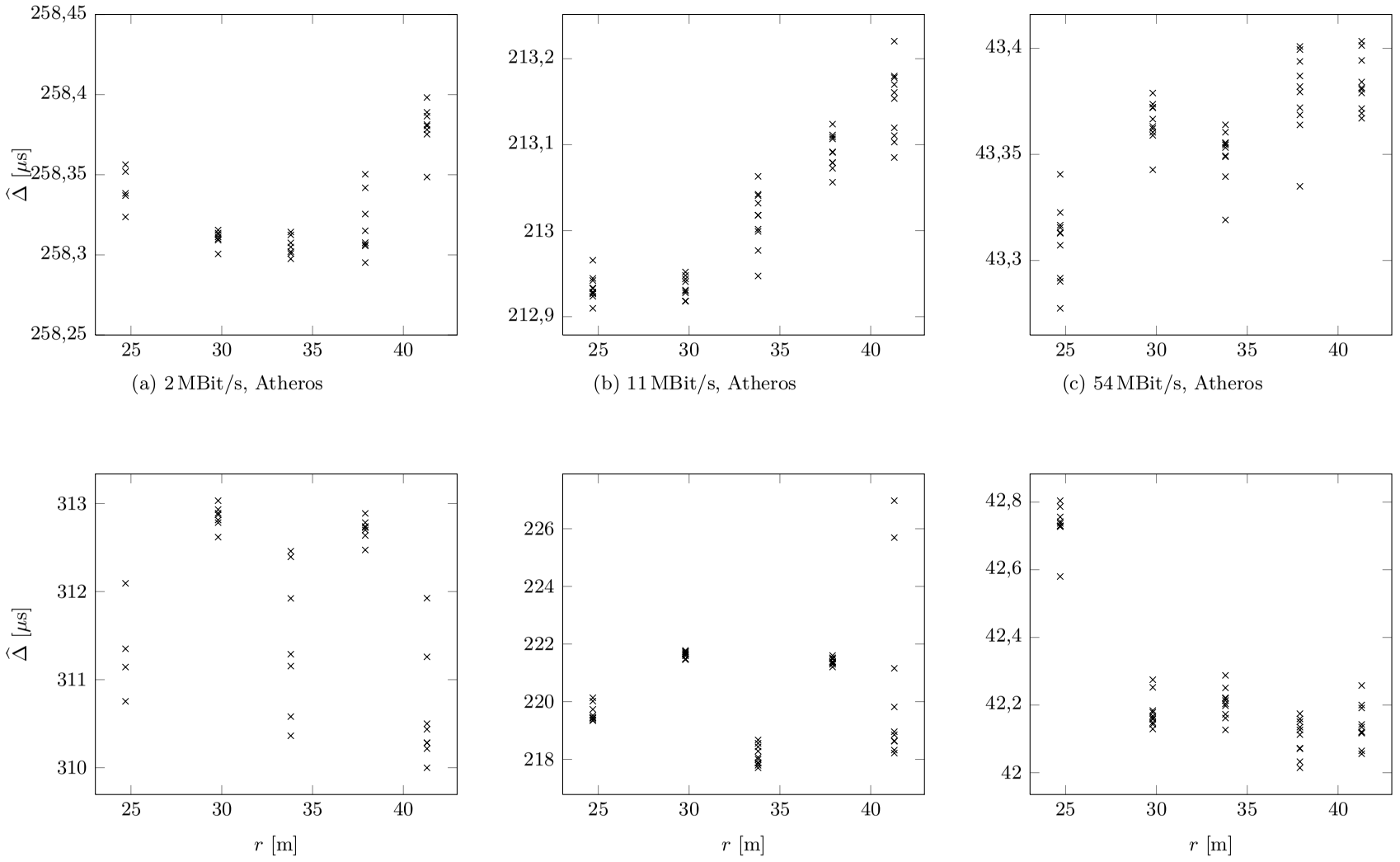



Abbildung 5.12: Lageplan des zentralen Hörsaal- und Seminargebäudes der TU Chemnitz

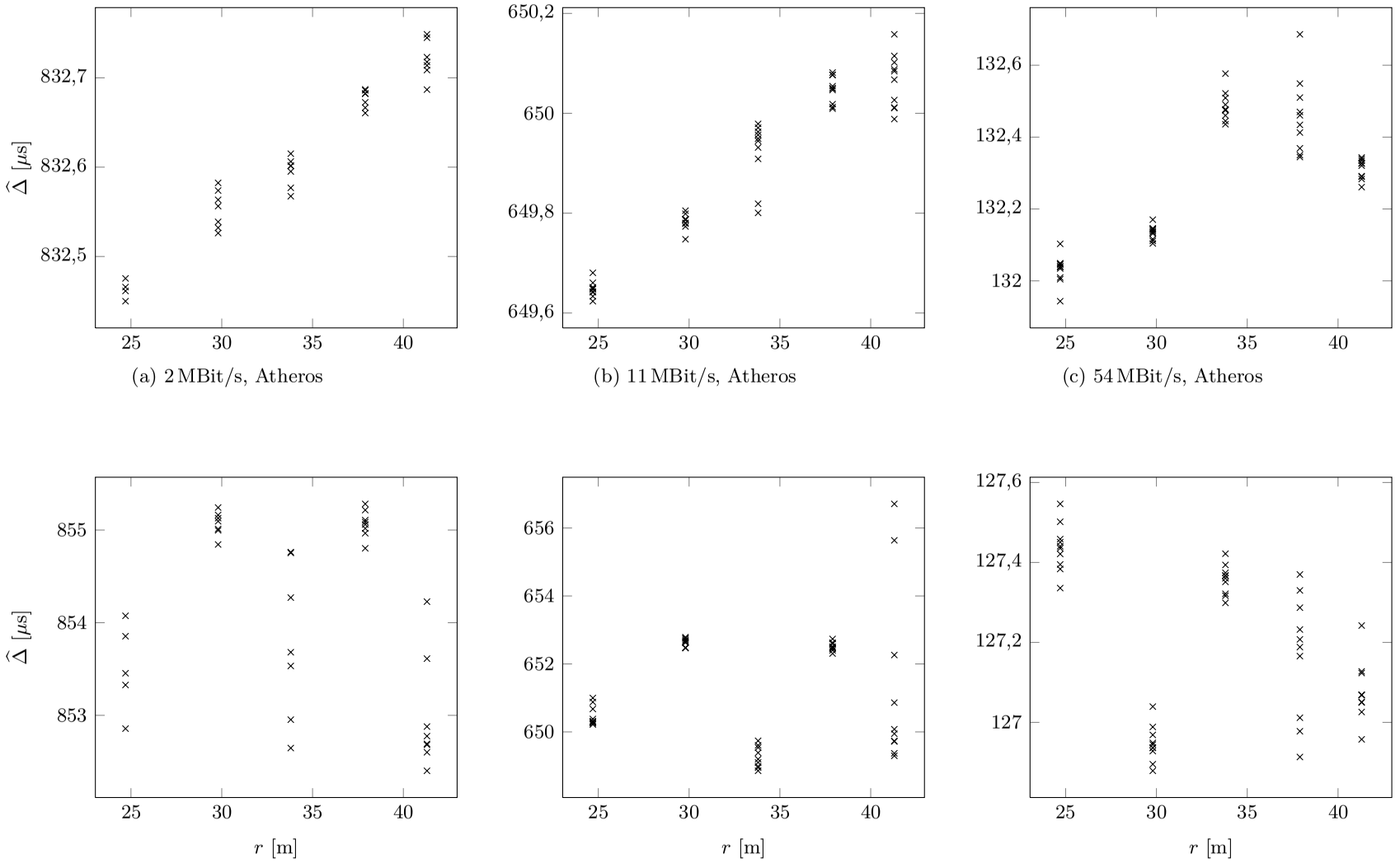
*Die Grundriss-Grafik basiert der Arbeit von Ronald Schmidt und wurde von ihm mit freundlicher Genehmigung zur Verfügung gestellt.*




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

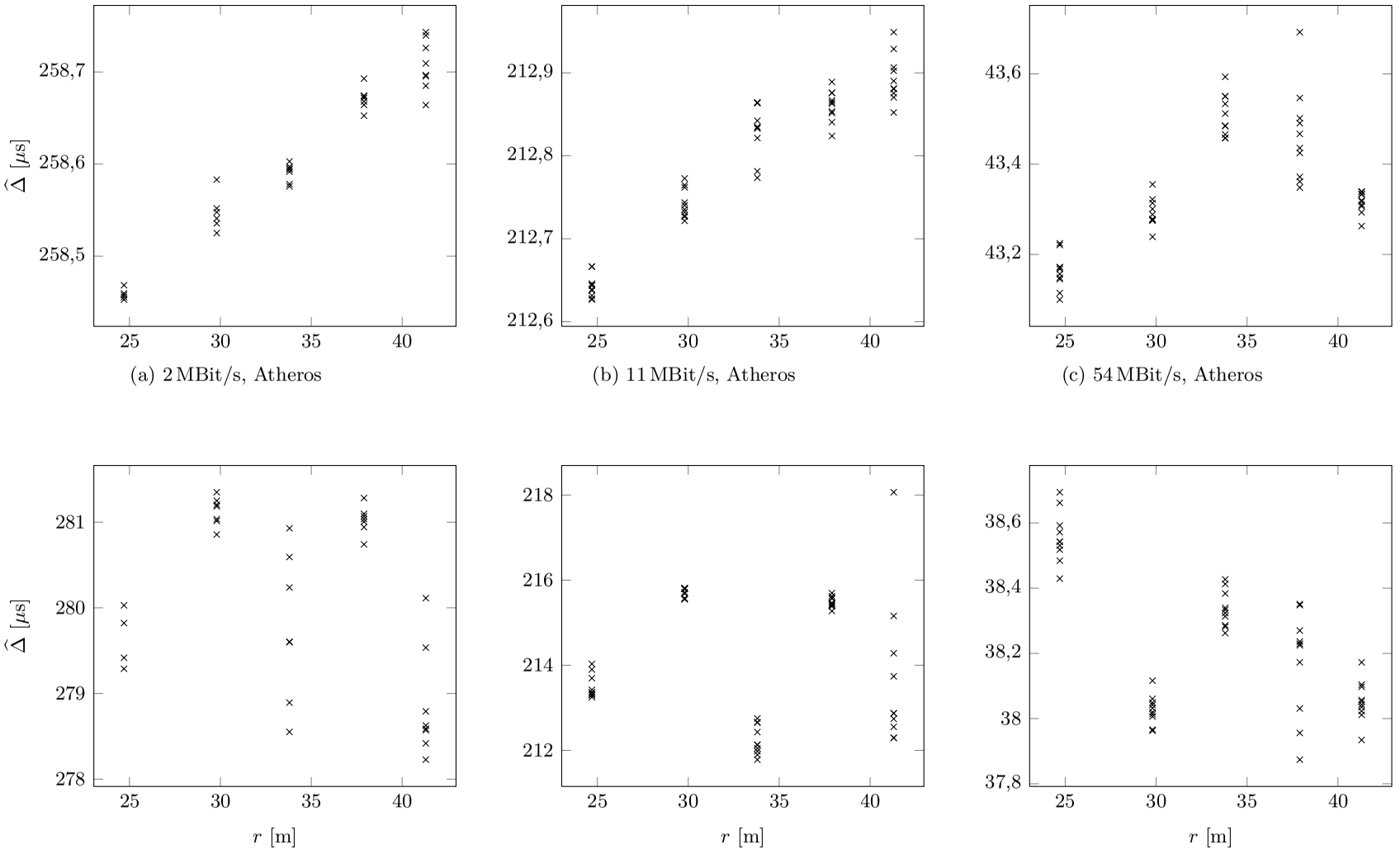
Abbildung 5.13: Korrelation $\widehat{\Delta}$ vs. $r$ für das NULL-ACK-Verfahren (TSF)




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

Abbildung 5.14: Korrelation $\widehat{\Delta}$ vs. $r$ für das RTS-CTS-NULL-ACK-Verfahren (TSF)




(a) 2 MBit/s, Atheros (b) 11 MBit/s, Atheros (c) 54 MBit/s, Atheros

(d) 2 MBit/s, Intel (e) 11 MBit/s, Intel (f) 54 MBit/s, Intel

Abbildung 5.15: Korrelation $\widehat{\Delta}$ vs. $r$ für das RTS-CTS-Verfahren (TSF)

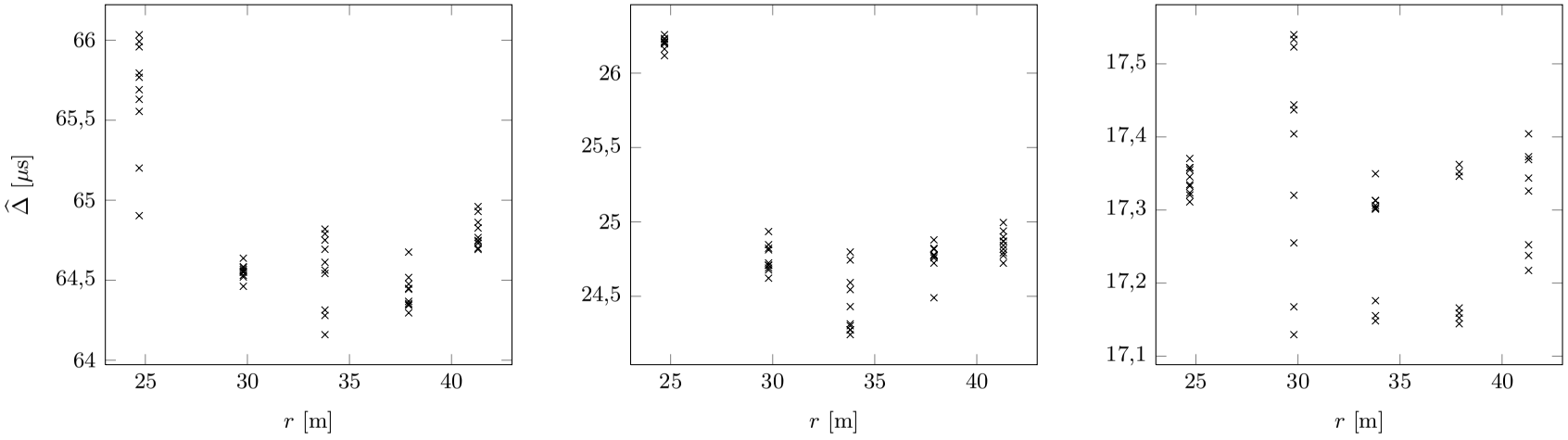
| Hardware | Verfahren | Bitrate | Korr.-Koeff. | $c$ [m/s] | $\Delta_0$ [$\mu$s] |
|---|---|---|---|---|---|
| Atheros | NULL-ACK | 54 MBit/s | 0,776 | 496.766.052 ± 11,73% | 43,22 |
| Atheros | NULL-ACK | 11 MBit/s | 0,915 | 142.886.740 ± 6,33% | 212,56 |
| Atheros | NULL-ACK | 2 MBit/s | 0,429 | — | — |
| Atheros | RTS-CTS-NULL-ACK | 54 MBit/s | 0,703 | 179.405.986 ± 14,60% | 131,54 |
| Atheros | RTS-CTS-NULL-ACK | 11 MBit/s | 0,959 | 149.290.496 ± 4,24% | 649,00 |
| Atheros | RTS-CTS-NULL-ACK | 2 MBit/s | 0,978 | 259.605.399 ± 3,87% | 832,09 |
| Atheros | RTS-CTS | 54 MBit/s | 0,539 | — | — |
| Atheros | RTS-CTS | 11 MBit/s | 0,951 | 131.867.842 ± 4,71% | 212,29 |
| Atheros | RTS-CTS | 2 MBit/s | 0,978 | 133.163.771 ± 3,80% | 258,09 |
| Intel | NULL-ACK | 54 MBit/s | −0,788 | — | — |
| Intel | NULL-ACK | 11 MBit/s | 0,134 | — | — |
| Intel | NULL-ACK | 2 MBit/s | −0,310 | — | — |
| Intel | RTS-CTS-NULL-ACK | 54 MBit/s | −0,380 | — | — |
| Intel | RTS-CTS-NULL-ACK | 11 MBit/s | 0,156 | — | — |
| Intel | RTS-CTS-NULL-ACK | 2 MBit/s | −0,185 | — | — |
| Intel | RTS-CTS | 54 MBit/s | −0,593 | — | — |
| Intel | RTS-CTS | 11 MBit/s | 0,038 | — | — |
| Intel | RTS-CTS | 2 MBit/s | −0,318 | — | — |
| *nach manueller Entfernung von Messfehlern* | | | | | |
| Atheros | RTS-CTS-NULL-ACK | 54 MBit/s | 0,968 | 240.452.531 ± 4,92% | 131,63 |
| Atheros | RTS-CTS | 54 MBit/s | 0,749 | 249.522.476 ± 16,72% | 43,00 |

Tabelle 5.5: Kenngrößen zu den Abbildungen 5.13 bis 5.15

Die Messungen liefern ähnliche Ergebnisse wie im ersten Versuch. Die Streuung der Schätzwerte ist jedoch tendenziell stärker. Zunächst lässt sich feststellen, dass der Intel-Adapter auch in diesem Fall keine verwertbaren Ergebnisse liefert. Dies stützt die These, dass die Ursache für dieses Verhalten im Hardware-Aufbau des Intel-Adapters zu suchen ist und nicht in der Reflektorstation begründet ist. Der Atheros-Adapter liefert für DSSS-Bitraten deutlich lineare Korrelationen. Im Bereich der OFDM-Raten sind die Messungen stärker gestört. Bei den 54 MBit/s-Messungen mit RTS-CTS-Handshake erscheint es so, als wären die $\widehat{\Delta}$-Werte für die Referenzentfernungen 33.8 m und 37.9 m generell zu groß geschätzt. Die Streuung und der systematische Fehler lassen sich durch Ausbreitungseffekte erklären. Vermutlich nahm die Welle nicht den „direkten Weg“ zum Empfänger, da dieser wegen der baulichen Gegebenheiten eine hohe Dämpfung aufwies. Das Signal wurde damit höchstwahscheinlich mehrfach an den Wänden reflektiert, bis es den Empfänger erreichte. OFDM-Übertragungen sind generell toleranter, was die Mehrwegausbreitung betrifft. Dies führt dazu, dass nicht alle Einzelmessungen die Signallaufzeit für den selben Ausbreitungsweg bestimmen, was in einer höheren Streuung resultiert.

Bei Betrachtung der $c$-Werte einer linearen Regression an die Datenpunkte (siehe Tabelle 5.5) fällt auf, dass die Lichtgeschwindigkeit fast immer stark unterschätzt wird. Diese starken Abweichungen sind auch nicht durch die geringe Lichtgeschwindigkeit in evt. passierten Wänden zu erklären. Es liegt näher, dass das Signal mehrfach reflektiert wurde und der „Funkweg“ bis auf das Doppelte des geometrischen Wegs anwuchs.

In Versuch 1 konnte bei den DSSS-Bitraten keine Korrelation für die NULL-ACK-Sequenzen festgestellt werden (siehe Abbildungen 5.6a und 5.6b). Bei den Messungen am Access Point kann dieses Verhalten nicht festgestellt werden, was darauf hindeutet, dass der im ersten Versuch beobachtete Effekt durch die Hardware der Reflektorstation verursacht wird.




(a) 2 MBit/s (b) 11 MBit/s (c) 54 MBit/s

Abbildung 5.16: Korrelation $\widehat{\Delta}$ vs. $r$ für das Zeitstempelverfahren

Als letzter Punkt in diesem Kapitel soll noch auf die Auswertung des Zeitstempelverfahrens eingegangen werden. Die Korrelationsidagramme für die drei verwendeten Bitraten sind in Abbildung 5.16 dargestellt. In keinem der Fälle ist eine Korrelation feststellbar. Bei Betrachtung des Verhaltens des Intel-Adapters zu den NULL-ACK-Versuchen ist dieses Verhalten zu erwarten. Die Sende- und Empfangszeitstempel werden vom Intel-Adapter erzeugt und unterliegen demnach den selben Streuungseffekten, die eine Laufzeitmessung unmöglich machen.

# Kapitel 6

# Ausblick

## 6.1 Zusammenfassung

Im Rahmen dieser Arbeit konnte gezeigt werden, dass eine Laufzeitmessung von WLAN-Signalen, die die notwendige Genauigkeit für eine Positionsbestimmung besitzt, prinzipiell möglich ist. Spezialhardware ist hierfür nicht erforderlich. In Ergänzung zu ähnlichen Arbeiten wie [GH05] konnte das NULL-ACK-Messverfahren vorgeschlagen werden, welches keine Assoziierung der Stationen in Infrastrukturnetzwerken erfordert, das Distribution System nicht involviert und nicht durch kryptografische Sicherungsmechanismen beeinflusst wird. Nach Kenntnis des Autors wurde dieses Verfahren bisher noch nicht zur Laufzeitmessung herangezogen. Die Schwierigkeiten, die sich bei der Umsetzung der NULL-ACK-Methode im Vergleich zu bereits in der Literatur präsentierten Methoden ergeben, konnten durch die Nutzung des Linux MAC80211-Subsystems gelöst werden. Zur Integration in zukünftige Arbeiten wurde ein Framework entwickelt, welches auf MAC80211 aufsetzt. Das ISO-Abbild einer Linux Live-CD, die das Framework samt eines entsprechend gepatchten Linux-Kerns enthält, befindet sich auf der DVD in der Anlage zu dieser Arbeit.

Die anfangs aufgestellten Anforderungen konnten zum größten Teil erfüllt werden. Ob die Eigenen- bzw. Fremdlokalisierung möglich ist, hängt nicht von der Implementierung, sondern vom Messszenario ab. Die Hardware-Unabhängigkeit wird nur von WLAN-Adaptern bzw. deren Treibern eingeschränkt, die der Anwendung keine TSF-Zeitstempel zur Verfügung stellen. Da jedes WLAN-Gerät aber einen TSF-Zähler besitzt, sollten die Zeitstempel mit vertretbarem Aufwand zugänglich gemacht werden können. Das NULL-ACK-Verfahren und alle seine Ableger sind in bestehende WLAN-Netze integrierbar. Das Zeitstempelverfahren überschreitet die Grenzen des WLAN-Standards und sollte demnach nicht weiter in Betracht gezogen werden.

Eine praxistaugliche Möglichkeit zur WLAN-Lokalisierung verspricht diese Arbeit hingegen noch nicht. Bei den durchgeführten Probeversuchen wurden dazu noch zu viele Störeinflüsse festgestellt. Problematisch stellt sich vor allem die offensichtlich fehlende Eignung einiger WLAN-Adapter wie dem IWL6000 für Laufzeitlokalisierung dar. In gewissem Sinne ist es daher schon fast ironisch, dass die brauchbarsten Ergebnisse durch einem USB-Stick mit weniger als 8 € Anschaffungswert erzielt wurden, dessen Zeitbasis zudem noch inkonsistente Zeitstem-

pel liefert. Ob sich die Messwerte der weniger geeigneten WLAN-Geräte durch verbesserte Filterverfahren dennoch nutzen lassen, ist allerdings nicht auszuschließen.

Die Messergebnisse deuten sehr darauf hin, dass das Multipath-Problem den Trilaterationsansatz zur Positionsbestimmung unbrauchbar macht. Weitere Untersuchungen müssen zeigen, ob evt. mit der Fingerprinting-Methode dennoch eine Lokalisierung möglich ist. Die Signallaufzeit steht auf Grund der Multipath-Effekte nicht in direktem Zusammenhang mit dem geometrischen Abstand der Stationen, ist aber höchstwahrscheinlich doch charakteristisch für den Ort der zu lokalisierenden Station.

Eine wesentliche Einschränkung für eine praktische Umsetzung bildet die zusätzliche Monitorschnittstelle, deren einzige Funktion es ist, die fehlenden Sendezeitstempel zu kompensieren. An dieser Stelle erscheint es lohnend, zuallerst dieses Problem in späteren Weiterentwicklungen zu beseitigen. Denkbar wäre eine Lösung, die von den eingangs aufgestellten Anforderungen abweicht und doch Modifikationen an der Firmware der WLAN-Adpater vornimmt.

## 6.2 Zukünftige Entwicklungen

Auf dem Gebiet der laufzeitbasierten WLAN-Lokalisierung gibt es noch eine Reihe weiterer Ansätze, die über den Umfang dieser Arbeit hinaus gehen und Bestandteil zukünftiger Arbeiten sein könnten.

Zunächst wäre es lohnend, die hier gezeigten Versuche in 802.11n-Netzen durchzuführen, deren Verbreitung derzeit immer weiter zunimmt. Die im Rahmen dieser Arbeit entwickelte Software kann für diese Versuche ohne weitere Modifikationen verwendet werden. Referenzenmessungen waren im Erstellungszeitraum dieser Arbeit nicht möglich, da erste 802.11n-fähige Access Points innerhalb der Technischen Universität Chemnitz erst in jüngster Vergangenheit ausgerollt wurden. Die Unterstützung von 802.11n im Ad-hoc-Modus war in diesem Zeitraum ebenfalls noch Gegenstand der Kernel-Entwicklung.[1]

Eine wichtige Voraussetzung zur Lokalisierung ist, dass in der Nähe der Mobilstation ausreichend Referenzstationen im Empfangsreichweite sind, um genug Messungen vornehmen zu können, die ein Lösen der Trilaterationsgleichungen erlaubt. In Infrastrukturnetzen bedeutet dies, dass sich Zellen überlappen müssen. Gerade im 2.4 GHz-Band ist dieser Zustand unerwünscht, da nur maximal drei sich nichtüberschneidende 20 MHz-Kanäle bereitstehen. Aus diesem Grund sollte zur Lokalisierungszwecken auf das breitere 5 GHz-Band ausgewichen werden, indem die räumliche Überlappung von Funkzellen kein Problem darstellt.

Für die in Abschnitt 3.6 vorgeschlagenen Szenarien zur Hyperbelnavigation konnten im Rahmen dieser Arbeit keine Versuche durchgeführt werden, da die entsprechende Hardware für die Vielzahl der beteiligten Stationen nicht zur Verfügung stand. Dennoch erscheint es sinnvoll, einen solchen Versuchsaufbau zu entwerfen und zu überprüfen, ob es möglich ist, Laufzeitdifferenzen mit der entsprechend notwendigen Genauigkeit zu bestimmen.

Mit der Ergänzung 802.11v, deren Verabschiedung am 2. Februar 2011 stattfand,[2] wurden erstmals Lokalisierungsmethoden in den WLAN-Standard aufgenommen, die auf der Mes-

[1] vgl. `http://permalink.gmane.org/gmane.linux.kernel.wireless.general/63191`
[2] vgl. `http://www.ieee802.org/11/Reports/802.11_Timelines.htm`

sung von Signallaufzeiten und Empfangsfeldstärken beruhen.[3] Der Text von 802.11v war zum Zeitpunkt der Drucklegung dieser Arbeit noch nicht öffentlich verfügbar, wodurch leider keine Aussagen über die technischen Hintergründe möglich sind. Es bleibt zu hoffen, dass 802.11v eine rasche Verbreitung erfährt. Eine standardseitige Unterstützung von Lokalisierungsverfahren lässt sehr gute Genauigkeit im Vergleich zu anderen Realisierungsmethoden erwarten.

Die Nachfolge von 802.11n sind Bereits in Form der Ergänzungen 802.11ac[4] und 802.11ad[5] absehbar. Sie sollen bis zu Bruttodatenraten in der Größe von GBit/s erzielen. Die notwendiger Weise höheren Bandbreiten erzwingen ein Ausweichen auf das 5 GHz-Band (802.11ac) bzw. das 60 GHz-Band (802.11ad), wobei sich für die Variante nach 802.11ac das selbe Überlappungsproblem anbahnt, wie es heute bereits im 2.4 GHz-Band existiert.[6] Die Verabschiedung dieser Standards ist allerdings nicht vor Ende 2012 (802.11ac) bzw. Ende 2013 (802.11ad) zu erwarten.[7] Die durch Dämpfungseffekte kleineren Zellen in den höheren Bändern kommen dabei aber prinzipiell der Lokalisierung zu Gute. Mit 802.11ad ist eine Funkzelle innerhalb von Gebäuden auf den Raum beschränkt, in dem sich die Station befindet. Durch Bestimmung der MAC-Adresse eines in Empfangsreichweite befindlichen 802.11ad Access Points kann damit breits der Raum festgestellt werden, in dem sich eine Mobilstation aufhält. Zudem besteht mit hoher Wahrscheinlichkeit eine Sichtverbindung zwischen Mobilstation und Referenzstation, was eine Betrachtung des Multipath-Problems erübrigt. Werden z. B. in den vier oberen Eckpunkten eines rechteckigen Raumes je ein 802.11ad Access Point installiert, ist innerhalb des Raumes eine Trilateration „auf Sicht“ möglich.

Einen anderen Ansatz verspricht die Erweiterung 802.11ah,[8] die Frequenzen unterhalb von 1 GHz für WLAN zugänglich machen will. Signale in diesem Frequenzband können Wände sehr gut durchdringen und kommen daher z. B. in GSM-Netzen zum Einsatz. Eine Untersuchung, ob auf diese Weise ebenfalls dem Multipath-Problem begegnet werden kann, ist sinnvoll.

Wie bereits im letzten Abschnitt angerissen wurden, können einige Schwachstellen der hier beschriebenen Ansätze durch Aufgeben einiger der eingangs formulierten Anforderungen umgangen werden. Sind Firmwaremodifikationen zugelassen, sollte es möglich sein, Sendezeitstempel bestimmen zu können. Vielversprechend hierfür erscheint der Treiber `carl9170`[9], der auf einer offenen Firmware aufsetzt. Neben der Erfassung von Sendezeitstempeln kann auch auf Zeitquellen mit einer feineren Auflösung als dem TSF zugegriffen werden.[10]

Verbesserungsmöglichkeiten existieren nicht nur auf technologischer Ebene. Mit Kalman-Filtern oder Partikelfiltern ist es möglich, nicht nur eine einzelne Positionen zu ermitteln, sondern eine ganze Bewegungstrajektorie vorherzusagen [CBC06, FRH07, FFH09]. Laufzeitmessungen können zur Verbesserung der Genauigkeit mit Feldstärkenmessungen kombiniert werden [ARW07]. Sind lediglich symbolische Positionsangaben wie „Der Nutzer befindet sich in Raum $X$“ zu bestimmen, können Markov-Ketten zur Anwendung kommen [HFL+04]. Jeder Zustand einer Markov-Kette entspricht einer bestimmten symbolischen Position. Mit einer

[3] vgl. `https://mentor.ieee.org/802.11/dcn/05/11-05-0827-09-000v-tgv-objectives.doc`
[4] `http://grouper.ieee.org/groups/802/11/Reports/tgac_update.htm`
[5] `http://grouper.ieee.org/groups/802/11/Reports/tgad_update.htm`
[6] vgl. `http://www.heise.de/netze/meldung/Zwei-Wege-zum-Gigabit-WLAN-1186315.html`
[7] vgl. `http://www.ieee802.org/11/Reports/802.11_Timelines.htm`
[8] `http://grouper.ieee.org/groups/802/11/Reports/tgah_update.htm`
[9] `http://linuxwireless.org/en/users/Drivers/carl9170`
[10] vgl. `http://permalink.gmane.org/gmane.linux.kernel.wireless.general/62467`

Initialmessung kann für jeden Zustand eine Wahrscheinlichkeit dafür bestimmt werden, dass sich der Nutzer an der entsprechenden Position aufhält. Da die möglichen Zustandsübergänge im Allgemeinen eingeschränkt sind, lässt sich diese Wahrscheinlichkeitsverteilung mit jeder Bewegung des Nutzers präzisieren.

Unter der Aussicht, dass WLAN-Funkzellen in Zukunft kleiner werden und die Hardware die Bestimmung von Signallaufzeiten unterstützt, lässt sich abschließend festhalten, dass die Lokalisierung durch Laufzeitmessungen in Zukunft aller Voraussicht nach an Bedeutung gewinnt.

# Anhänge

# Anhang A

# Hardware

| Gerät | Treiber | Bemerkung |
|---|---|---|
| IWL6000 | `iwlagn` | Minimaler Patch in `iwlagn_rx_reply_tx()` erforderlich, um TSFT in Radiotap-Header auszuliefern. Nutzt einen eigenen Rate-Control-Algorithmus, der aber mittels Kernelpatch umgangen werden kann. |
| TL-WN422G (v2) | `ath9k_htc` | USB. Externe Dipolantenne. Gute Empfängerempfindlichkeit. TSF-Zähler liefert inkonsistente Werte, die aber u. U. korrigiert werden können. Der Rate-Control-Algorithmus ist in der Firmware implementiert. Es gibt keine Möglichkeit, Frames mit einer bestimmten Datenrate einzuschleusen. Der TSF-Zeitstempel wird zum Ende der Aussendung erfasst. |
| TL-WN721N | `ath9k_htc` | Wie oben, nur interne omnidirektionale Antenne. |
| AWUS036EH | `rtl8187` | USB. Schlechte Empfängerempfindlichkeit. Externe Dipolantenne mit starker Richtcharakteristik. Ad-hoc-Modus wird vom Treiber (noch) nicht unterstützt. Frames mit beliebiger Datenrate können ohne Treibermodifikation eingeschleust werden. |

Tabelle A.1: Verwendete Hardware

# Anhang B

# Patches

## B.1 Kernel-Patches

### B.1.1 Bitraten-Korrekturpatch

```
diff -Naur --exclude-from exclude linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.c linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c
--- linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.c 2010-11-16 11:52:07.237000030 +0100
+++ linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c 2011-01-14 23:13:17.101000048 +0100
@@ -1069,16 +1069,23 @@
       break;

     case IEEE80211_RADIOTAP_RATE: {
-      info->control.rates[0].idx = 0;
+      s8 rate = 0;
+
       if (*iterator.this_arg) {
         int i;
+
         for (i = 0; i < sband->n_bitrates; i++)
-          if (sband->bitrates[i].bitrate <=
-            *iterator.this_arg * 5) {
-            info->control.rates[0].idx = i;
+        {
+          if (sband->bitrates[i].bitrate == *iterator.this_arg * 5) {
+            rate = i;
+            break;
           }
+        }
       }
+
+      info->control.rates[0].idx = rate;
+      info->control.rts_cts_rate_idx = rate;
+
       info->flags |= IEEE80211_TX_CTL_RC_BYPASS;
       info->control.rates[0].flags = 0;
       info->control.rates[1].idx = -1;
```

### B.1.2 Bitraten-Patch für IWLAGN-Treiber

```
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5/drivers/net/wireless/iwlwifi/iwl-agn-tx.c linux-2.6.35-gentoo-r5-osg-20101106/drivers/net/wireless/iwlwifi/iwl-agn-tx.c
--- linux-2.6.35-gentoo-r5/drivers/net/wireless/iwlwifi/iwl-agn-tx.c  2010-09-02 14:37:50.947999996 +0200
+++ linux-2.6.35-gentoo-r5-osg-20101106/drivers/net/wireless/iwlwifi/iwl-agn-tx.c 2010-10-28 15:12:27.366638700 +0200
@@ -421,7 +421,8 @@

  /* DATA packets will use the uCode station table for rate/antenna
   * selection */
- if (ieee80211_is_data(fc)) {
+ if (ieee80211_is_data(fc) &&
+     !(info->flags & IEEE80211_TX_CTL_RC_BYPASS)) {
    tx_cmd->initial_rate_index = 0;
    tx_cmd->tx_flags |= TX_CMD_FLG_STA_RATE_MSK;
    return;
@@ -451,8 +452,21 @@
    rate_flags |= RATE_MCS_CCK_MSK;

  /* Set up antennas */
- priv->mgmt_tx_ant = iwl_toggle_tx_ant(priv, priv->mgmt_tx_ant);
- rate_flags |= iwl_ant_idx_to_flags(priv->mgmt_tx_ant);
+ if(!ieee80211_is_data(fc))
+ {
+   priv->mgmt_tx_ant = iwl_toggle_tx_ant(priv, priv->mgmt_tx_ant);
+   rate_flags |= iwl_ant_idx_to_flags(priv->mgmt_tx_ant);
+ }
+ else
+ {
+   u8 ant_idx;
+
+   ant_idx = info->flags & IEEE80211_TX_CTL_ANTENNA ? info->antenna_sel_tx : 0;
+   if(ant_idx >= RATE_ANT_NUM)
+     ant_idx = 0;
+
+   rate_flags |= iwl_ant_idx_to_flags(ant_idx);
+ }

  /* Set the rate in the TX cmd */
  tx_cmd->rate_n_flags = iwl_hw_set_rate_n_flags(rate_plcp, rate_flags);
```

### B.1.3 Short-Preamble-Patch

```
diff -Naur --exclude-from exclude linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.c linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c
--- linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.c 2011-01-14 23:13:17.101000048 +0100
+++ linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c 2011-02-01 12:16:34.121999992 +0100
@@ -1022,6 +1022,7 @@
  struct ieee80211_tx_info *info = IEEE80211_SKB_CB(skb);
  int ret = ieee80211_radiotap_iterator_init(&iterator, rthdr, skb->len,
              NULL);
+ bool short_preamble;

  sband = tx->local->hw.wiphy->bands[tx->channel->band];

@@ -1066,6 +1067,8 @@
        info->flags &= ~IEEE80211_TX_INTFL_DONT_ENCRYPT;
      if (*iterator.this_arg & IEEE80211_RADIOTAP_F_FRAG)
        tx->flags |= IEEE80211_TX_FRAGMENTED;
+     if (*iterator.this_arg & IEEE80211_RADIOTAP_F_SHORTPRE)
```

```
17 +       short_preamble = true;
18        break;
19
20      case IEEE80211_RADIOTAP_RATE: {
21 @@ -1092,6 +1095,9 @@
22        info->control.rates[2].idx = -1;
23        info->control.rates[3].idx = -1;
24        info->control.rates[4].idx = -1;
25 +
26 +      if(short_preamble)
27 +        info->flags |= IEEE80211_TX_RC_USE_SHORT_PREAMBLE;
28        break;
29      }
```

### B.1.4 RTS-CTS-Patch

```
1  diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.
     c linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c
2  --- linux-2.6.35-gentoo-r12-osg-tmp/net/mac80211/tx.c 2010-11-08
     10:35:57.412000004 +0100
3  +++ linux-2.6.35-gentoo-r12-osg/net/mac80211/tx.c 2010-11-15 14:09:34.373000008
     +0100
4  @@ -1089,6 +1089,10 @@
5      }
6
7      case IEEE80211_RADIOTAP_TX_FLAGS:
8  +      if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_CTS)
9  +        info->control.rates[0].flags |= IEEE80211_TX_RC_USE_CTS_PROTECT;
10 +      if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_RTS)
11 +        info->control.rates[0].flags |= IEEE80211_TX_RC_USE_RTS_CTS;
12        if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_NACK)
13          info->flags |= IEEE80211_TX_CTL_NO_ACK;
14        if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_TSF)
```

### B.1.5 TX-Zeitstempel für IWLAGN-Treiber

```
1  diff -Naur linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-tx.
     c linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/iwlwifi/iwl-agn-tx.c
2  --- linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-tx.c
     2010-11-06 17:36:14.535999966 +0100
3  +++ linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/iwlwifi/iwl-agn-tx.c
     2010-11-07 12:36:28.294000051 +0100
4  @@ -388,6 +388,9 @@
5      tx_cmd->timeout.pm_frame_timeout = 0;
6    }
7
8  + if (info->flags & IEEE80211_TX_CTL_TSF)
9  +   tx_flags |= TX_CMD_FLG_TSF_MSK;
10 +
11   tx_cmd->driver_txop = 0;
12   tx_cmd->tx_flags = tx_flags;
13   tx_cmd->next_frame_len = 0;
```

### B.1.6 Radiotap-Erweiterung für TX-Zeitstempel

```
1  diff -Naur linux-2.6.35-gentoo-r5-osg-tmp/include/net/ieee80211_radiotap.h linux
     -2.6.35-gentoo-r5-osg/include/net/ieee80211_radiotap.h
2  --- linux-2.6.35-gentoo-r5-osg-tmp/include/net/ieee80211_radiotap.h 2010-11-07
     12:39:14.437000049 +0100
3  +++ linux-2.6.35-gentoo-r5-osg/include/net/ieee80211_radiotap.h 2010-11-07
     12:05:28.613000026 +0100
4  @@ -264,6 +264,9 @@
5               * retries */
```

```
 #define IEEE80211_RADIOTAP_F_TX_CTS  0x0002  /* used cts 'protection' */
 #define IEEE80211_RADIOTAP_F_TX_RTS  0x0004  /* used rts/cts handshake */
+#define IEEE80211_RADIOTAP_F_TX_NACK 0x0008  /* don't expect ACK */
+#define IEEE80211_RADIOTAP_F_TX_SEQ  0x0010  /* don't insert sequence number */
+#define IEEE80211_RADIOTAP_F_TX_TSF  0x0020  /* insert TSF-Stamp */

 /* For IEEE80211_RADIOTAP_RATE_MCS */
 #define IEEE80211_RADIOTAP_RATE_MCS_40MHZ  0x01 /* 40 MHz channel width */
diff -Naur linux-2.6.35-gentoo-r5-osg-tmp/include/net/mac80211.h linux-2.6.35-
  gentoo-r5-osg/include/net/mac80211.h
--- linux-2.6.35-gentoo-r5-osg-tmp/include/net/mac80211.h 2010-11-07
  12:39:24.544000051 +0100
+++ linux-2.6.35-gentoo-r5-osg/include/net/mac80211.h 2010-11-07
  12:29:19.749000047 +0100
@@ -291,6 +291,7 @@
  * @IEEE80211_TX_STAT_ISR_TSF: TSF was captured during ISR.
  * @IEEE80211_TX_STAT_ISR_TSC: TSC was captured during ISR.
  * @IEEE80211_TX_STAT_ISR_HPET: HPET was captured during ISR.
+ * @IEEE80211_TX_CTL_TSF: insert TSF by firmware.
  */
 enum mac80211_tx_control_flags {
  IEEE80211_TX_CTL_REQ_TX_STATUS    = BIT(0),
@@ -322,6 +323,7 @@
  IEEE80211_TX_STAT_ISR_TSF   = BIT(27),
  IEEE80211_TX_STAT_ISR_TSC   = BIT(28),
  IEEE80211_TX_STAT_ISR_HPET    = BIT(29),
+ IEEE80211_TX_CTL_TSF      = BIT(30),
 };


diff -Naur linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/tx.c linux-2.6.35-gentoo-r5
  -osg/net/mac80211/tx.c
--- linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/tx.c  2010-11-07
  12:38:44.726000052 +0100
+++ linux-2.6.35-gentoo-r5-osg/net/mac80211/tx.c  2010-11-07 12:10:43.291000009
  +0100
@@ -1088,6 +1088,13 @@
      break;
    }

+   case IEEE80211_RADIOTAP_TX_FLAGS:
+     if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_NACK)
+       info->flags |= IEEE80211_TX_CTL_NO_ACK;
+     if(*(u16*)iterator.this_arg & IEEE80211_RADIOTAP_F_TX_TSF)
+       info->flags |= IEEE80211_TX_CTL_TSF;
+     break;
+
    case IEEE80211_RADIOTAP_DATA_RETRIES:
      info->control.rates[0].count = *iterator.this_arg;
      break;
@@ -1243,7 +1250,7 @@
    tx->flags |= IEEE80211_TX_UNICAST;
    if (unlikely(local->wifi_wme_noack_test))
      info->flags |= IEEE80211_TX_CTL_NO_ACK;
-   else
+   else if (!(info->flags & IEEE80211_TX_CTL_INJECTED))
      info->flags &= ~IEEE80211_TX_CTL_NO_ACK;
  }
```

### B.1.7 Erweiterung von MAC80211 und Radiotap um weitere Zeitstempel-Datenfelder

```
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/include/linux/
  skbuff.h linux-2.6.35-gentoo-r5-osg/include/linux/skbuff.h
--- linux-2.6.35-gentoo-r5-osg-tmp/include/linux/skbuff.h 2010-09-02
```

```
  14:37:50.963999997 +0200
+++ linux-2.6.35-gentoo-r5-osg/include/linux/skbuff.h 2010-11-06
  11:30:32.207000050 +0100
@@ -328,7 +328,7 @@
    * want to keep them across layers you have to do a skb_clone()
    * first. This is owned by whoever has the skb queued ATM.
    */
- char      cb[48] __aligned(8);
+ char      cb[64] __aligned(8);

  unsigned long   _skb_refdst;
 #ifdef CONFIG_XFRM
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/include/net/
  ieee80211_radiotap.h linux-2.6.35-gentoo-r5-osg/include/net/ieee80211_radiotap.h
--- linux-2.6.35-gentoo-r5-osg-tmp/include/net/ieee80211_radiotap.h 2010-11-06
  09:29:55.026000033 +0100
+++ linux-2.6.35-gentoo-r5-osg/include/net/ieee80211_radiotap.h 2010-11-06
  11:05:18.855000053 +0100
@@ -183,6 +183,17 @@
  *
  *     Number of unicast retries a transmitted frame used.
  *
+ * IEEE80211_RADIOTAP_ISR_TSF           __le32       microseconds
+ *
+ *     Lower 32 bit of TSF counter at execution of ISR
+ *
+ * IEEE80211_RADIOTAP_ISR_TSC           __le32       timer ticks
+ *
+ *     Timestamp at execution of ISR based on TSC
+ *
+ * IEEE80211_RADIOTAP_ISR_HPET          __le32       timer ticks
+ *
+ *     Timestamp at execution of ISR based on HPET
  */
 enum ieee80211_radiotap_type {
  IEEE80211_RADIOTAP_TSFT = 0,
@@ -204,6 +215,9 @@
  IEEE80211_RADIOTAP_RTS_RETRIES = 16,
  IEEE80211_RADIOTAP_DATA_RETRIES = 17,
  IEEE80211_RADIOTAP_RATE_MCS = 19,
+ IEEE80211_RADIOTAP_ISR_TSF = 20,
+ IEEE80211_RADIOTAP_ISR_TSC = 21,
+ IEEE80211_RADIOTAP_ISR_HPET = 22,

  /* valid in every it_present bitmap, even vendor namespaces */
  IEEE80211_RADIOTAP_RADIOTAP_NAMESPACE = 29,
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/include/net/
  mac80211.h linux-2.6.35-gentoo-r5-osg/include/net/mac80211.h
--- linux-2.6.35-gentoo-r5-osg-tmp/include/net/mac80211.h 2010-11-06
  10:04:43.356000110 +0100
+++ linux-2.6.35-gentoo-r5-osg/include/net/mac80211.h 2010-11-06
  12:38:14.766000069 +0100
@@ -288,6 +288,9 @@
  * @IEEE80211_TX_CTL_STBC: Enables Space-Time Block Coding (STBC) for this
  * frame and selects the maximum number of streams that it can use.
  * @IEEE80211_TX_CTL_RC_BYPASS: Dont use rate control on the frame.
+ * @IEEE80211_TX_STAT_ISR_TSF: TSF was captured during ISR.
+ * @IEEE80211_TX_STAT_ISR_TSC: TSC was captured during ISR.
+ * @IEEE80211_TX_STAT_ISR_HPET: HPET was captured during ISR.
  */
 enum mac80211_tx_control_flags {
  IEEE80211_TX_CTL_REQ_TX_STATUS    = BIT(0),
@@ -316,6 +319,9 @@
 #define IEEE80211_TX_CTL_STBC_SHIFT    23
  IEEE80211_TX_CTL_RC_BYPASS    = BIT(25),
  IEEE80211_TX_CTL_ANTENNA    = BIT(26),
```

```
+ IEEE80211_TX_STAT_ISR_TSF    = BIT(27),
+ IEEE80211_TX_STAT_ISR_TSC    = BIT(28),
+ IEEE80211_TX_STAT_ISR_HPET    = BIT(29),
 };


@@ -454,7 +460,9 @@
       u64 ampdu_ack_map;
       int ack_signal;
       u8 ampdu_len;
-      /* 7 bytes free */
+      u32 isr_tsf;
+      u32 isr_tsc;
+      u32 isr_hpet;
     } status;
     struct {
       struct ieee80211_tx_rate driver_rates[
@@ -539,6 +547,9 @@
   * @RX_FLAG_INTERNAL_CMTR: set internally after frame was reported
   * on cooked monitor to avoid double-reporting it for multiple
   * virtual interfaces
+ * @RX_FLAG_ISR_TSF: TSF was captured during ISR.
+ * @RX_FLAG_ISR_TSC: TSC was captured during ISR.
+ * @RX_FLAG_ISR_HPET: HPET was captured during ISR.
   */
 enum mac80211_rx_flags {
   RX_FLAG_MMIC_ERROR  = 1<<0,
@@ -553,6 +564,9 @@
   RX_FLAG_40MHZ    = 1<<10,
   RX_FLAG_SHORT_GI  = 1<<11,
   RX_FLAG_INTERNAL_CMTR = 1<<12,
+ RX_FLAG_ISR_TSF    = 1<<13,
+ RX_FLAG_ISR_TSC    = 1<<14,
+ RX_FLAG_ISR_HPET  = 1<<15,
 };


@@ -576,6 +590,9 @@
   */
 struct ieee80211_rx_status {
   u64 mactime;
+ u32 isr_tsf;
+ u32 isr_tsc;
+ u32 isr_hpet;
   enum ieee80211_band band;
   int freq;
   int signal;
@@ -1932,7 +1949,7 @@
   * The TX headroom reserved by mac80211 for its own tx_status functions.
   * This is enough for the radiotap header.
   */
-#define IEEE80211_TX_STATUS_HEADROOM 14
+#define IEEE80211_TX_STATUS_HEADROOM 32


diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/
  ieee80211_i.h linux-2.6.35-gentoo-r5-osg/net/mac80211/ieee80211_i.h
--- linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/ieee80211_i.h 2010-11-06
  09:29:55.028000031 +0100
+++ linux-2.6.35-gentoo-r5-osg/net/mac80211/ieee80211_i.h 2010-11-06
  12:35:45.196000067 +0100
@@ -1080,11 +1080,15 @@
   */
 struct ieee80211_tx_status_rtap_hdr {
   struct ieee80211_radiotap_header hdr;
```

```
- u8 rate;
- u8 padding_for_rate;
- __le16 tx_flags;
- u8 data_retries;
- u8 mcs; /*HT rates*/
+ union {
+   struct {
+     u8 rate;
+     u8 padding_for_rate;
+     __le16 tx_flags;
+     u8 data_retries;
+   };
+   u8 buf[24];
+ };
 } __attribute__ ((packed));


diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/rx.c
   linux-2.6.35-gentoo-r5-osg/net/mac80211/rx.c
--- linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/rx.c  2010-11-06
  09:29:55.029000030 +0100
+++ linux-2.6.35-gentoo-r5-osg/net/mac80211/rx.c  2010-11-06 11:33:59.601000048
  +0100
@@ -85,6 +85,14 @@
    len += 2;
  if (len & 1) /* padding for RX_FLAGS if necessary */
    len++;
+
+ len = ALIGN(len, 4);
+ if(status->flag & RX_FLAG_ISR_TSF)
+   len += 4;
+ if(status->flag & RX_FLAG_ISR_TSC)
+   len += 4;
+ if(status->flag & RX_FLAG_ISR_HPET)
+   len += 4;

  return len;
 }
@@ -200,6 +208,26 @@
      *pos |= IEEE80211_RADIOTAP_RATE_MCS_SHORT_GI;
    pos++;
  }
+
+ pos = PTR_ALIGN(pos, 4);
+ if(status->flag & RX_FLAG_ISR_TSF) {
+   put_unaligned_le32(status->isr_tsf, pos);
+   rthdr->it_present |=
+     cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_TSF);
+   pos += 4;
+ }
+ if(status->flag & RX_FLAG_ISR_TSC) {
+   put_unaligned_le32(status->isr_tsc, pos);
+   rthdr->it_present |=
+     cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_TSC);
+   pos += 4;
+ }
+ if(status->flag & RX_FLAG_ISR_HPET) {
+   put_unaligned_le32(status->isr_hpet, pos);
+   rthdr->it_present |=
+     cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_HPET);
+   pos += 4;
+ }
 }


diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/
```

```
  status.c linux-2.6.35-gentoo-r5-osg/net/mac80211/status.c
--- linux-2.6.35-gentoo-r5-osg-tmp/net/mac80211/status.c  2010-11-06
  09:29:55.030000030 +0100
+++ linux-2.6.35-gentoo-r5-osg/net/mac80211/status.c  2010-11-06
  12:39:36.359000080 +0100
@@ -173,6 +173,7 @@
  int retry_count = -1, i;
  int rates_idx = -1;
  bool send_to_cooked;
+ unsigned int pos, len;

  for (i = 0; i < IEEE80211_TX_MAX_RATES; i++) {
    /* the HW cannot have attempted that rate */
@@ -318,17 +319,33 @@

  /* send frame to monitor interfaces now */

- if (skb_headroom(skb) < sizeof(*rthdr)) {
+ len = sizeof(struct ieee80211_radiotap_header) + 5;
+ if (info->status.rates[0].flags & IEEE80211_TX_RC_MCS)
+   len++;
+ if (info->flags & IEEE80211_TX_STAT_ISR_TSF) {
+   len = ALIGN(len, 4);
+   len += 4;
+ }
+ if (info->flags & IEEE80211_TX_STAT_ISR_TSC) {
+   len = ALIGN(len, 4);
+   len += 4;
+ }
+ if (info->flags & IEEE80211_TX_STAT_ISR_HPET) {
+   len = ALIGN(len, 4);
+   len += 4;
+ }
+
+ if (skb_headroom(skb) < len) {
    printk(KERN_ERR "ieee80211_tx_status: headroom too small\n");
    dev_kfree_skb(skb);
    return;
  }

  rthdr = (struct ieee80211_tx_status_rtap_hdr *)
-       skb_push(skb, sizeof(*rthdr));
+       skb_push(skb, len);

- memset(rthdr, 0, sizeof(*rthdr));
- rthdr->hdr.it_len = cpu_to_le16(sizeof(*rthdr));
+ memset(rthdr, 0, len);
+ rthdr->hdr.it_len = cpu_to_le16(len);
  rthdr->hdr.it_present =
    cpu_to_le32((1 << IEEE80211_RADIOTAP_TX_FLAGS) |
          (1 << IEEE80211_RADIOTAP_DATA_RETRIES) |
@@ -352,15 +369,37 @@
      !(info->status.rates[0].flags & IEEE80211_TX_RC_MCS))
    rthdr->rate = sband->bitrates[
        info->status.rates[0].idx].bitrate / 5;
+
+ pos = 5;
+
  /* HT rates */
  if (info->status.rates[0].flags & IEEE80211_TX_RC_MCS) {
    rthdr->hdr.it_present |= cpu_to_le32(1 << IEEE80211_RADIOTAP_RATE_MCS);
    rthdr->rate = 0;
-   rthdr->mcs = info->status.rates[0].idx;
+   rthdr->buf[pos] = info->status.rates[0].idx;
    if (info->status.rates[0].flags & IEEE80211_TX_RC_40_MHZ_WIDTH)
      rthdr->tx_flags |= cpu_to_le16(IEEE80211_RADIOTAP_RATE_MCS_40MHZ);
```

```
250     if (info->status.rates[0].flags & IEEE80211_TX_RC_SHORT_GI)
251       rthdr->tx_flags |= cpu_to_le16(IEEE80211_RADIOTAP_RATE_MCS_SHORT_GI);
252 +   pos++;
253 + }
254 +
255 + /* Timestamps */
256 + pos = ALIGN(pos, 4);
257 + if (info->flags & IEEE80211_TX_STAT_ISR_TSF) {
258 +   rthdr->hdr.it_present |= cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_TSF);
259 +   *((__le32*)(&rthdr->buf[pos])) = cpu_to_le32(info->status.isr_tsf);
260 +   pos += 4;
261 + }
262 + if (info->flags & IEEE80211_TX_STAT_ISR_TSC) {
263 +   rthdr->hdr.it_present |= cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_TSC);
264 +   *((__le32*)(&rthdr->buf[pos])) = cpu_to_le32(info->status.isr_tsc);
265 +   pos += 4;
266 + }
267 + if (info->flags & IEEE80211_TX_STAT_ISR_HPET) {
268 +   rthdr->hdr.it_present |= cpu_to_le32(1 << IEEE80211_RADIOTAP_ISR_HPET);
269 +   *((__le32*)(&rthdr->buf[pos])) = cpu_to_le32(info->status.isr_hpet);
270 +   pos += 4;
271   }
272
273   /* for now report the total retry_count */
```

### B.1.8 Erfassung von Zeitstempeln im IWLANG-Treiber

```
1  diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/
     wireless/iwlwifi/iwl-agn-ict.c linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/
     iwlwifi/iwl-agn-ict.c
2  --- linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-ict.c
     2010-08-02 00:11:14.000000000 +0200
3  +++ linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/iwlwifi/iwl-agn-ict.c
     2010-11-06 16:19:28.621000007 +0100
4  @@ -32,6 +32,7 @@
5   #include <linux/sched.h>
6   #include <linux/gfp.h>
7   #include <net/mac80211.h>
8  +#include <asm/hpet.h>
9
10  #include "iwl-dev.h"
11  #include "iwl-core.h"
12 @@ -223,6 +224,9 @@
13    if (!priv)
14      return IRQ_NONE;
15
16 + priv->tsc  = get_cycles();
17 + priv->hpet = hpet_readl(HPET_COUNTER);
18 +
19    /* dram interrupt table not set yet,
20     * use legacy interrupt.
21     */
22 diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/
     wireless/iwlwifi/iwl-agn-lib.c linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/
     iwlwifi/iwl-agn-lib.c
23 --- linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-lib.c
     2010-09-02 14:40:18.588999969 +0200
24 +++ linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/iwlwifi/iwl-agn-lib.c
     2010-11-06 12:55:03.852000083 +0100
25 @@ -253,6 +253,11 @@
26        iwl_wake_queue(priv, txq_id);
27    }
28
29 + info->status.isr_tsc = priv->tsc;
30 + info->status.isr_hpet = priv->hpet;
31 + info->flags |= IEEE80211_TX_STAT_ISR_TSC;
```

```
32 + info->flags |= IEEE80211_TX_STAT_ISR_HPET;
33 +
34   iwlagn_txq_check_empty(priv, sta_id, tid, txq_id);
35
36   iwl_check_abort_status(priv, tx_resp->frame_count, status);
37 @@ -1055,6 +1060,11 @@
38    * this W/A doesnt propagate it to the mac80211 */
39   rx_status.flag |= RX_FLAG_TSFT;
40
41 + rx_status.isr_tsc = priv->tsc;
42 + rx_status.isr_hpet = priv->hpet;
43 + rx_status.flag |= RX_FLAG_ISR_TSC;
44 + rx_status.flag |= RX_FLAG_ISR_HPET;
45 +
46   priv->ucode_beacon_time = le32_to_cpu(phy_res->beacon_time_stamp);
47
48   /* Find max signal strength (dBm) among 3 antenna/receiver chains */
49 diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/
     wireless/iwlwifi/iwl-dev.h linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/
     iwlwifi/iwl-dev.h
50 --- linux-2.6.35-gentoo-r5-osg-tmp/drivers/net/wireless/iwlwifi/iwl-dev.h
     2010-08-02 00:11:14.000000000 +0200
51 +++ linux-2.6.35-gentoo-r5-osg/drivers/net/wireless/iwlwifi/iwl-dev.h 2010-11-06
     12:50:20.430000081 +0100
52 @@ -1378,6 +1378,9 @@
53   bool hw_ready;
54
55   struct iwl_event_log event_log;
56 +
57 + u32 tsc;
58 + u32 hpet;
59  }; /*iwl_priv */
60
61  static inline void iwl_txq_ctx_activate(struct iwl_priv *priv, int txq_id)
```

### B.1.9 Erfassung von Zeitstempeln im RTL8187-Treiber

```
1  diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/
     wireless/rtl818x/rtl8187_dev.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/
     rtl818x/rtl8187_dev.c
2  --- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/rtl818x/rtl8187_dev.c
     2010-08-02 00:11:14.000000000 +0200
3  +++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/rtl818x/rtl8187_dev.c
     2010-11-09 11:10:26.738000006 +0100
4  @@ -27,6 +27,7 @@
5   #include <linux/etherdevice.h>
6   #include <linux/eeprom_93cx6.h>
7   #include <net/mac80211.h>
8  +#include <asm/hpet.h>
9
10  #include "rtl8187.h"
11  #include "rtl8187_rtl8225.h"
12 @@ -192,6 +193,11 @@
13   struct ieee80211_tx_info *info = IEEE80211_SKB_CB(skb);
14   struct ieee80211_hw *hw = info->rate_driver_data[0];
15   struct rtl8187_priv *priv = hw->priv;
16 + cycles_t tsc;
17 + unsigned int hpet;
18 +
19 + tsc = get_cycles();
20 + hpet = hpet_readl(HPET_COUNTER);
21
22   skb_pull(skb, priv->is_rtl8187b ? sizeof(struct rtl8187b_tx_hdr) :
23              sizeof(struct rtl8187_tx_hdr));
24 @@ -216,6 +222,12 @@
25       info->flags |= IEEE80211_TX_STAT_ACK;
```

```
26      }
27    }
28  +
29  + info->status.isr_tsc = tsc;
30  + info->status.isr_hpet = hpet;
31  + info->flags |= IEEE80211_TX_STAT_ISR_TSC |
32  +                IEEE80211_TX_STAT_ISR_HPET;
33  +
34    if (priv->is_rtl8187b)
35      ieee80211_tx_status_irqsafe(hw, skb);
36    else {
37  @@ -323,6 +335,11 @@
38    int rate, signal;
39    u32 flags;
40    unsigned long f;
41  + cycles_t tsc;
42  + unsigned int hpet;
43  +
44  + tsc = get_cycles();
45  + hpet = hpet_readl(HPET_COUNTER);
46
47    spin_lock_irqsave(&priv->rx_queue.lock, f);
48    __skb_unlink(skb, &priv->rx_queue);
49  @@ -366,6 +383,11 @@
50      rx_status.mactime = le64_to_cpu(hdr->mac_time);
51    }
52
53  + rx_status.isr_tsc = tsc;
54  + rx_status.isr_hpet = hpet;
55  + rx_status.flag |= RX_FLAG_ISR_TSC |
56  +                   RX_FLAG_ISR_HPET;
57  +
58    rx_status.signal = signal;
59    priv->signal = signal;
60    rate = (flags >> 20) & 0xF;
```

### B.1.10 Erfassung von Zeitstempeln im ATK9K_HTC-Treiber

```
1  diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/
     wireless/ath/ath9k/hif_usb.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/
     ath/ath9k/hif_usb.c
2  --- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/hif_usb.c
     2010-08-02 00:11:14.000000000 +0200
3  +++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/hif_usb.c
     2010-11-09 20:01:46.720000012 +0100
4  @@ -14,6 +14,8 @@
5    * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
6    */
7
8  +#include <asm/hpet.h>
9  +
10  #include "htc.h"
11
12  #define ATH9K_FW_USB_DEV(devid, fw)          \
13 @@ -47,7 +49,7 @@
14
15    if (cmd) {
16      ath9k_htc_txcompletion_cb(cmd->hif_dev->htc_handle,
17 -            cmd->skb, 1);
18 +            cmd->skb, 1, 0, 0);
19      kfree(cmd);
20    }
21
22 @@ -109,6 +111,11 @@
23    struct tx_buf *tx_buf = (struct tx_buf *) urb->context;
24    struct hif_device_usb *hif_dev;
```

```
25    struct sk_buff *skb;
26  + cycles_t tsc;
27  + unsigned int hpet;
28  +
29  + tsc = get_cycles();
30  + hpet = hpet_readl(HPET_COUNTER);
31
32    if (!tx_buf || !tx_buf->hif_dev)
33      return;
34  @@ -144,7 +151,7 @@
35    /* Complete the queued SKBs. */
36    while ((skb = __skb_dequeue(&tx_buf->skb_queue)) != NULL) {
37      ath9k_htc_txcompletion_cb(hif_dev->htc_handle,
38  -           skb, 1);
39  +           skb, 1, tsc, hpet);
40      TX_STAT_INC(skb_completed);
41    }
42
43  @@ -329,7 +336,7 @@
44   };
45
46   static void ath9k_hif_usb_rx_stream(struct hif_device_usb *hif_dev,
47  -           struct sk_buff *skb)
48  +           struct sk_buff *skb, u32 tsc, u32 hpet)
49   {
50    struct sk_buff *nskb, *skb_pool[MAX_PKT_NUM_IN_TRANSFER];
51    int index = 0, i = 0, chk_idx, len = skb->len;
52  @@ -432,7 +439,7 @@
53   err:
54    for (i = 0; i < pool_index; i++) {
55      ath9k_htc_rx_msg(hif_dev->htc_handle, skb_pool[i],
56  -         skb_pool[i]->len, USB_WLAN_RX_PIPE);
57  +         skb_pool[i]->len, USB_WLAN_RX_PIPE, tsc, hpet);
58      RX_STAT_INC(skb_completed);
59    }
60   }
61  @@ -443,6 +450,11 @@
62    struct hif_device_usb *hif_dev = (struct hif_device_usb *)
63      usb_get_intfdata(usb_ifnum_to_if(urb->dev, 0));
64    int ret;
65  + cycles_t tsc;
66  + unsigned int hpet;
67  +
68  + tsc = get_cycles();
69  + hpet = hpet_readl(HPET_COUNTER);
70
71    if (!skb)
72      return;
73  @@ -464,7 +476,7 @@
74
75    if (likely(urb->actual_length != 0)) {
76      skb_put(skb, urb->actual_length);
77  -   ath9k_hif_usb_rx_stream(hif_dev, skb);
78  +   ath9k_hif_usb_rx_stream(hif_dev, skb, tsc, hpet);
79    }
80
81   resubmit:
82  @@ -514,7 +526,7 @@
83
84      /* Process the command first */
85      ath9k_htc_rx_msg(hif_dev->htc_handle, skb,
86  -           skb->len, USB_REG_IN_PIPE);
87  +           skb->len, USB_REG_IN_PIPE, 0, 0);
88
89
90      nskb = alloc_skb(MAX_REG_IN_BUF_SIZE, GFP_ATOMIC);
```

```
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc.h linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc.h
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc.h 2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc.h  2010-11-09 20:00:12.728000016 +0100
@@ -239,6 +239,8 @@
  bool in_process;
  struct sk_buff *skb;
  struct ath_htc_rx_status rxstatus;
+ u32 isr_tsc;
+ u32 isr_hpet;
  struct list_head list;
 };

@@ -410,11 +412,12 @@
 void ath9k_htc_swba(struct ath9k_htc_priv *priv, u8 beacon_pending);

 void ath9k_htc_rxep(void *priv, struct sk_buff *skb,
-       enum htc_endpoint_id ep_id);
+       enum htc_endpoint_id ep_id, u32 tsc, u32 hpet);
 void ath9k_htc_txep(void *priv, struct sk_buff *skb, enum htc_endpoint_id ep_id,
-       bool txok);
+       bool txok, u32 tsc, u32 hpet);
 void ath9k_htc_beaconep(void *drv_priv, struct sk_buff *skb,
-     enum htc_endpoint_id ep_id, bool txok);
+     enum htc_endpoint_id ep_id, bool txok,
+     u32 tsc, u32 hpet);

 void ath9k_htc_station_work(struct work_struct *work);
 void ath9k_htc_aggr_work(struct work_struct *work);
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_drv_beacon.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_drv_beacon.c
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_drv_beacon.c 2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_drv_beacon.c 2010-11-09 19:58:54.278000013 +0100
@@ -166,7 +166,8 @@
 }

 void ath9k_htc_beaconep(void *drv_priv, struct sk_buff *skb,
-     enum htc_endpoint_id ep_id, bool txok)
+     enum htc_endpoint_id ep_id, bool txok,
+     u32 tsc, u32 hpet)
 {
  dev_kfree_skb_any(skb);
 }
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_drv_init.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_drv_init.c
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_drv_init.c 2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_drv_init.c 2010-11-09 19:52:26.996000013 +0100
@@ -126,7 +126,9 @@
          void (*tx) (void *,
                      struct sk_buff *,
                      enum htc_endpoint_id,
-                     bool txok),
+                     bool txok,
+                     u32 tsc,
+                     u32 hpet),
          enum htc_endpoint_id *ep_id)
 {
```

```
145   struct htc_service_connreq req;
146 diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/
      wireless/ath/ath9k/htc_drv_txrx.c linux-2.6.35-gentoo-r12-osg/drivers/net/
      wireless/ath/ath9k/htc_drv_txrx.c
147 --- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_drv_txrx.c
      2010-11-08 10:15:15.605000003 +0100
148 +++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_drv_txrx.c
      2010-11-09 19:57:39.460000013 +0100
149 @@ -14,6 +14,8 @@
150   * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
151   */
152
153 +#include <asm/hpet.h>
154 +
155  #include "htc.h"
156
157  /******/
158 @@ -194,8 +196,6 @@
159      fc = hdr->frame_control;
160      tx_info = IEEE80211_SKB_CB(skb);
161
162 -    memset(&tx_info->status, 0, sizeof(tx_info->status));
163 -
164      rcu_read_lock();
165
166      sta = ieee80211_find_sta(priv->vif, hdr->addr1);
167 @@ -246,7 +246,8 @@
168  }
169
170  void ath9k_htc_txep(void *drv_priv, struct sk_buff *skb,
171 -        enum htc_endpoint_id ep_id, bool txok)
172 +        enum htc_endpoint_id ep_id, bool txok,
173 +        u32 tsc, u32 hpet)
174  {
175    struct ath9k_htc_priv *priv = (struct ath9k_htc_priv *) drv_priv;
176    struct ath_common *common = ath9k_hw_common(priv->ah);
177 @@ -269,7 +270,13 @@
178      return;
179    }
180
181    tx_info = IEEE80211_SKB_CB(skb);
182 + memset(&tx_info->status, 0, sizeof(tx_info->status));
183 +
184 + tx_info->status.isr_tsc = tsc;
185 + tx_info->status.isr_hpet = hpet;
186 + tx_info->flags |= IEEE80211_TX_STAT_ISR_TSC |
187 +                   IEEE80211_TX_STAT_ISR_HPET;
188
189    if (txok)
190      tx_info->flags |= IEEE80211_TX_STAT_ACK;
191 @@ -590,6 +597,11 @@
192    struct sk_buff *skb;
193    unsigned long flags;
194    struct ieee80211_hdr *hdr;
195 + cycles_t tsc;
196 + unsigned int hpet;
197 +
198 + tsc = get_cycles();
199 + hpet = hpet_readl(HPET_COUNTER);
200
201    do {
202      spin_lock_irqsave(&priv->rx.rxbuflock, flags);
203 @@ -613,6 +625,11 @@
204        goto requeue;
205      }
206
```

```
+   rx_status.isr_tsc = rxbuf->isr_tsc;
+   rx_status.isr_hpet = rxbuf->isr_hpet;
+   rx_status.flag |= RX_FLAG_ISR_TSC |
+                     RX_FLAG_ISR_HPET;
+
    memcpy(IEEE80211_SKB_RXCB(rxbuf->skb), &rx_status,
           sizeof(struct ieee80211_rx_status));
    skb = rxbuf->skb;
@@ -637,7 +654,7 @@
 }

 void ath9k_htc_rxep(void *drv_priv, struct sk_buff *skb,
-       enum htc_endpoint_id ep_id)
+       enum htc_endpoint_id ep_id, u32 tsc, u32 hpet)
 {
   struct ath9k_htc_priv *priv = (struct ath9k_htc_priv *)drv_priv;
   struct ath_hw *ah = priv->ah;
@@ -662,6 +679,8 @@
   spin_lock(&priv->rx.rxbuflock);
   rxbuf->skb = skb;
   rxbuf->in_process = true;
+ rxbuf->isr_tsc = tsc;
+ rxbuf->isr_hpet = hpet;
   spin_unlock(&priv->rx.rxbuflock);

   tasklet_schedule(&priv->rx_tasklet);
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/
  wireless/ath/ath9k/htc_hst.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/
  ath/ath9k/htc_hst.c
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_hst.c
  2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_hst.c
  2010-11-09 19:48:21.969000015 +0100
@@ -319,7 +319,8 @@
 }

 void ath9k_htc_txcompletion_cb(struct htc_target *htc_handle,
-            struct sk_buff *skb, bool txok)
+            struct sk_buff *skb, bool txok,
+            u32 tsc, u32 hpet)
 {
   struct htc_endpoint *endpoint;
   struct htc_frame_hdr *htc_hdr = NULL;
@@ -344,7 +345,7 @@
     if (endpoint->ep_callbacks.tx) {
       endpoint->ep_callbacks.tx(endpoint->ep_callbacks.priv,
               skb, htc_hdr->endpoint_id,
-              txok);
+              txok, tsc, hpet);
     }
   }

@@ -365,7 +366,8 @@
  * endpoint RX handlers, which have to free the SKB.
  */
 void ath9k_htc_rx_msg(struct htc_target *htc_handle,
-         struct sk_buff *skb, u32 len, u8 pipe_id)
+         struct sk_buff *skb, u32 len, u8 pipe_id,
+         u32 tsc, u32 hpet)
 {
   struct htc_frame_hdr *htc_hdr;
   enum htc_endpoint_id epid;
@@ -422,7 +424,7 @@
     endpoint = &htc_handle->endpoint[epid];
     if (endpoint->ep_callbacks.rx)
       endpoint->ep_callbacks.rx(endpoint->ep_callbacks.priv,
```

```
-             skb, epid);
+             skb, epid, tsc, hpet);
  }
 }

diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_hst.h linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_hst.h
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/ath/ath9k/htc_hst.h 2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/ath/ath9k/htc_hst.h 2010-11-09 20:01:19.606000013 +0100
@@ -93,8 +93,8 @@

 struct htc_ep_callbacks {
  void *priv;
- void (*tx) (void *, struct sk_buff *, enum htc_endpoint_id, bool txok);
- void (*rx) (void *, struct sk_buff *, enum htc_endpoint_id);
+ void (*tx) (void *, struct sk_buff *, enum htc_endpoint_id, bool txok, u32 tsc, u32 hpet);
+ void (*rx) (void *, struct sk_buff *, enum htc_endpoint_id, u32 tsc, u32 hpet);
 };

 #define HTC_TX_QUEUE_SIZE 256
@@ -230,9 +230,11 @@
 void htc_start(struct htc_target *target);

 void ath9k_htc_rx_msg(struct htc_target *htc_handle,
-         struct sk_buff *skb, u32 len, u8 pipe_id);
+         struct sk_buff *skb, u32 len, u8 pipe_id,
+         u32 tsc, u32 hpet);
 void ath9k_htc_txcompletion_cb(struct htc_target *htc_handle,
-            struct sk_buff *skb, bool txok);
+            struct sk_buff *skb, bool txok,
+            u32 tsc, u32 hpet);

 struct htc_target *ath9k_htc_hw_alloc(void *hif_handle,
               struct ath9k_htc_hif *hif,
```

### B.1.11 Zugriff auf HPET-Register durch Kernel-Module

```
diff -Naur linux-2.6.35-gentoo-r5-osg-tmp/arch/x86/kernel/hpet.c linux-2.6.35-gentoo-r5-osg/arch/x86/kernel/hpet.c
--- linux-2.6.35-gentoo-r5-osg-tmp/arch/x86/kernel/hpet.c 2010-08-02 00:11:14.000000000 +0200
+++ linux-2.6.35-gentoo-r5-osg/arch/x86/kernel/hpet.c 2010-11-06 10:42:38.221000061 +0100
@@ -56,6 +56,7 @@
 {
  return readl(hpet_virt_address + a);
 }
+EXPORT_SYMBOL(hpet_readl);

 static inline void hpet_writel(unsigned int d, unsigned int a)
 {
```

### B.1.12 Aktivierung der Radiotap-MACTIME im IWLANG-Treiber

```
diff -Naur --exclude-from=exclude linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-lib.c linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/iwlwifi/iwl-agn-lib.c
--- linux-2.6.35-gentoo-r12-osg-tmp/drivers/net/wireless/iwlwifi/iwl-agn-lib.c 2010-11-08 10:35:46.911000003 +0100
+++ linux-2.6.35-gentoo-r12-osg/drivers/net/wireless/iwlwifi/iwl-agn-lib.c 2010-11-16 11:53:53.802000029 +0100
```

```
@@ -1058,7 +1058,7 @@

  /* TSF isn't reliable. In order to allow smooth user experience,
   * this W/A doesn't propagate it to the mac80211 */
- /*rx_status.flag |= RX_FLAG_TSFT;*/
+ rx_status.flag |= RX_FLAG_TSFT;

  rx_status.isr_tsc = priv->tscjiffies;
  rx_status.isr_hpet = priv->hpetjiffies;
```

## B.2 Wireshark-Patches

### B.2.1 Radaiotap-Datenfelder

```
diff -Naur wireshark-1.2.12/epan/dissectors/packet-radiotap.c wireshark-1.2.12-osg/epan/dissectors/packet-radiotap.c
--- wireshark-1.2.12/epan/dissectors/packet-radiotap.c  2010-10-11 20:29:38.000000000 +0200
+++ wireshark-1.2.12-osg/epan/dissectors/packet-radiotap.c  2010-11-07 10:28:01.759000017 +0100
@@ -89,7 +89,14 @@
     IEEE80211_RADIOTAP_DB_ANTSIGNAL = 12,
     IEEE80211_RADIOTAP_DB_ANTNOISE = 13,
     IEEE80211_RADIOTAP_RX_FLAGS = 14,
+    IEEE80211_RADIOTAP_TX_FLAGS = 15,
+    IEEE80211_RADIOTAP_RTS_RETRIES = 16,
+    IEEE80211_RADIOTAP_DATA_RETRIES = 17,
     IEEE80211_RADIOTAP_XCHANNEL = 18,
+    IEEE80211_RADIOTAP_MCS = 19,
+    IEEE80211_RADIOTAP_ISR_TSF = 20,
+    IEEE80211_RADIOTAP_ISR_TSC = 21,
+    IEEE80211_RADIOTAP_ISR_HPET = 22,
     IEEE80211_RADIOTAP_EXT = 31
 };

@@ -200,6 +207,9 @@
 static int hf_radiotap_channel_flags_quarter = -1;
 static int hf_radiotap_rxflags = -1;
 static int hf_radiotap_rxflags_badplcp = -1;
+static int hf_radiotap_txflags = -1;
+static int hf_radiotap_rtsretries = -1;
+static int hf_radiotap_dataretries = -1;
 static int hf_radiotap_xchannel = -1;
 static int hf_radiotap_xchannel_frequency = -1;
 static int hf_radiotap_xchannel_flags = -1;
@@ -221,6 +231,12 @@
 #if 0
 static int hf_radiotap_xchannel_maxpower = -1;
 #endif
+static int hf_radiotap_mcs = -1;
+static int hf_radiotap_mcs_ht40 = -1;
+static int hf_radiotap_mcs_shortgi = -1;
+static int hf_radiotap_isr_tsf = -1;
+static int hf_radiotap_isr_tsc = -1;
+static int hf_radiotap_isr_hpet = -1;
 static int hf_radiotap_fhss_hopset = -1;
 static int hf_radiotap_fhss_pattern = -1;
 static int hf_radiotap_datarate = -1;
@@ -250,7 +266,14 @@
 static int hf_radiotap_present_db_antnoise = -1;
 static int hf_radiotap_present_hdrfcs = -1;
 static int hf_radiotap_present_rxflags = -1;
+static int hf_radiotap_present_txflags = -1;
+static int hf_radiotap_present_rtsretries = -1;
```

```
48 +static int hf_radiotap_present_dataretries = -1;
49  static int hf_radiotap_present_xchannel = -1;
50 +static int hf_radiotap_present_mcs = -1;
51 +static int hf_radiotap_present_isr_tsf = -1;
52 +static int hf_radiotap_present_isr_tsc = -1;
53 +static int hf_radiotap_present_isr_hpet = -1;
54  static int hf_radiotap_present_ext = -1;
55
56  /* "present.flags" flags */
57 @@ -274,6 +297,7 @@
58  static gint ett_radiotap_rxflags = -1;
59  static gint ett_radiotap_channel_flags = -1;
60  static gint ett_radiotap_xchannel_flags = -1;
61 +static gint ett_radiotap_mcs_flags = -1;
62
63  static dissector_handle_t ieee80211_handle;
64  static dissector_handle_t ieee80211_datapad_handle;
65 @@ -445,7 +469,14 @@
66  #define RADIOTAP_MASK_DB_ANTSIGNAL          0x00001000
67  #define RADIOTAP_MASK_DB_ANTNOISE           0x00002000
68  #define RADIOTAP_MASK_RX_FLAGS              0x00004000
69 +#define RADIOTAP_MASK_TX_FLAGS              0x00008000
70 +#define RADIOTAP_MASK_RTS_RETRIES           0x00010000
71 +#define RADIOTAP_MASK_DATA_RETRIES          0x00020000
72  #define RADIOTAP_MASK_XCHANNEL              0x00040000
73 +#define RADIOTAP_MASK_MCS                   0x00080000
74 +#define RADIOTAP_MASK_ISR_TSF               0x00100000
75 +#define RADIOTAP_MASK_ISR_TSC               0x00200000
76 +#define RADIOTAP_MASK_ISR_HPET              0x00400000
77  #define RADIOTAP_MASK_EXT                   0x80000000
78
79      /* Boolean 'present' flags */
80 @@ -524,6 +555,21 @@
81   FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_RX_FLAGS,
82   "Specifies␣if␣the␣RX␣flags␣field␣is␣present", HFILL } },
83
84 +    { &hf_radiotap_present_txflags,
85 +      { "TX␣flags", "radiotap.present.txflags",
86 + FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_TX_FLAGS,
87 + "Specifies␣if␣the␣TX␣flags␣field␣is␣present", HFILL } },
88 +
89 +    { &hf_radiotap_present_rtsretries,
90 +      { "RTS␣retries", "radiotap.present.rts_retries",
91 + FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_RTS_RETRIES,
92 + "Specifies␣if␣the␣RTS␣retries␣field␣is␣present", HFILL } },
93 +
94 +    { &hf_radiotap_present_dataretries,
95 +      { "DATA␣retries", "radiotap.present.data_retries",
96 + FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_DATA_RETRIES,
97 + "Specifies␣if␣the␣DATA␣retries␣field␣is␣present", HFILL } },
98 +
99      { &hf_radiotap_present_hdrfcs,
100       { "FCS␣in␣header", "radiotap.present.fcs",
101   FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_RX_FLAGS,
102 @@ -534,6 +580,26 @@
103   FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_XCHANNEL,
104   "Specifies␣if␣the␣extended␣channel␣info␣field␣is␣present", HFILL } },
105
106 +    { &hf_radiotap_present_mcs,
107 +      { "MCS", "radiotap.present.mcs",
108 +        FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_MCS,
109 +        "Specifies␣if␣the␣MCS␣field␣is␣present", HFILL } },
110 +
111 +    { &hf_radiotap_present_isr_tsf,
112 +      { "ISR␣TSF", "radiotap.present.isrtsf",
113 +        FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_ISR_TSF,
```

```
114 +        "Specifies␣if␣TSF␣at␣entrance␣of␣interrupt␣service␣routine␣is␣present",
    HFILL } },
115 +
116 +    { &hf_radiotap_present_isr_tsc,
117 +      { "ISR␣TSC", "radiotap.present.isrtsc",
118 +        FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_ISR_TSC,
119 +        "Specifies␣if␣TSC␣at␣entrance␣of␣interrupt␣service␣routine␣is␣present",
    HFILL } },
120 +
121 +    { &hf_radiotap_present_isr_hpet,
122 +      { "ISR␣HPET", "radiotap.present.isrhpet",
123 +        FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_ISR_HPET,
124 +        "Specifies␣if␣HPET␣at␣entrance␣of␣interrupt␣service␣routine␣is␣present",
    HFILL } },
125 +
126      { &hf_radiotap_present_ext,
127        { "Ext", "radiotap.present.ext",
128    FT_BOOLEAN, 32, NULL, RADIOTAP_MASK_EXT,
129  @@ -661,6 +727,18 @@
130     FT_BOOLEAN, 24, NULL, IEEE80211_RADIOTAP_F_RX_BADPLCP,
131     "Frame␣with␣bad␣PLCP", HFILL } },
132
133 +    { &hf_radiotap_txflags,
134 +      { "TX␣flags", "radiotap.txflags",
135 + FT_UINT16, BASE_HEX, NULL, 0x0, "", HFILL } },
136 +
137 +    { &hf_radiotap_rtsretries,
138 +      { "RTS␣retries", "radiotap.rts_retries",
139 + FT_UINT8, BASE_DEC, NULL, 0x0, "", HFILL } },
140 +
141 +    { &hf_radiotap_dataretries,
142 +      { "DATA␣retries", "radiotap.data_retries",
143 + FT_UINT8, BASE_DEC, NULL, 0x0, "", HFILL } },
144 +
145      { &hf_radiotap_xchannel,
146        { "Channel␣number", "radiotap.xchannel",
147    FT_UINT32, BASE_DEC, NULL, 0x0, "", HFILL } },
148  @@ -782,6 +860,36 @@
149    FT_BOOLEAN, BASE_NONE, NULL, 0x0,
150    "Specifies␣if␣this␣frame␣has␣a␣bad␣frame␣check␣sequence", HFILL } },
151
152 +    { &hf_radiotap_mcs,
153 +      { "MCS␣index", "radiotap.mcs",
154 +        FT_UINT8, BASE_DEC, NULL, 0x0,
155 +        "Index␣of␣MCS␣rate", HFILL } },
156 +
157 +    { &hf_radiotap_mcs_ht40,
158 +      { "40MHz", "radiotap.mcs.ht40",
159 +        FT_BOOLEAN, 8, NULL, 0x01,
160 +        "40MHz␣channel␣width", HFILL } },
161 +
162 +    { &hf_radiotap_mcs_shortgi,
163 +      { "Short␣GI", "radiotap.mcs.shortgi",
164 +        FT_BOOLEAN, 8, NULL, 0x02,
165 +        "Short␣guard␣interval", HFILL } },
166 +
167 +    { &hf_radiotap_isr_tsf,
168 +      { "ISR␣TSF␣timestamp", "radiotap.isrtsf",
169 +        FT_UINT32, BASE_DEC, NULL, 0x0,
170 +        "Lower␣32bit␣of␣TSF␣at␣entrance␣in␣ISR", HFILL } },
171 +
172 +    { &hf_radiotap_isr_tsc,
173 +      { "ISR␣TSC␣timestamp", "radiotap.isrtsc",
174 +        FT_UINT32, BASE_DEC, NULL, 0x0,
175 +        "Lower␣32bit␣of␣TSC␣at␣entrance␣in␣ISR", HFILL } },
176 +
```

```
177 +    { &hf_radiotap_isr_hpet,
178 +      { "ISR␣HPET␣timestamp", "radiotap.isrhpet",
179 +        FT_UINT32, BASE_DEC, NULL, 0x0,
180 +        "HPET␣at␣entrance␣in␣ISR", HFILL } },
181 +
182    };
183    static gint *ett[] = {
184      &ett_radiotap,
185 @@ -789,7 +897,8 @@
186      &ett_radiotap_flags,
187      &ett_radiotap_rxflags,
188      &ett_radiotap_channel_flags,
189 -    &ett_radiotap_xchannel_flags
190 +    &ett_radiotap_xchannel_flags,
191 +    &ett_radiotap_mcs_flags
192    };
193    module_t *radiotap_module;
194
195 @@ -920,8 +1029,22 @@
196     proto_tree_add_item(present_tree, hf_radiotap_present_rxflags,
197       tvb, 4, 4, TRUE);
198   }
199 + proto_tree_add_item(present_tree, hf_radiotap_present_txflags,
200 +     tvb, 4, 4, TRUE);
201 + proto_tree_add_item(present_tree, hf_radiotap_present_rtsretries,
202 +     tvb, 4, 4, TRUE);
203 + proto_tree_add_item(present_tree, hf_radiotap_present_dataretries,
204 +     tvb, 4, 4, TRUE);
205   proto_tree_add_item(present_tree, hf_radiotap_present_xchannel,
206       tvb, 4, 4, TRUE);
207 + proto_tree_add_item(present_tree, hf_radiotap_present_mcs,
208 +     tvb, 4, 4, TRUE);
209 + proto_tree_add_item(present_tree, hf_radiotap_present_isr_tsf,
210 +     tvb, 4, 4, TRUE);
211 + proto_tree_add_item(present_tree, hf_radiotap_present_isr_tsc,
212 +     tvb, 4, 4, TRUE);
213 + proto_tree_add_item(present_tree, hf_radiotap_present_isr_hpet,
214 +     tvb, 4, 4, TRUE);
215   proto_tree_add_item(present_tree, hf_radiotap_present_ext,
216       tvb, 4, 4, TRUE);
217     }
218 @@ -1285,6 +1408,105 @@
219                 length_remaining-=2;
220             }
221       break;
222 + case IEEE80211_RADIOTAP_TX_FLAGS:
223 +     {
224 +         proto_item *it;
225 +
226 +                 align_offset = ALIGN_OFFSET(offset, 2);
227 +                 offset += align_offset;
228 +                 length_remaining -= align_offset;
229 +                 if (length_remaining < 2)
230 +                     break;
231 +                 if (tree) {
232 +                     flags = tvb_get_letohs(tvb, offset);
233 +                     it = proto_tree_add_uint(radiotap_tree, hf_radiotap_txflags,
234 +                             tvb, offset, 2, flags);
235 +                 }
236 +                 offset+=2;
237 +                 length_remaining-=2;
238 +             }
239 +     break;
240 + case IEEE80211_RADIOTAP_RTS_RETRIES:
241 +     if (length_remaining < 1)
242 +   break;
```

```
243 +     if (tree) {
244 +   proto_tree_add_uint(radiotap_tree, hf_radiotap_rtsretries,
245 +         tvb, offset, 1, tvb_get_guint8(tvb, offset));
246 +     }
247 +     offset++;
248 +     length_remaining--;
249 +     break;
250 + case IEEE80211_RADIOTAP_DATA_RETRIES:
251 +     if (length_remaining < 1)
252 +   break;
253 +     if (tree) {
254 +   proto_tree_add_uint(radiotap_tree, hf_radiotap_dataretries,
255 +         tvb, offset, 1, tvb_get_guint8(tvb, offset));
256 +     }
257 +     offset++;
258 +     length_remaining--;
259 +     break;
260 + case IEEE80211_RADIOTAP_MCS: {
261 +     proto_item *it;
262 +     proto_tree *flags_tree;
263 +
264 +     if (length_remaining < 2)
265 +   break;
266 +
267 +     if (tree) {
268 +   it = proto_tree_add_uint(radiotap_tree, hf_radiotap_mcs,
269 +     tvb, offset, 1, tvb_get_guint8(tvb, offset));
270 +   flags_tree = proto_item_add_subtree(it, ett_radiotap_mcs_flags);
271 +   flags = tvb_get_letohs(tvb, offset+1);
272 +   proto_tree_add_boolean(flags_tree, hf_radiotap_mcs_ht40,
273 +     tvb, offset+1, 1, flags);
274 +   proto_tree_add_boolean(flags_tree, hf_radiotap_mcs_shortgi,
275 +     tvb, offset+1, 1, flags);
276 +     }
277 +
278 +     offset += 2;
279 +     length_remaining -= 2;
280 + }
281 +     break;
282 + case IEEE80211_RADIOTAP_ISR_TSF:
283 +     align_offset = ALIGN_OFFSET(offset, 4);
284 +     offset += align_offset;
285 +     length_remaining -= align_offset;
286 +     if (length_remaining < 4)
287 +   break;
288 +     if (tree) {
289 +   proto_tree_add_uint(radiotap_tree, hf_radiotap_isr_tsf,
290 +       tvb, offset, 4, tvb_get_letohl(tvb, offset));
291 +     }
292 +     offset+=4;
293 +     length_remaining-=4;
294 +     break;
295 + case IEEE80211_RADIOTAP_ISR_TSC:
296 +     align_offset = ALIGN_OFFSET(offset, 4);
297 +     offset += align_offset;
298 +     length_remaining -= align_offset;
299 +     if (length_remaining < 4)
300 +   break;
301 +     if (tree) {
302 +   proto_tree_add_uint(radiotap_tree, hf_radiotap_isr_tsc,
303 +       tvb, offset, 4, tvb_get_letohl(tvb, offset));
304 +     }
305 +     offset+=4;
306 +     length_remaining-=4;
307 +     break;
308 + case IEEE80211_RADIOTAP_ISR_HPET:
```

```
309 +     align_offset = ALIGN_OFFSET(offset, 4);
310 +     offset += align_offset;
311 +     length_remaining -= align_offset;
312 +     if (length_remaining < 4)
313 +   break;
314 +     if (tree) {
315 +   proto_tree_add_uint(radiotap_tree, hf_radiotap_isr_hpet,
316 +       tvb, offset, 4, tvb_get_letohl(tvb, offset));
317 +     }
318 +     offset+=4;
319 +     length_remaining-=4;
320 +     break;
321   default:
322       /*
323        * This indicates a field whose size we do not
```

### B.2.2 Zugriff auf TX-Zeitstempel

```
1  diff -Naur wireshark-1.2.12-osg/epan/dissectors/packet-ieee80211.c wireshark
     -1.2.12-osg2/epan/dissectors/packet-ieee80211.c
2  --- wireshark-1.2.12-osg/epan/dissectors/packet-ieee80211.c 2010-10-11
     20:29:36.000000000 +0200
3  +++ wireshark-1.2.12-osg2/epan/dissectors/packet-ieee80211.c  2011-02-19
     23:45:23.689000098 +0100
4  @@ -6950,11 +6950,12 @@
5         /* Davide Schiera -------------------------------------------------------
              */
6   #endif
7
8  -      /*
9  -       * No-data frames don't have a body.
10 -       */
11        if (DATA_FRAME_IS_NULL(frame_type_subtype))
12 +      {
13 +        next_tvb = tvb_new_subset (tvb, hdr_len, len, reported_len);
14 +        call_dissector(data_handle, next_tvb, pinfo, tree);
15          return;
16 +      }
17
18        if (!wlan_subdissector) {
19          guint fnum = 0;
```

# Anhang C

# Quelltexte

## C.1 Das Programm nullinject

nullinject.c

```
1  #include <stdlib.h>
2  #include <stdint.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <ctype.h>
6  #include <unistd.h>
7  #include <error.h>
8  #include <endian.h>
9  #include <sys/ioctl.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <net/if.h>
13 #include <net/if_arp.h>
14 #include <pcap/pcap.h>
15 #include "ieee80211_radiotap.h"
16
17
18 #define ALIGN(x, a)     (((x) + (a - 1)) & ~(a - 1))
19
20
21 static char *opt_interface      = NULL;
22 static unsigned int opt_count   = 0;
23 static char *opt_bss            = NULL;
24 static char *opt_src            = NULL;
25 static char *opt_dst            = NULL;
26 static unsigned int opt_bitrate = 0;
27 static int opt_mcs              = -1;
28 static int opt_antenna          = -1;
29 static unsigned int opt_time    = 0;
30 static unsigned int opt_rts     = 0;
31 static unsigned int opt_short   = 0;
32 static unsigned int opt_noack   = 0;
33 static unsigned int opt_tsf     = 0;
34
35 static uint8_t conf_bss[6];
36 static uint8_t conf_src[6];
37 static uint8_t conf_dst[6];
38
39 static const unsigned int bitrates[] = { 2, 4, 11, 22, 12, 18, 24, 36, 48, 72, 96,
     108 };
```

```
40
41
42
43  void usage(const char *prog)
44  {
45    printf("Usage: %s\n", prog);
46    printf("  -i <interface>\n");
47    printf(" [-n <number of packets>]\n");
48    printf(" [-b <bssid>]\n");
49    printf(" [-s <src mac>]\n");
50    printf("  -d <dst mac>\n");
51    printf(" [-r <bitrate>]\n");
52    printf(" [-m <mcs index>]\n");
53    printf(" [-a <antenna>]\n");
54    printf(" [-t <delay>]\n");
55    printf(" [-p]\t\t\t\t# use RTS-CTS\n");
56    printf(" [-o]\t\t\t\t# use short preamble\n");
57    printf(" [-k]\t\t\t\t# don't wait for ACK\n");
58    printf(" [-x]\t\t\t\t# insert TSF timestamp\n");
59    printf("\n");
60    printf("Bitrates:\n");
61    printf("  0 => let interface decide\n");
62    printf("  1 =>  1   MBit/s\t");
63    printf("  2 =>  2   MBit/s\n");
64    printf("  3 =>  5.5 MBit/s\t");
65    printf("  4 => 11   MBit/s\n");
66    printf("  5 =>  6   MBit/s\t");
67    printf("  6 =>  9   MBit/s\n");
68    printf("  7 => 12   MBit/s\t");
69    printf("  8 => 18   MBit/s\n");
70    printf("  9 => 24   MBit/s\t");
71    printf(" 10 => 36   MBit/s\n");
72    printf(" 11 => 48   MBit/s\t");
73    printf(" 12 => 54   MBit/s\n");
74  }
75
76
77  void opts(int argc, char **argv)
78  {
79    int ret;
80
81
82    opterr = 0;
83    while((ret = getopt(argc, argv, "i:n:b:s:d:r:m:a:t:pokx")) != -1)
84    {
85      switch(ret)
86      {
87      case 'i': opt_interface = optarg;       break;
88      case 'n': opt_count     = atoi(optarg); break;
89      case 'b': opt_bss       = optarg;       break;
90      case 's': opt_src       = optarg;       break;
91      case 'd': opt_dst       = optarg;       break;
92      case 'r': opt_bitrate   = atoi(optarg); break;
93      case 'm': opt_mcs       = atoi(optarg); break;
94      case 'a': opt_antenna   = atoi(optarg); break;
95      case 't': opt_time      = atoi(optarg); break;
96      case 'p': opt_rts       = 1;            break;
97      case 'o': opt_short     = 1;            break;
98      case 'k': opt_noack     = 1;            break;
99      case 'x': opt_tsf       = 1;            break;
100     }
101   }
102 }
103
104
105 static int str2mac(const char *str, uint8_t *mac)
```

```
106 {
107   char *p;
108   unsigned int i;
109 
110   if(strlen(str) != 17)
111     return -1;
112 
113   for(i = 0; i < 6; i++)
114     if(!isxdigit(str[0]) || !isxdigit(str[1]))
115       return -1;
116   for(i = 0; i < 6; i++)
117   {
118     mac[i] = strtol(str + 3 * i, &p, 16);
119     if(str + 3 * i == p)
120       return -1;
121   }
122 
123   return 0;
124 }
125 
126 
127 static int mkradiotap(uint8_t *ptr, unsigned int rate, int mcs, int antenna)
128 {
129   uint32_t flags = 0;
130   unsigned int pos;
131 
132 
133 
134   ptr[0] = PKTHDR_RADIOTAP_VERSION;
135   ptr[1] = 0x00;
136   *(uint16_t*)(ptr + 2) = htole16(0);
137 
138   pos = 8;
139 
140 
141   // Flags
142   flags |= 1 << IEEE80211_RADIOTAP_FLAGS;
143   *(uint8_t*)(ptr + pos) = IEEE80211_RADIOTAP_F_FCS;
144   if(opt_short)
145     *(uint8_t*)(ptr + pos) |= IEEE80211_RADIOTAP_F_SHORTPRE;
146   pos += 1;
147 
148   // Datenrate
149   if(rate && mcs < 0)
150   {
151     flags |= 1 << IEEE80211_RADIOTAP_RATE;
152     *(uint8_t*)(ptr + pos) = bitrates[rate - 1];
153     pos += 1;
154   }
155 
156   // Antenne
157   if(antenna >= 0)
158   {
159     flags |= 1 << IEEE80211_RADIOTAP_ANTENNA;
160     *(uint8_t*)(ptr + pos) = antenna;
161     pos += 1;
162   }
163 
164   // TX-Flags
165   if(opt_rts | opt_noack | opt_tsf)
166   {
167     pos = ALIGN(pos, 2);
168     flags |= 1 << IEEE80211_RADIOTAP_TX_FLAGS;
169     *(uint16_t*)(ptr + pos) = htole16(
170       opt_rts   ? IEEE80211_RADIOTAP_F_TX_RTS  : 0 |
171       opt_noack ? IEEE80211_RADIOTAP_F_TX_NACK : 0 |
```

```
172       opt_tsf   ? IEEE80211_RADIOTAP_F_TX_TSF  : 0 |
173       opt_tsf   ? IEEE80211_RADIOTAP_F_TX_SEQ  : 0);
174     pos += 2;
175   }
176
177   // MCS index
178   if(mcs >= 0)
179   {
180     flags |= 1 << IEEE80211_RADIOTAP_RATE_MCS;
181     *(uint16_t*)(ptr + pos) = mcs;
182     pos += 2;
183   }
184
185
186   // Header-Größe
187   *(uint16_t*)(ptr + 2) = htole16(pos);
188
189   // Flags
190   *(uint32_t*)(ptr + 4) = htole32(flags);
191
192
193   return pos;
194 }
195
196
197 static int mkieee80211(uint8_t *ptr, uint8_t *src, uint8_t *dst, uint8_t *bss,
      unsigned int seq)
198 {
199   unsigned int i;
200
201
202   ptr[0] = 0x48;
203   ptr[1] = 0x00;
204   ptr[2] = 0;
205   ptr[3] = 0;
206   for(i = 0; i < 6; i++) ptr[4  + i] = dst[i];
207   for(i = 0; i < 6; i++) ptr[10 + i] = src[i];
208   for(i = 0; i < 6; i++) ptr[16 + i] = bss[i];
209   *(uint16_t*)(ptr + 22) = htole16((seq % 0x1000) << 4);
210
211
212   return 24;
213 }
214
215
216 static int mkframe(uint8_t *ptr, unsigned int i)
217 {
218   unsigned int len;
219
220
221   len = 0;
222
223   len += mkradiotap(ptr, opt_bitrate, opt_mcs, opt_antenna);
224   len += mkieee80211(ptr + len, conf_src, conf_dst, conf_bss, i);
225
226   if(opt_tsf)
227   {
228     memset(ptr + len, 0, 8); len += 8;  // MACTIME
229   }
230
231   memset(ptr + len, 0, 4); len += 4;    // FCS
232
233
234   return len;
235 }
236
```

```
237
238 static void doit()
239 {
240   char errbuf[PCAP_ERRBUF_SIZE];
241   pcap_t *pcap;
242   int result;
243   uint8_t pktbuf[256];
244   unsigned int pktlen;
245   unsigned int i;
246
247
248   strcpy(errbuf, "");
249   pcap = pcap_open_live(opt_interface, 800, 1, 20, errbuf);
250
251   if(pcap == NULL)
252     error(-1, 0, "ERROR: %s\n", errbuf);
253
254   if(strlen(errbuf))
255     fprintf(stderr, "WARNING: %s\n", errbuf);
256
257
258   memset(pktbuf, 0, 64);
259   pktlen = mkframe(pktbuf, 0);
260
261   printf("%d packets "
262          "%02x:%02x:%02x:%02x:%02x:%02x -> "
263          "%02x:%02x:%02x:%02x:%02x:%02x "
264          "via %s (bssid  %02x:%02x:%02x:%02x:%02x:%02x)\n",
265          opt_count,
266          conf_src[0], conf_src[1], conf_src[2], conf_src[3], conf_src[4], conf_src
               [5],
267          conf_dst[0], conf_dst[1], conf_dst[2], conf_dst[3], conf_dst[4], conf_dst
               [5],
268          opt_interface,
269          conf_bss[0], conf_bss[1], conf_bss[2], conf_bss[3], conf_bss[4], conf_bss
               [5]);
270
271   for(i = 0; i < opt_count; i++)
272   {
273     if(opt_tsf)
274       pktlen = mkframe(pktbuf, i);
275
276     result = pcap_inject(pcap, pktbuf, pktlen);
277     if(result < 0)
278     {
279       fprintf(stderr, "ERROR: %s\n", pcap_geterr(pcap));
280       break;
281     }
282
283     usleep(opt_time);
284   }
285
286   printf("%d packets transmitted\n", i);
287
288
289   pcap_close(pcap);
290 }
291
292
293 int main(int argc, char **argv)
294 {
295   if(argc == 1)
296   {
297     usage(argv[0]);
298     return 0;
299   }
```

```
300
301
302   opts(argc, argv);
303
304   // INTERFACE
305   if(opt_interface == NULL)
306     error(-1, 0, "No␣interface␣given.");
307
308   // Anzahl Pakete
309   if(opt_count <= 0)
310     opt_count = 1;
311
312   // SRC MAC
313   if(opt_src == NULL)
314   {
315     int s;
316     struct ifreq ifr;
317     int result;
318     unsigned int i;
319
320
321     s = socket(AF_INET, SOCK_STREAM, 0);
322     if(s < 0)
323     {
324       perror("socket()");
325       return -1;
326     }
327
328     strncpy(ifr.ifr_name, opt_interface, IFNAMSIZ);
329     result = ioctl(s, SIOCGIFHWADDR, &ifr);
330     if(result < 0)
331     {
332       perror("ioctl(SIOCGIFHWADDR)");
333       return -1;
334     }
335
336     if(ifr.ifr_hwaddr.sa_family != ARPHRD_IEEE80211_RADIOTAP)
337       error(-1, 0, "%s␣ist␣not␣a␣radiotap␣device\n", opt_interface);
338     for(i = 0; i < 6; i++)
339       conf_src[i] = ifr.ifr_hwaddr.sa_data[i];
340
341     close(s);
342   }
343   else
344   {
345     if(str2mac(opt_src, conf_src))
346       error(-1, 0, "source␣MAC␣invalid");
347   }
348
349   // DST MAC
350   if(opt_dst == NULL)
351     error(-1, 0, "no␣destination␣MAC␣given");
352   else if(str2mac(opt_dst, conf_dst))
353     error(-1, 0, "destination␣MAC␣invalid");
354
355   // BSSID
356   if(opt_bss == NULL || str2mac(opt_bss, conf_bss))
357     conf_bss[0] = conf_bss[1] = conf_bss[2] =
358     conf_bss[3] = conf_bss[4] = conf_bss[5] = 0;
359
360   // Bitrate
361   if(opt_bitrate > 12)
362     error(-1, 0, "invalid␣bitrate␣given");
363
364   doit();
365
```

```
366 |
367 |   return 0;
368 | }
```

ieee80211_radiotap.h

```
  1 | /*
  2 |  * Copyright (c) 2003, 2004 David Young.  All rights reserved.
  3 |  *
  4 |  * Redistribution and use in source and binary forms, with or without
  5 |  * modification, are permitted provided that the following conditions
  6 |  * are met:
  7 |  * 1. Redistributions of source code must retain the above copyright
  8 |  *    notice, this list of conditions and the following disclaimer.
  9 |  * 2. Redistributions in binary form must reproduce the above copyright
 10 |  *    notice, this list of conditions and the following disclaimer in the
 11 |  *    documentation and/or other materials provided with the distribution.
 12 |  * 3. The name of David Young may not be used to endorse or promote
 13 |  *    products derived from this software without specific prior
 14 |  *    written permission.
 15 |  *
 16 |  * THIS SOFTWARE IS PROVIDED BY DAVID YOUNG ``AS IS'' AND ANY
 17 |  * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 18 |  * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 19 |  * PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL DAVID
 20 |  * YOUNG BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 21 |  * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
 22 |  * TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 23 |  * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 24 |  * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 25 |  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 26 |  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 27 |  * OF SUCH DAMAGE.
 28 |  */
 29 |
 30 | /*
 31 |  * Modifications to fit into the linux IEEE 802.11 stack,
 32 |  * Mike Kershaw (dragorn@kismetwireless.net)
 33 |  */
 34 |
 35 | #ifndef IEEE80211RADIOTAP_H
 36 | #define IEEE80211RADIOTAP_H
 37 |
 38 | /* Base version of the radiotap packet header data */
 39 | #define PKTHDR_RADIOTAP_VERSION         0
 40 |
 41 | /* Name                                 Data type    Units
 42 |  * ----                                 ---------    -----
 43 |  *
 44 |  * IEEE80211_RADIOTAP_TSFT              __le64       microseconds
 45 |  *
 46 |  *      Value in microseconds of the MAC's 64-bit 802.11 Time
 47 |  *      Synchronization Function timer when the first bit of the
 48 |  *      MPDU arrived at the MAC. For received frames, only.
 49 |  *
 50 |  * IEEE80211_RADIOTAP_CHANNEL           2 x __le16   MHz, bitmap
 51 |  *
 52 |  *      Tx/Rx frequency in MHz, followed by flags (see below).
 53 |  *
 54 |  * IEEE80211_RADIOTAP_FHSS              __le16       see below
 55 |  *
 56 |  *      For frequency-hopping radios, the hop set (first byte)
 57 |  *      and pattern (second byte).
 58 |  *
 59 |  * IEEE80211_RADIOTAP_RATE              u8           500kb/s
 60 |  *
```

```
61  *      Tx/Rx data rate
62  *
63  * IEEE80211_RADIOTAP_DBM_ANTSIGNAL     s8           decibels from
64  *                                                   one milliwatt (dBm)
65  *
66  *      RF signal power at the antenna, decibel difference from
67  *      one milliwatt.
68  *
69  * IEEE80211_RADIOTAP_DBM_ANTNOISE      s8           decibels from
70  *                                                   one milliwatt (dBm)
71  *
72  *      RF noise power at the antenna, decibel difference from one
73  *      milliwatt.
74  *
75  * IEEE80211_RADIOTAP_DB_ANTSIGNAL      u8           decibel (dB)
76  *
77  *      RF signal power at the antenna, decibel difference from an
78  *      arbitrary, fixed reference.
79  *
80  * IEEE80211_RADIOTAP_DB_ANTNOISE       u8           decibel (dB)
81  *
82  *      RF noise power at the antenna, decibel difference from an
83  *      arbitrary, fixed reference point.
84  *
85  * IEEE80211_RADIOTAP_LOCK_QUALITY      __le16       unitless
86  *
87  *      Quality of Barker code lock. Unitless. Monotonically
88  *      nondecreasing with "better" lock strength. Called "Signal
89  *      Quality" in datasheets.  (Is there a standard way to measure
90  *      this?)
91  *
92  * IEEE80211_RADIOTAP_TX_ATTENUATION    __le16       unitless
93  *
94  *      Transmit power expressed as unitless distance from max
95  *      power set at factory calibration.  0 is max power.
96  *      Monotonically nondecreasing with lower power levels.
97  *
98  * IEEE80211_RADIOTAP_DB_TX_ATTENUATION __le16       decibels (dB)
99  *
100 *      Transmit power expressed as decibel distance from max power
101 *      set at factory calibration.  0 is max power.  Monotonically
102 *      nondecreasing with lower power levels.
103 *
104 * IEEE80211_RADIOTAP_DBM_TX_POWER      s8           decibels from
105 *                                                   one milliwatt (dBm)
106 *
107 *      Transmit power expressed as dBm (decibels from a 1 milliwatt
108 *      reference). This is the absolute power level measured at
109 *      the antenna port.
110 *
111 * IEEE80211_RADIOTAP_FLAGS             u8           bitmap
112 *
113 *      Properties of transmitted and received frames. See flags
114 *      defined below.
115 *
116 * IEEE80211_RADIOTAP_ANTENNA           u8           antenna index
117 *
118 *      Unitless indication of the Rx/Tx antenna for this packet.
119 *      The first antenna is antenna 0.
120 *
121 * IEEE80211_RADIOTAP_RX_FLAGS          __le16       bitmap
122 *
123 *     Properties of received frames. See flags defined below.
124 *
125 * IEEE80211_RADIOTAP_TX_FLAGS          __le16       bitmap
126 *
```

```
127  *     Properties of transmitted frames. See flags defined below.
128  *
129  * IEEE80211_RADIOTAP_RTS_RETRIES       u8           data
130  *
131  *     Number of rts retries a transmitted frame used.
132  *
133  * IEEE80211_RADIOTAP_RATE_MCS       2 x u8          data, bitmap
134  *
135  *     First byte is the MCS index of the rate,
136  *     second one has flags about channel width and guard interval
137  *
138  * IEEE80211_RADIOTAP_DATA_RETRIES      u8           data
139  *
140  *     Number of unicast retries a transmitted frame used.
141  *
142  * IEEE80211_RADIOTAP_ISR_TSF           __le32       microseconds
143  *
144  *     Lower 32 bit of TSF counter at execution of ISR
145  *
146  * IEEE80211_RADIOTAP_ISR_TSC           __le32       timer ticks
147  *
148  *     Timestamp at execution of ISR based on TSC
149  *
150  * IEEE80211_RADIOTAP_ISR_HPET          __le32       timer ticks
151  *
152  *     Timestamp at execution of ISR based on HPET
153  */
154 enum ieee80211_radiotap_type {
155         IEEE80211_RADIOTAP_TSFT = 0,
156         IEEE80211_RADIOTAP_FLAGS = 1,
157         IEEE80211_RADIOTAP_RATE = 2,
158         IEEE80211_RADIOTAP_CHANNEL = 3,
159         IEEE80211_RADIOTAP_FHSS = 4,
160         IEEE80211_RADIOTAP_DBM_ANTSIGNAL = 5,
161         IEEE80211_RADIOTAP_DBM_ANTNOISE = 6,
162         IEEE80211_RADIOTAP_LOCK_QUALITY = 7,
163         IEEE80211_RADIOTAP_TX_ATTENUATION = 8,
164         IEEE80211_RADIOTAP_DB_TX_ATTENUATION = 9,
165         IEEE80211_RADIOTAP_DBM_TX_POWER = 10,
166         IEEE80211_RADIOTAP_ANTENNA = 11,
167         IEEE80211_RADIOTAP_DB_ANTSIGNAL = 12,
168         IEEE80211_RADIOTAP_DB_ANTNOISE = 13,
169         IEEE80211_RADIOTAP_RX_FLAGS = 14,
170         IEEE80211_RADIOTAP_TX_FLAGS = 15,
171         IEEE80211_RADIOTAP_RTS_RETRIES = 16,
172         IEEE80211_RADIOTAP_DATA_RETRIES = 17,
173         IEEE80211_RADIOTAP_RATE_MCS = 19,
174         IEEE80211_RADIOTAP_ISR_TSF = 20,
175         IEEE80211_RADIOTAP_ISR_TSC = 21,
176         IEEE80211_RADIOTAP_ISR_HPET = 22,
177
178         /* valid in every it_present bitmap, even vendor namespaces */
179         IEEE80211_RADIOTAP_RADIOTAP_NAMESPACE = 29,
180         IEEE80211_RADIOTAP_VENDOR_NAMESPACE = 30,
181         IEEE80211_RADIOTAP_EXT = 31
182 };
183
184 /* Channel flags. */
185 #define IEEE80211_CHAN_TURBO    0x0010  /* Turbo channel */
186 #define IEEE80211_CHAN_CCK      0x0020  /* CCK channel */
187 #define IEEE80211_CHAN_OFDM     0x0040  /* OFDM channel */
188 #define IEEE80211_CHAN_2GHZ     0x0080  /* 2 GHz spectrum channel. */
189 #define IEEE80211_CHAN_5GHZ     0x0100  /* 5 GHz spectrum channel */
190 #define IEEE80211_CHAN_PASSIVE  0x0200  /* Only passive scan allowed */
191 #define IEEE80211_CHAN_DYN      0x0400  /* Dynamic CCK-OFDM channel */
192 #define IEEE80211_CHAN_GFSK     0x0800  /* GFSK channel (FHSS PHY) */
```

```
193
194 /* For IEEE80211_RADIOTAP_FLAGS */
195 #define IEEE80211_RADIOTAP_F_CFP        0x01    /* sent/received
196                                                  * during CFP
197                                                  */
198 #define IEEE80211_RADIOTAP_F_SHORTPRE   0x02    /* sent/received
199                                                  * with short
200                                                  * preamble
201                                                  */
202 #define IEEE80211_RADIOTAP_F_WEP        0x04    /* sent/received
203                                                  * with WEP encryption
204                                                  */
205 #define IEEE80211_RADIOTAP_F_FRAG       0x08    /* sent/received
206                                                  * with fragmentation
207                                                  */
208 #define IEEE80211_RADIOTAP_F_FCS        0x10    /* frame includes FCS */
209 #define IEEE80211_RADIOTAP_F_DATAPAD    0x20    /* frame has padding between
210                                                  * 802.11 header and payload
211                                                  * (to 32-bit boundary)
212                                                  */
213 #define IEEE80211_RADIOTAP_F_BADFCS     0x40    /* bad FCS */
214
215 /* For IEEE80211_RADIOTAP_RX_FLAGS */
216 #define IEEE80211_RADIOTAP_F_RX_BADPLCP 0x0002  /* frame has bad PLCP */
217
218 /* For IEEE80211_RADIOTAP_TX_FLAGS */
219 #define IEEE80211_RADIOTAP_F_TX_FAIL    0x0001  /* failed due to excessive
220                                                  * retries */
221 #define IEEE80211_RADIOTAP_F_TX_CTS     0x0002  /* used cts 'protection' */
222 #define IEEE80211_RADIOTAP_F_TX_RTS     0x0004  /* used rts/cts handshake */
223 #define IEEE80211_RADIOTAP_F_TX_NACK    0x0008  /* don't expect ACK */
224 #define IEEE80211_RADIOTAP_F_TX_SEQ     0x0010  /* don't insert sequence number */
225 #define IEEE80211_RADIOTAP_F_TX_TSF     0x0020  /* insert TSF-Stamp */
226
227 /* For IEEE80211_RADIOTAP_RATE_MCS */
228 #define IEEE80211_RADIOTAP_RATE_MCS_40MHZ       0x01 /* 40 MHz channel width */
229 #define IEEE80211_RADIOTAP_RATE_MCS_SHORT_GI    0x02 /* short guard interval */
230
231 /* Ugly macro to convert literal channel numbers into their mhz equivalents
232  * There are certianly some conditions that will break this (like feeding it '30')
233  * but they shouldn't arise since nothing talks on channel 30. */
234 #define ieee80211chan2mhz(x) \
235         (((x) <= 14) ? \
236         (((x) == 14) ? 2484 : ((x) * 5) + 2407) : \
237         ((x) + 1000) * 5)
238
239 #endif                          /* IEEE80211_RADIOTAP_H */
```

# Anhang D

# APIs

## D.1 Messframework

**ip.up(**⟨iface⟩**)** aktiviert das Interface ⟨iface⟩. ⟨iface⟩ muss ein String sein.

**ip.down(**⟨iface⟩**)** deaktiviert das Interface ⟨iface⟩.

**ip.getmac(**⟨iface⟩**)** gibt die MAC-Adresse des Interface ⟨iface⟩ als String zurück.

**iw.addmon(**⟨iface⟩**,** ⟨moniface⟩**)** fügt dem Wireless-Interface ⟨iface⟩ ein Monitor-VIF mit dem Namen ⟨moniface⟩ hinzu.

**iw.del(**⟨iface⟩**)** entfernt das VIF mit dem Namen ⟨iface⟩.

**iw.settype(**⟨iface⟩**,** ⟨type⟩**)** setzt den Modus des VIF ⟨iface⟩. Der Parameter ⟨type⟩ kann dabei die Werte `"managed"`, `"ibss"` oder `"monitor"` annehmen. Das Interface muss dekativiert sein, damit diese Operation Erfolg hat.

**iw.setchan(**⟨iface⟩**,** ⟨chan⟩**)** stellt den Kanal mit der Kanalnummer ⟨chan⟩ auf dem Wireless-Interface ⟨iface⟩ ein. Das Interface muss zuvor aktiviert worden sein.

**iw.setrates(**⟨iface⟩**,** ⟨rates⟩**)** setzt die Bitraten-Maske des Interface ⟨iface⟩ im 2.4 und 5 GHz-Band. Der Parameter ⟨rates⟩ muss dazu ein Feld der erlaubten Bitraten enthalten. Ist ⟨rates⟩ gleich `nil`, wird die Bitratenmaske in beiden Bändern gelöscht.

**iw.joinibss(**⟨iface⟩**,** ⟨ssid⟩**,** ⟨chan⟩**)** lässt das Interface ⟨iface⟩ der Ad-hoc-Zelle mit der SSID ⟨ssid⟩ auf dem Kanal ⟨chan⟩ beitreten. Existiert diese Zelle noch nicht, wird sie von dieser Station erzeugt.

**iw.leaveibss(**⟨iface⟩**)** verlässt die Ad-hoc-Zelle, an der das Interface ⟨iface⟩ gebunden ist.

**iw.joinap(**⟨iface⟩**,** ⟨ssid⟩**,** ⟨chan⟩**)** nimmt über das Interface ⟨iface⟩ eine Verbindung mit der Infrastrukturzelle ⟨ssid⟩ auf Kanal ⟨chan⟩ auf.

**iw.leaveap(**⟨iface⟩**)** beendet die Verbindung mit dem Access Point, an dem das Interface ⟨iface⟩ gebunden ist.

`tshark.capture(`⟨iface⟩`,` ⟨file⟩`,` ⟨maxpacket⟩`,` ⟨filter⟩`)` startet den Netzwerksniffer Wireshark auf Interface ⟨iface⟩. Die Frames werden in der Datei ⟨file⟩ gespeichert. Ist ⟨maxpacket⟩ ein numerischer Wert, wird der Sniffer nach Aufzeichnung dieser Anzahl von Frames beendet. Der Parameter ⟨filter⟩ stellt einen PCAP-Filterstring dar, dem alle aufgezeichneten Frames entsprechen müssen. Ist der Filterstring `nil`, werden alle Frames aufgezeichnet.

`tshark.kill()` beendet alle laufenden Wireshark-Instanzen.

## D.2 Auswerteframework

`import.import(`⟨pcapfile⟩`)` importiert die Frames aus der Datei ⟨pcapfile⟩. ⟨pcapfile⟩ muss ein String sein. Die Frames werden als Array zurückgegeben. Jeder Frame wird als Tabelle von Key-Value-Paaren dargestellt.

`frame.is_rts(`⟨f⟩`)` gibt `true` zurück, wenn der Frame ⟨f⟩ ein RTS-Frame ist, andernfalls ist die Rückgabe `false`.

`frame.is_cts(`⟨f⟩`)` gibt `true` zurück, wenn der Frame ⟨f⟩ ein CTS-Frame ist, andernfalls ist die Rückgabe `false`.

`frame.is_data(`⟨f⟩`)` gibt `true` zurück, wenn der Frame ⟨f⟩ zur Kategorie der Datenframes gehört, andernfalls ist die Rückgabe `false`. Es ist zu beachten, dass NULL-Frames zu den Datenframes gehören.

`frame.is_null(`⟨f⟩`)` gibt `true` zurück, wenn der Frame ⟨f⟩ ein NULL-Frame ist, andernfalls ist die Rückgabe `false`.

`frame.is_ack(`⟨f⟩`)` gibt `true` zurück, wenn der Frame ⟨f⟩ ein ACK-Frame ist, andernfalls ist die Rückgabe `false`.

`frame.calc_duration(`⟨f⟩`)` gibt die Zeit in Mikrosekunden zurück, die für die Übertragung des Frames erforderlich ist. Es sind nur die Berechnungsvorschriften für DSSS-PHYs und OFDM-PHYs implementiert.

`list.proj(`⟨lin⟩`,` ⟨field⟩`)` nimmt die im Relationen Datenmodell als „Projektion“ bezeichnete Operation auf einem Array von Tupeln ⟨lin⟩ vor. Aus jedem Tupel wird der Wert des Felds mit dem Bezeichner ⟨field⟩ extrahiert. Die Wert werden in Form eines Arrays von der Funktion zurückgegeben.

Ein Großteil der Statistik-Funktionen nehmen eine optionale Mapping-Funktion $\langle \text{fn} \rangle : \mathbb{R} \to \mathbb{R}$ entgegen. Bis aus die unten erwähnten Ausnahmen durchlaufen alle Elemente des Eingabearrays vor einer weiteren Verarbeitung die Mapping-Funktion. Ist keine Mapping-Funktion angegeben, wird an ihrer Stelle die Identität als Mapping-Funktion verwendet.

`util.hist(`⟨xs⟩`,` ⟨fn⟩`)` berechnet ein Histogramm über die Werte im Array ⟨xs⟩. Das Histrogramm wird als Tabelle von Key-Value-Paaren zurückgegeben. Der Schlüssel eines solchen Paares entspricht dabei dem Wert der Ausprägung und das Wert-Element des Paares enthält die Anzahl der Vorkommen in ⟨xs⟩.

**util.meanvar(**⟨xs⟩**,** ⟨fn⟩**)** gibt ein Tupel zurück, welches das empirische Mittel und die empirische Varianz der Werte im Array ⟨xs⟩ angibt.

**util.cov(**⟨xs1⟩**,** ⟨xs2⟩**)** berechnet die Kovarianz der Stichprobenpaare $(xs1_i, xs2_i)$.

**util.cor(**⟨xs1⟩**,** ⟨xs2⟩**)** berechnet den Korrelationskoeffizienten der Stichprobenpaare $(xs1_i, xs2_i)$.

**util.clip(**⟨xs⟩**,** ⟨lower⟩**,** ⟨upper⟩**,** ⟨fn⟩**)** gibt ein Array zurück, welches nur die Werte aus ⟨xs⟩ enthält, die im Intervall [ ⟨lower⟩, ⟨upper⟩ ] liegen. Die Werte aus ⟨xs⟩ durchlaufen die Mapping-Funktion nur für den Vergleich mit den Intervallgrenzen. In die Rückgabeliste werden ihre ursprünglichen Werte übernommen.

**util.csplit(**⟨xs⟩**,** ⟨n⟩**)** zerlegt das Eingabearray $\langle xs \rangle = (x_1, \ldots, x_N)$ in $\lfloor N/n \rfloor$ Arrays der Länge $n$. Die Rückgabe besteht aus einem Array, welches die Einzelarrays enthält. Das $i$-te Teilarray besteht aus den Werten $(x_{n \cdot i+1}, \ldots, x_{n \cdot i+n})$.

**util.esplit(**⟨xs⟩**,** ⟨n⟩**)** arbeitet analog zu **csplit**. Das $i$-te Teilarray besteht allerdings aus den Werten $(x_i, x_{i+\lfloor N/n \rfloor}, \ldots, x_{i+(n-1)\cdot\lfloor N/n \rfloor})$.

**util.rsplit(**⟨xs⟩**,** ⟨n⟩**)** arbeitet analog zu **csplit**. Die Werte der Teilarrays werden allerdings zufällig aus ⟨xs⟩ ausgewählt.

**util.kappasigma(**⟨xs⟩**,** ⟨kappa⟩**,** ⟨sigma⟩**,** ⟨mu⟩**,** ⟨fn⟩**)** wendet das $\kappa$-$\sigma$-Verfahren mit den Parametern ⟨kappa⟩ und ⟨sigma⟩ auf die Werte im Array ⟨xs⟩ an. Ist der optionale Parameter ⟨mu⟩ gleich **nil**, wird für das Clipping-Intervall keine Mindestbreite festgesetzt. Das zurückgegebene Array besteht ebenfalls aus Werten aus ⟨xs⟩. Die Eingabewerte durchlaufen die Mapping-Funktion nur zur Berechnung von Mittelwert und Varianz.

**util.autocor(**⟨xs⟩**,** ⟨tau⟩**)** berechnet die Autokorrelation über den Werten von ⟨xs⟩ mit dem Lag-Parameter ⟨tau⟩.

# Anhang E

# Inhalt der beiliegenden DVD

- `/`
  - `da.pdf` ..... PDF-Datei dieser Arbeit
  - `livelinux.iso` ..... Image der Live-CD
  - `messungen/`
    - `adhoc/` ..... Rohdaten von Versuch 1
    - `managed_null/` ..... Rohdaten von Versuch 2 (NULL-ACK-Verfahren)
    - `managed_tsf/` ..... Rohdaten von Versuch 2 (Zeitstempelverfahren)
  - `packages/` ..... Quellarchive verwendeter Softwarepakete
  - `patches/` ..... Patches aus Anhang B
  - `src/`
    - `auswertetool/` ..... Auswerteframework
    - `messtool/` ..... Messframework
    - `nullinject/` ..... Das Programm `nullinject`

# Literaturverzeichnis


[ARW07] Abusubaih, M. ; Rathke, B. ; Wolisz, A.: A dual distance measurement scheme for indoor IEEE 802.11 wireless local area networks. In: *Mobile Wireless Communications Networks, 2007 9th IFIP International Conference on*, 2007, S. 121 –125

[Bera] Berg, Johannes: *The 802.11 subsystems – for kernel developers.* `http://linuxwireless.org/80211books/`

[Berb] Berg, Johannes: *mac80211 overview.* `http://wireless.kernel.org/en/developers/Documentation/mac80211?action=AttachFile&do=get&target=mac80211.pdf`

[BP00] Bahl, P. ; Padmanabhan, V.N.: RADAR: an in-building RF-based user location and tracking system. In: *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* Bd. 2, 2000, S. 775–784

[CBAI07] Ciurana, M. ; Barcelo-Arroyo, F. ; Izquierdo, F.: A ranging system with IEEE 802.11 data frames. In: *Radio and Wireless Symposium, 2007 IEEE*, 2007, S. 133–136

[CBC06] Ciurana, Marc ; Barceló, Francisco ; Cugno, Sebastiano: Indoor tracking in WLAN location with TOA measurements. In: *Proceedings of the 4th ACM international workshop on Mobility management and wireless access*, 2006 (MobiWac '06). – ISBN 1–59593–488–X, S. 121–125

[FFH09] Fross, Daniel ; Fross, André ; Heinkel, Ulrich: *IEEE 802.15.4a basiertes Sensor- und Lokalisierungsnetz.* Posterbeitrag, 2009

[FRH07] Fross, Daniel ; Rössler, Marco ; Heinkel, Ulrich: Entwurf eines Systems zur Positionsbestimmung auf Basis von Entfernungsmessungen zu Referenzpunkten. (2007). `http://archiv.tu-chemnitz.de/pub/2007/0082`

[Gas05] Gast, Matthew: *802.11 Wireless Networks: The Definitive Guide, Second Edition (Definitive Guide).* O'Reilly Media, Inc., 2005 `http://www.worldcat.org/isbn/0596100523`. – ISBN 0596100523

[GH05] Günther, André ; Hoene, Christian: Measuring Round Trip Times to Determine the Distance Between WLAN Nodes. Version: 2005. `http://dx.`

doi.org/10.1007/11422778_62. In: *NETWORKING 2005.* 2005. – DOI 10.1007/11422778_62, S. 768–779

[Has93] Hashemi, H.: The indoor radio propagation channel. In: *Proceedings of the IEEE* 81 (1993), Nr. 7, S. 943–968. http://dx.doi.org/10.1109/5.231342. – DOI 10.1109/5.231342

[HFL+04] Haeberlen, Andreas ; Flannery, Eliot ; Ladd, Andrew M. ; Rudys, Algis ; Wallach, Dan S. ; Kavraki, Lydia E.: Practical robust localization over large-scale 802.11 wireless networks. In: *Proceedings of the 10th annual international conference on Mobile computing and networking*, 2004 (MobiCom '04). – ISBN 1–58113–868–7, S. 70–84

[HHSW09] Halperin, Daniel ; Hu, Wenjun ; Sheth, Anmol ; Wetherall, David: *802.11 with Multiple Antennas for Dummies.* 2009

[HW08] Hoene, C. ; Willmann, J.: Four-way TOA and software-based trilateration of IEEE 802.11 devices. In: *Personal, Indoor and Mobile Radio Communications, 2008. PIMRC 2008. IEEE 19th International Symposium on*, 2008, S. 1–6

[IEE07] IEEE Computer Society (Hrsg.): *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification.* IEEE, 3 Park Avenue, New York, NY 10016-5997, USA: IEEE Computer Society, Juni 2007

[IEE09] IEEE Computer Society (Hrsg.): *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification – Amendment 5: Enhancements for Higher Throughput.* IEEE, 3 Park Avenue, New York, NY 10016-5997, USA: IEEE Computer Society, Oktober 2009

[Ier06] Ierusalimschy, Roberto: Programmieren mit Lua. (2006). https://www.opensourcepress.de/index.php?26&backPID=15&tt_products=87. ISBN 3–937514–22–8

[IFC06] Ierusalimschy, Roberto ; Figueiredo, Luiz Henrique d. ; Celes, Waldemar: *Lua 5.1 Reference Manual.* Lua.Org, 2006 http://www.lua.org/manual/5.1/. – ISBN 8590379833

[Int04] Intel Corporation: IA-PC HPET (High Precision Event Timers) Specification. 2004. – Specification

[Int07] Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual. Version: 2007. http://www.intel.com/products/processor/manuals/index.htm. 2007. – Specification

[KK04] Kaemarungsi, Kamol ; Krishnamurthy, Prashant: Properties of Indoor Received Signal Strength for WLAN Location Fingerprinting. In: *Mobile and Ubiquitous Systems, Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MobiQuitous'04)* (2004), S. 14–23. http://dx.doi.org/10.1109/MOBIQ.2004.1331706. – DOI 10.1109/MOBIQ.2004.1331706. ISBN 0–7695–2208–4

[KQ06] Kunst, Eva-Katharina ; Quade, Jürgen: Kernel- und Treiberprogrammierung mit dem Kernel 2.6 – Folge 27. In: *Linux-Magazin* (2006), April, Nr. 4. http://www.linux-magazin.de/Heft-Abo/Ausgaben/2006/04/Kern-Technik

[LC02] Lepak, J. ; Crescimanno, M.: Speed of light measurement using ping. In: *ArXiv Physics e-prints* (2002), Januar. http://arxiv.org/abs/physics/0201053v2

[LDBL07] Liu, Hui ; Darabi, H. ; Banerjee, P. ; Liu, Jing: Survey of Wireless Indoor Positioning Techniques and Systems. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 37 (2007), November, Nr. 6, S. 1067–1080. http://dx.doi.org/10.1109/TSMCC.2007.905750. – DOI 10.1109/TSMCC.2007.905750. – ISSN 1094–6977

[LPLaY00] Li, Xinrong ; Pahlavan, K. ; Latva-aho, M. ; Ylianttila, M.: Comparison of indoor geolocation methods in DSSS and OFDM wireless LAN systems. In: *Vehicular Technology Conference, 2000. IEEE VTS-Fall VTC 2000. 52nd* Bd. 6, 2000, S. 3015–3020

[MAC] *The official Linux Wireless wiki.* http://wireless.kernel.org/

[MDF+00] McCrady, D.D. ; Doyle, L. ; Forstrom, H. ; Dempsey, T. ; Martorana, M.: Mobile ranging using low-accuracy clocks. In: *Microwave Theory and Techniques, IEEE Transactions on* 48 (2000), Juni, Nr. 6, S. 951–958. http://dx.doi.org/10.1109/22.846721. – DOI 10.1109/22.846721. – ISSN 0018–9480

[Mur07] Murphy, William S. Jr.: *Determination Of A Position In Three Dimensions Using Approximate Distances And Trilateration.* Juli 2007

[PCA] *libpcap.* http://www.tcpdump.org/

[PKB02] Pahlavan, K. ; Krishnamurthy, P. ; Beneat, A.: Wideband radio propagation modeling for indoor geolocation applications. In: *Communications Magazine, IEEE* 36 (2002), August, Nr. 4, S. 60–65. http://dx.doi.org/10.1109/35.667414. – DOI 10.1109/35.667414

[PLM02] Pahlavan, K. ; Li, Xinrong ; Makela, J. P.: Indoor geolocation science and technology. In: *IEEE Communications Magazine* 40 (2002), Feb, Nr. 2, S. 112–118. http://dx.doi.org/10.1109/35.983917. – DOI 10.1109/35.983917. – ISSN 01636804

[Rad] *Beschreibung des Radiotap-Protokolls.* http://www.radiotap.org/

[Rec08] Rech, Jörg: *Wireless LANs: 802.11-WLAN-Technologie und praktische Umsetzung im Detail.* Heise Zeitschriften Verlag GmbH & Co. KG, Hannover, 2008. – ISBN 9783936931518

[SA07] Saunders, Simon R. ; Aragon, Alejandro: *Antennas and Propagation for Wireless Communication Systems.* 2. A. Wiley & Sons, 2007 http://www.worldcat.org/isbn/0470848790. – ISBN 0470848790

[SGAK05] Sujak, B. ; Ghodgaonkar, D.K. ; Ali, B.M. ; Khatun, S.: Indoor propagation channel models for WLAN 802.11b at 2.4 GHz ISM band. In: *Applied Electromagnetics, 2005. APACE 2005. Asia-Pacific Conference on*, 2005, S. 5 pp.

[Siz04] Sizun, H.: *Radio Wave Propagation for Telecommunication Applications (Foundations of Engineering Mechanics)*. Springer Verlag, 2004. – ISBN 3540407588

[Tim10] Timmer, Kai: *Gruppengestützte Lokalisierung von Smartphones*. Februar 2010

[Tou] Tourrilhes, Jean: *Wireless Extensions for Linux*. `http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html`

[TP08] *ThinkPad Computer Hardware Maintenance Manual – R50r, R51e, R52.* : *ThinkPad Computer Hardware Maintenance Manual – R50r, R51e, R52*. Lenovo, April 2008. `http://www-307.ibm.com/pc/support/site.wss/MIGR-54932.html`

[TP10] *ThinkPad T410 and T410i Hardware Maintenance Manual.* : *ThinkPad T410 and T410i Hardware Maintenance Manual*. Lenovo, Oktober 2010. `http://www-307.ibm.com/pc/support/site.wss/MIGR-74470.html`

[VK04] Velayos, Hector ; Karlsson, Gunnar: Limitations in range estimation for wireless LAN. In: *Proc. 1st Workshop on Positioning, Navigation and Communication (WPNC'04)*, 2004

[WKP09] Wibowo, Sigit B. ; Klepal, Martin ; Pesch, Dirk: Time of Flight Ranging using Off-the-self IEEE802.11 WiFi Tags. In: *International Conference on Positioning and Context-Awareness (POCA 2009)*, 2009

[WS] *Wireshark*. `http://www.wireshark.org/`

[YA05] Youssef, Moustafa ; Agrawala, Ashok: The Horus WLAN location determination system. In: *Proceedings of the 3rd international conference on Mobile systems, applications, and services*. New York, NY, USA : ACM, 2005 (MobiSys '05). – ISBN 1–931971–31–5, S. 205–218

# Danksagung

Mit Abgabe und Verteidigung dieser Arbeit ist mein Studium beendet. An dieser Stelle ist es angebracht, einer Reihe von Personen meinen Dank auszusprechen. Zuallererst danke ich meiner gesamten Familie – besonders meinen Eltern – für die herzliche ideelle und materielle Unterstützung. In dankbarer Erinnerung bleiben mir auch meine beiden Großmütter, die die Ergebnisse ihrer Bemühungen nicht mehr erleben können.

Meinen Freunden Ron, Chris und Robin gelang es in den vergangen Jahren stets meinen Ehrgeiz herauszufordern, wofür ich ihnen sehr dankbar bin. Jens und Pierre danke ich für das Aufspüren der unzähligen Tippfehler in dieser Arbeit. Die gelegentlichen mathematischen Fachsimpeleien mit Ralph und Jens halfen mir viel. Froh bin ich auch über die Unterstützung von Prof. Werner, der meinen Themenvorschlag für diese Arbeit ohne Vorbehalte annahm. Stellvertretend für das Rechenzentrum danke ich Ullrich Fritsche dafür, dass er meine WLAN-Experimente innerhalb des Campusnetzes tolerierte.

Über die oben genannten Personen hinaus, konnte ich mich stets auf den Rückhalt vieler Freunde, Kommilitonen und Kollegen verlassen. Danke! Versteht es bitte keineswegs als Geringschätzung, dass ich euch nicht alle namentlich erwähnen kann.

Unbekannter Weise möchte ich mich abschließend bei Donald, Leslie, Till und all den anderen TEX-Größen für den ästhetischen Text und die schönen Grafiken bedanken.

## Selbstständigkeitserklärung

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Diplomarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

.....
Mario Haustein