Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwurf und Implementierung eines Systems zur präzisen Messung und Charakterisierung der Latenz von WiFi-Links

Julian Siebert

Studiengang:	Informatik, B.Sc.
Prüfer/in:	Prof. Dr. Christian Becker
Betreuer/in:	Dr. rer. nat. Frank Dürr
Beginn am:	18. Februar 2023
Beendet am:	18. August 2023

Kurzfassung

Allgegenwärtige Konnektivität hat die Gewohnheiten der Menschen drastisch verändert und eine Vielzahl neuartiger Anwendungen sowohl in Industrie als auch bei Privatnutzern ermöglicht. Bereits heute läuft ein großer Anteil der Kommunikation über das Internet über den Standard IEEE 802.11, besser bekannt unter dem Markennamen WiFi. Die aufkommenden Anwendungen cyber-physischer Systeme in der Industrie, aber auch AR/VR-Anwendungen oder Onlinespiele für Privatnutzer erfordern dabei eine deterministische Güte der Übertragung. Während sich die Datenrate über WiFi seit Einführung des ersten Standards stark erhöht hat, bleibt eine zuverlässige Beschränkung der Latenz eine offene Herausforderung. Daher besteht die Notwendigkeit, Latenz und Jitter präzise messen und charakterisieren zu können. Da Sender und Empfänger keine gemeinsame Uhr besitzen, gestaltet sich die präzise Erfassung der Latenz einer Übertragungsrichtung bislang schwierig.

In dieser Arbeit wird ein System entworfen und implementiert, welches mithilfe einer drahtgebundenen Verbindung der GPIO-Pins von Mikrocontrollern als Sender und Empfänger die Messung der Latenzen mit einer Uhr ermöglicht und das Erstellen von Messungen so deutlich vereinfacht. So können Latenz und Jitter der Kommunikation über einen WiFi-Link präzise erfasst und charakterisiert werden. Mithilfe des entworfenen Systems wird die Charakteristik der Übertragung über einen WiFi-Link in realer Umgebung evaluiert. Insbesondere wird dabei die Verbesserung der Charakteristik bei Verwendung von Enhanced Distributed Channel Access (EDCA), sowie des Time-Aware Shapers (TAS) zur Reduktion der Latenzvarianz analysiert. Durch EDCA kann eine Verringerung der maximalen Latenz erzielt werden; mit Einsatz des TAS zur Kompensation der variablen Paketverzögerung wird eine sehr hohe Zuverlässigkeit erzielt. Zuletzt wird die Latenz der Übertragung durch Scheduling mehrerer Stationen deutlich reduziert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

AR/VR Augmented Reality / Virtual Reality. 15

CSMA-CA Carrier Sense Multiple Access – Collision Avoidance. 18

CSMA-CD Carrier Sense Multiple Access – Collision Detection. 18

DCF Distributed Coordination Function. 19

DHCP Dynamic Host Configuration Protocol. 57

EDCA Enhanced Distributed Channel Access. 17

FRC Free-Running Counter. 64

GCL Gate Control List. 26

GPIO General Purpose Input/Output. 41

HCF Hybrid Coordination Function. 19

IFS Inter Frame Spacing. 19

IP Internet Protocol. 57

ISM Industrial, Scientific and Medical Band. 30

ISR Interrupt Service Routine. 59

NTP Network Time Protokol. 39

PCF Point Coordination Function. 18

PCP Priority Code Point. 25

PTP Precision Time Protocol. 26

TAS Time-Aware Shaper. 17

TGbe IEEE Task Group for 802.11be. 35

TRM Technical Reference Manual. 43

TSN Time-Sensitive Networking. 15

1 Einleitung

Die Art und Weise, wie wir leben und arbeiten, verändert sich durch das Internet zunehmend [Cas14]. Immer mehr und immer unterschiedlichere Geräte kommunizieren miteinander und ermöglichen auf diese Weise eine Vielzahl von Anwendungsfällen - sowohl für Privatnutzer als auch für die Industrie. So konsumieren Privatnutzer Medien und Nachrichten, spielen Computerspiele oder entdecken neuartige Anwendungen der Augmented Reality / Virtual Reality (AR/VR). In der Industrie gewinnen digitale Zwillinge, kollaborative Robotik oder die vorausschauende Wartung von Maschinen an Bedeutung.

Um diese Anwendungsfälle zu ermöglichen, ob zu Hause oder in der Industrie, sollen Daten mit verschiedenartigen Anforderungen über das selbe Netzwerk übertragen werden. Verschiedene Anwendungen sind dabei mehr oder weniger tolerant gegenüber unerwarteten Verzögerungen in der Informationsübertragung. Für einige Anwendungen genügt ein Dienst ohne garantierte Übertragungsqualität, während andere Anwendungen harte Anforderungen, etwa die Festlegung einer maximalen Latenz, erfordern können. So genügt für die Übermittlung von Logdateien oder den Download einer großen Datei im Hintergrund die Nutzung der gerade verfügbaren Bandbreite. Im Gegensatz dazu kann die rechtzeitige Übertragung von Informationen erforderlich sein: Es hätte katastrophale Folgen, wenn die Aktuatoren eines Flugzeuges aufgrund von Überlastung des Netzwerks nur verzögert gesteuert werden könnten. Erhält ein Roboter den Befehl, den Hebearm zu bewegen verspätet, kann dies Arbeiter gefährden. Auch in der Unterhaltungselektronik sind unvorhersehbare Informationsverzögerungen ein Problem: Bei Computerspielen könnte ein Spieler ein kritisches Spiel verlieren, weil sein Gegner verspätet dargestellt wird, bei AR/VR-Anwendungen führt eine verzögerte Darstellung zu *motion sickness* [CKY20]. Viele dieser Anwendungsfälle werden derzeit noch durch kabelgebundene Systeme ermöglicht.

Vernetzte Geräte sind heutzutage allgegenwärtig - eine entscheidende Rolle dabei spielt drahtlose Kommunikation [Cas14]. Drahtlose Kommunikation bietet verschiedene Vorteile gegenüber der Verwendung von Kabeln: Geräte unabhängig von Netzwerkdosen und Kabelstrecken verbinden zu können, erlaubt höhere Flexibilität und damit einfachere wie schnellere Bereitstellung von Geräten – diese können auch eingesetzt werden, wo Verkabelung kaum möglich oder schlicht zu teuer ist. Darüber hinaus wird die Verwendung vernetzter, tragbarer Geräte wie Smartphones oder Wearables erst durch Drahtlosnetzwerke möglich gemacht. Eine weit verbreitete Möglichkeit der drahtlosen Kommunikation mit vergleichsweise hoher Bandbreite sind die Funknetze der Protokollfamilie IEEE 802.11, bekannter unter dem Markennamen *WiFi*. WiFi wird durch eine Vielzahl von Geräten unterstützt, die Unterstützung durch etwa Smartphones oder Laptops wird von Konsumenten heutzutage erwartet.

In der Task-Group IEEE 802.1 wird unter der Bezeichnung *Time-Sensitive Networking (TSN)* eine Reihe von Standards erarbeitet, die Mechanismen zur Übertragung von Daten via Ethernet mit definierter Dienstgüte – insbesondere hoher Verfügbarkeit und Scheduling für beschränkte, niedrige Latenz – spezifizieren. Um dem Bedarf eines geeigneten Netzwerks für alle Anwendungen gerecht zu

werden, bedarf es einer Erweiterung der Mechanismen für spezifizierte Dienstgüte auf WiFi-Geräte – allen voran, um die genannten Anwendungen auch mit den Vorteilen der Drahtlosnetzwerke möglich zu machen. Während die Datenrate seit Einführung des ersten WiFi-Standards immer weiter angestiegen ist und inzwischen Übertragungen mit einer Geschwindigkeit von $10\,\mathrm{Gbit\,s^{-1}}$ ermöglicht, ist der zuverlässige Erhalt einer niedrigen Latenz eine offene Herausforderung.

Zur Evaluierung bestehender und zukünftig vorgeschlagener Lösungsansätze ist die Messung der Ende-zu-Ende-Latenz der Übertragung über drahtlose Links erforderlich. Herausfordernd ist dabei, diese Messungen präzise zu gestalten. Bisherige Ansätze beschränken sich oft auf die Messung der Round-Trip Latenz oder bedürften spezialisierter Netzwerkkarten. In vielen der zukünftigen Anwendungen, etwa mobilen Robotern oder Produktionsanlagen, kommen jedoch keine spezialisierten Netzwerkkarten, sondern kostengünstige Mikrocontroller zum Einsatz. Das entworfene System soll daher nah an diesen Einsatzbereichen konzipiert sein.

In dieser Arbeit wird ein System entworfen und implementiert, welches mithilfe von Mikrocontrollern als Sender und Empfänger die präzise Messung der Latenz einer Übertragungsrichtung ermöglicht und das Erstellen von Messungen so deutlich vereinfacht. Dazu kommt eine drahtgebundene Verbindung zwischen den Mikrocontrollern zum Einsatz, die ermöglicht, Sende- und Empfangszeitstempel mithilfe von einer Uhr zu erfassen. Durch Verwendung spezieller Funktionalität der Mikrocontroller zur hardwaregestützten Erfassung von Signalen können die Messungen sehr präzise erfasst werden.

Zur Evaluation bestehender Funktionalität des WiFi-Standards zur Priorisierung von Rahmen sowie einem Ansatz zur Kompensation der variablen Paketverzögerung werden verschiedene Szenarien entworfen, in denen das Messsystem zum Einsatz kommt. So werden unterschiedliche Aspekte zur Verbesserung der Dienstgüte von WiFi untersucht und diskutiert.

Die Arbeit ist in folgender Weite gegliedert: In Kapitel 2 werden die notwendigen technischen Grundlagen für das Verständnis der Arbeit gelegt. Dazu wird eine Einführung in WiFi sowie ein Schlüsselkonzept von Time-Sensitive Networking, den Time-Aware Shaper, gegeben. Außerdem erfolgt eine knappe Vorstellung der verwendeten Mikrocontroller. Kapitel 3 liefert einen Überblick über Verwandte Arbeiten. In Kapitel 4 wird das System beschrieben, sowie die Problemstellung erläutert. Dazu werden als Metrik Key Performance Indicators eingeführt. Kapitel 5 beschreibt den Entwurf des Messsystems. Dort werden verschiedene Szenarien eingeführt, die als Versuchsaufbau umgesetzt werden und jeweils die Untersuchung einer Fragestellung ermöglichen. Schließlich wird die Umsetzung dieses Messsystems entworfen. In Kapitel 6 wird dessen Implementierung beschrieben. Zuletzt werden die Messergebnisse der Szenarien in Kapitel 7 vorgestellt und diskutiert.

2 Technischer Hintergrund

Der Entwurf eines Messsystems erfordert technisches Verständnis der verwendeten Technologien. In diesem Kapitel werden die relevanten technischen Grundlagen eingeführt.

Zunächst ist ein grundlegendes Verständnis von WiFi notwendig. Dazu wird im ersten Abschnitt des Kapitels auf WiFi, die Verfahren zur Kollisionsvermeidung und insbesondere die verteilte Koordinierungsfunktion im Detail eingegangen. Weiterhin wird ein Verfahren zur Priorisierung wichtiger Kommunikationsflüsse, Enhanced Distributed Channel Access (EDCA), vorgestellt.

Der zweite Abschnitt fokussiert sich auf die Entwicklung von Time-Sensitive Networking. Besonders wichtig ist dabei eine Technologie zum Scheduling von echtzeitkritischem Verkehr, der Time-Aware Shaper (TAS), dessen Funktionsweise erläutert wird.

Das Kapitel endet in einem Abschnitt über Mikrocontroller, in dem nach einer knappen Einführung die verwendeten Mikrocontroller-Plattformen dieser Arbeit vorgestellt werden.

2.1 WiFi

Unter dem Standard IEEE 802.11 – besser bekannt unter dem Markennamen WiFi – wird das Funkprotokoll stetig entwickelt, das unsere Gewohnheit zu kommunizieren drastisch verändert hat [KLA20]. Es ermöglicht heutzutage eine Vielzahl von Anwendungen und ist aus dem Alltag vieler Menschen nicht mehr wegzudenken. Auch in der Industrie spielt WiFi eine größer werdende Rolle. Nach Fertigstellung des ersten Standards mit einer Datenrate von $2\,\mathrm{Mbit\,s^{-1}}$ lag der Fokus der Weiterentwicklung auf einer Erhöhung der möglichen Datenrate. Mit diversen Erweiterungen des Standards ist WiFi mit der neuesten Generation „WiFi 6“ bereits in der Lage, theoretisch knapp $10\,\mathrm{Gbit\,s^{-1}}$ nominale Datenrate zu liefern [KLA20]. Eine offene Herausforderung bleibt, deterministische Dienstgüte für die Übertragung von Rahmen spezifizieren zu können.

2.1.1 Überblick

Mehrere Geräte – sogenannte Stationen – sind in der Lage, mithilfe ihrer Radioschnittstelle Kontakt miteinander aufzunehmen. Im Ad-Hoc-Modus kommunizieren mehrere Stationen direkt miteinander – für diese Arbeit relevanter ist jedoch der Infrastruktur-Modus. In diesem ist ein Access Point als diejenige Station beteiligt, die sowohl in das Drahtlosnetz integriert ist, als auch (meist per Ethernet) mit der kabelgebundenen Infrastruktur verbunden ist [Rot05]. So bietet ein Access Point nahtlose Konnektivität zwischen drahtlos und kabelgebundenen Geräten in einem gemeinsamen Netzwerk. Zur Information anderer Stationen über beispielsweise Netzwerkidentifikation oder unterstützte Funktionalität werden regelmäßig Rahmen, sogenannte Beacons, durch den Access Point versendet.

Ein Access Point und die weiteren Stationen, oft als Klienten bezeichnet, die auf derselben Frequenz übertragen, bezeichnet man als Basic Service Set (BSS), welches durch die MAC-Adresse des Access Points eindeutig identifiziert wird [Rot05]. Überlappen sich mehrere BSSs, also befindet sich eine sendende Station in Empfangsreichweite einer zu einem anderen BSS gehörigen Station, spricht man von Overlapping Basic Service Sets (OBSS) [IEE16, Clause 4.3].

WiFi verwendet die lizenzfreien ISM-Bänder (Industrial, Scientific and Medical Band). Diese Frequenzbereiche sind jedoch nicht ausschließlich für den Betrieb von WiFi vorgesehen, sondern können lizenzfrei auch durch andere Anwendungen verwendet werden – etwa Bluetooth-Geräte oder Babyphone operieren im selben Frequenzbereich; sogar Mikrowellenöfen können die Übertragung beeinträchtigen [PGB13]. Für die Betrachtung der WiFi-Performanz ist daher zu beachten, dass die Verfügbarkeit des Übertragungsmediums – selbst ohne andere sendende WiFi-Geräte – nicht garantiert ist.

2.1.2 Medienzugriffsverfahren

Aufgrund des geteilten Mediums kann es passieren, dass mehrere Geräte gleichzeitig darauf zugreifen. Übertragen zwei oder mehr Geräte gleichzeitig auf der gleichen Frequenz, kommt es zu einer Kollision – diese gilt es zu vermeiden. Daher wird ein Verfahren benötigt, welchen den Zugriff auf das Medium regelt [MGS+12].

Auch in kabelgebundenen Ethernet-Netzwerken besteht der Bedarf der Regelung des Zugriffs auf das Medium (dort das Kabel), dazu wird Carrier Sense Multiple Access – Collision Detection (CSMA-CD) verwendet [IEE22]. Die Geräte hören also das Übertragungsmedium ab und beginnen erst mit dem Sendevorgang, wenn dieses frei ist. Anschaulich wird dies auch als „Listen before Talk“ bezeichnet. Tritt eine Kollision auf, wird diese erkannt und die Übertragung abgebrochen.

Dieses Verfahren kann für Drahtlosnetzwerke nicht eingesetzt werden, da die Empfangsschnittstelle während des Sendevorgangs üblicherweise nicht zur Verfügung steht. Eine Erkennung einer Kollision in der laufenden Übertragung ist somit nicht möglich [KLA20]. Stattdessen werden Kollisionen in WiFi vermieden, das Verfahren heißt Carrier Sense Multiple Access – Collision Avoidance (CSMA-CA) [BC13]. Auf das Verfahren zur Vermeidung von Kollisionen, die verteilte Koordinierungsfunktion, wird noch im Detail eingegangen. Erfolgreiche Übertragungen werden positiv bestätigt. Erfolgt kein Acknowledgement, wird die Information erneut übertragen.

Mit einem WiFi Access Point können viele Klienten verbunden sein – also besteht auch innerhalb der verbundenen WiFi-Klienten der Bedarf der Koordinierung. Dies wird durch eine Koordinierungsfunktion (engl.: coordination function) übernommen. Dafür gibt es grundsätzlich drei Kategorien:

- *Zentral koordinierter Zugriff.* Ein Koordinator regelt den Zugriff und legt fest, welches Gerät wann Daten übertragen werden können. Der WiFi Access Point ermöglicht es Klienten, direkt auf den Kanal zugreifen zu können. Das Verfahren dazu, die Point Coordination Function (PCF), wurde allerdings in der Revision 2016 des 802.11-Standard abgeschafft [IEE16, Clause 10.4] .

- *Dezentraler Zugriff.* Jedes Gerät, das auf das drahtlose Medium zugreifen möchte, überprüft eigenständig, ob der Beginn einer Übertragung zu einer Kollision führen würde, und minimiert die Kollisionswahrscheinlichkeit. Die Distributed Coordination Function (DCF) wird im folgenden Abschnitt im Detail erläutert.
- *Hybrider Ansatz.* Es gibt einerseits Phasen mit zentral koordiniertem Zugriff, andererseits Phasen, in denen dezentraler Zugriff erfolgt. Für WiFi ist dies die Hybrid Coordination Function (HCF) [IEE16, Clause 10.22].

In der Praxis wird fast ausschließlich die Distributed Coordination Function (DCF) verwendet. Die PCF fand auch vor ihrer Abschaffung nie breite Verbreitung; nach [KLA20] ist auch die Hybrid Coordination Function (HCF) zu kompliziert für die Umsetzung in realen Geräten.

Distributed Coordination Function (DCF)

Bei Verwendung der DCF entscheidet jede WiFi-Station eigenständig über das Timing der Übertragung. Wird das drahtlose Medium durch die Übertragung einer anderen Station in Anspruch genommen, wird es als „belegt“ erkannt; umgekehrt wird es als „frei“ erkannt, wenn gerade keine andere Station überträgt [KIS+13]. Dies geschieht mithilfe des *Clear Channel Assessments*, einem Dienst, der durch die Bitübertragungsschicht zur Verfügung gestellt wird, und den aktuellen Zustand des drahtlosen Mediums beurteilt [IEE16, Clause 8]. Die Sicherungsschicht verwendet diesen Dienst, um den Zustand des drahtlosen Mediums zu ermitteln.

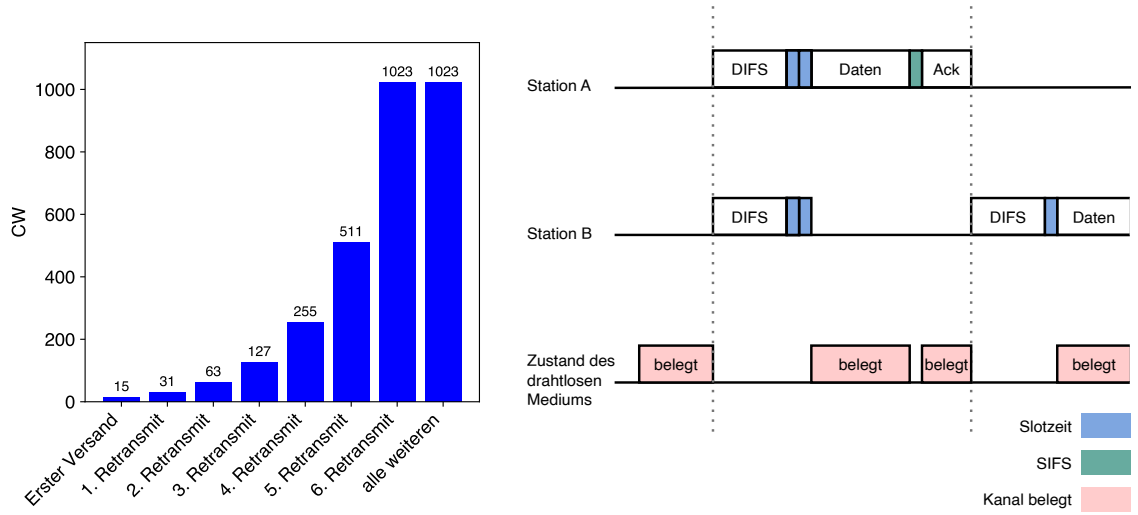
Möchte eine Station senden, wartet sie zunächst auf das freiwerden des Mediums. Wird das Medium frei oder ist es bereits frei, beginnt die Übertragung allerdings nicht sofort, sondern frühestens nach Verstreichen eines gewissen Zeitintervalls. Dieses Zeitintervall spezifiziert den minimalen zeitlichen Abstand zweier WiFi-Frames und wird folglich als Inter Frame Spacing (IFS) bezeichnet; es ist für unterschiedliche Arten von Frames unterschiedlich lang [IEE16].

Frametypen mit niedrigerem IFS haben dadurch eine höhere Priorität, da das Medium für einen kürzeren Zeitraum frei gewesen sein muss, bis die Übertragung beginnt und das Medium somit wieder belegt.

Die IEEE 802.11-Spezifikation in der Version von 2020 definiert insgesamt 10 verschiedene IFSs [21a, S. 2777]. Für die fünf relevantesten Intervalle wird in Tabelle 2.1 jeweils ein Beispiel-Frame angegeben, für den das jeweilige IFS verwendet wird.

IFS	**Interframe Space**	**Beispiel**
SIFS	Short Interframe Space	Acknowledgements
PIFS	Priority Interframe Space	TIM Rahmen (Beacons)
DIFS	DCF Interframe Space	Datenrahmen, gewisse Management-Rahmen
AIFS	Arbitration Interframe Space	EDCAF für Endgeräte mit QoS-Fähigkeit
EIFS	Extended Interframe Space	nach Rahmen mit falscher Frame Check Sequence

Tabelle 2.1: Interframe Spaces in IEEE 802.11



(a) CW-Wert je Retransmit **(b)** Beispiel der Kanalbelegung

Abbildung 2.1: Erläuterung der Backoff Time

Der Vollständigkeit halber ist anzumerken, dass es neben den genannten IFSs noch das inzwischen obsolete RIFS, sowie SBIFS, BRPIFS, MBIFS, LBIFS in Zusammenhang mit Beamforming im WiFi-Standard aufgeführt werden.

Die Dauer des jeweiligen IFS wird, mit der Ausnahme des AIFS, jeweils je Bitübertragungsschicht festgelegt. Die Bitübertragungsschicht (PHY) ist beispielsweise für Modulation / Demodulation oder Signalstärkenerfassung zuständig, um die physische Übertragung der Datenbits über das drahtlose Medium zu steuern. Insbesondere ist die Dauer der IFS unabhängig von der Bitrate der Stationen. Für das OFDM PHY [IEE16, Clause 17] liegt die Dauer für den Short Interframe Space beispielsweise bei 16 µs bei 20 MHz Kanalabstand oder 32 µs bei 10 MHz Kanalabstand; für das HT PHY [IEE16, Clause 19] bei 10 µs im 2,4 GHz Band.

Random Backoff Timer

Würden zwei Stationen mit Sendewunsch eines Datenrahmens sofort nach Ablauf der DIFS-Zeit senden, käme es zu einer Kollision. Als Gegenmaßnahme, zur Kollisionsvermeidung, gibt es zusätzlich die *Random Backoff Time*. Zunächst wird das Medium abgehört. Nach Erkennung eines freien Mediums stellt die Station die Übertragung zunächst für die Dauer des DIFS-Intervalls zurück, sofern die letzte Übertragung erfolgreich war, oder für die Dauer des EIFS-Intervalls, wenn die letzte Übertragung nicht erfolgreich war. Nach Ablauf des DIFS- bzw. EIFS-Intervalls stellt die Station die Übertragung zusätzlich für eine zufällige BackoffTime zurück.

Für diese BackoffTime gilt nach [IEE16, Clause 10.3]:

$$\text{BackoffTime} = \text{Random}() \cdot \text{aSlotTime}$$

Die Backoffzeit setzt sich also zusammen aus einer pseudozufällig gewählten Zahl und der Slotzeit „aSlotTime“. Diese bezeichnet – ähnliche wie bei SIFS – ein Zeitintervall, das je PHY festgelegt ist.

Am Beispiel der Übertragung eines gewöhnlichen Datenrahmens erklärt: Die Station mit Sendewunsch hört zunächst das Medium ab. Sobald es frei wird, wartet sie für die Dauer der DIFS-Zeit. Ist das Medium so lange frei geblieben, beginnt die Random Backoff Time.
Die pseudozufällige Anzahl abzuwartender Slotzeiten wird gewählt und einem rückwärts zählenden *Backoff Interval Counter* zugewiesen. Immer wenn das drahtlose Medium für eine Slotzeit frei war, wird dieser Zähler dekrementiert – sobald er Null erreicht, beginnt die Station mit der eigentlichen Übertragung. Überträgt in der Zwischenzeit eine andere Station, wird danach erneut die Dauer einer DIFS-Zeit abgewartet und mit dem gleichen Wert des Backoff Interval Counters fortgefahren. Dieses Vorgehen sorgt für Fairness zwischen den Stationen, da jede Station schließlich senden kann. Wurde der Rahmen empfangen, wird dessen erfolgreicher Erhalt durch ein Acknowledgement bestätigt. Auch dieses wird nicht unmittelbar gesendet, sondern erst, nachdem das Medium für die Dauer der SIFS-Zeit frei war. Abbildung 2.1b illustriert diesen Vorgang, im gewählten Beispiel muss Station A zwei Slotzeiten warten, Station B drei Slotzeiten.

Der Wert der Zufallsfunktion Random() entstammt einer Gleichverteilung über das Intervall $[0, CW]$. Der Parameter CW beschreibt die Größe des *Contention Windows* innerhalb von $CW_{min} \leq CW \leq CW_{max}$.
Initial hat CW den Wert CW_{min}. Zählen zwei Stationen ihre jeweiligen Random Backoff Timer gleichzeitig auf den Wert 0 herunter, kommt es zu einer Kollision; die Übertragung schlägt dadurch fehl. Mit jeder fehlgeschlagenen Übertragung nimmt CW den nächsten Wert in der Reihe an $(CW \leftarrow 2 \cdot CW + 1)$, bis CW_{max} erreicht ist. Dann wird auch bei weiteren Retransmits der Wert $CW = CW_{max}$ beibehalten [IEE16, Clause 10.3]; solange, bis beispielsweise durch die erfolgreiche Übertragung eines Rahmens CW auf CW_{min} zurückgesetzt wird.

CW_{min}, CW_{max} werden über die Beacons des Access Points mitgeteilt. Abbildung 2.3 zeigt die Darstellung eines Beacons in Wireshark; für Sender ohne gesondert spezifizierte Dienstgüte ist in diesem Beispiel $CW_{min} = 15, CW_{max} = 1023$. Für eben diese Werte zeigt Abbildung 2.1a den Anstieg des CW-Werts je Retransmit.

Insgesamt handelt es sich also um einen Mechanismus, der die Kollisionswahrscheinlichkeit senkt, indem bei fehlgeschlagenen Übertragungen eine probabilistisch größer werdende Anzahl an Slotzeiten abzuwarten, bevor erneut eine Übertragung gestartet wird.

Für das DSSS PHY [IEE16, Clause 15] liegt die Slotzeit beispielsweise bei 20 µs; für das OFDM PHY [IEE16, Clause 17] beträgt sie 9 µs bei 20 MHz Kanalabstand oder 13 µs bei 10 MHz Kanalabstand. Eine exemplarische Rechnung mit einer Slotzeit von 9 µs und dem „Worst Case“, bei $CW = 1023$ auch 1023 Slots abwarten zu müssen, ergibt sich zusätzlich zur Dauer des EIFS eine Haltezeit von $BackoffTime = 9\,\mu s \cdot 1023 = 9{,}207$ ms. Diese Zeit, in der der Rahmen zurückgehalten wird, ist um ein Vielfaches länger als das eigentliche IFS und macht den enormen Einfluss des Random Backoff Timers auf die Übertragungslatenz deutlich.

Nebenbei bemerkt ist die Slotzeit auch für DIFS relevant, denn DIFS = SIFS + 2 · aSlotTime [IEE16, Clause 10]. Exemplarisch für das HT PHY im 2,4 GHz Frequenzband berechnet ergibt sich damit DIFS = 10 µs + 2 · 9 µs = 28 µs.

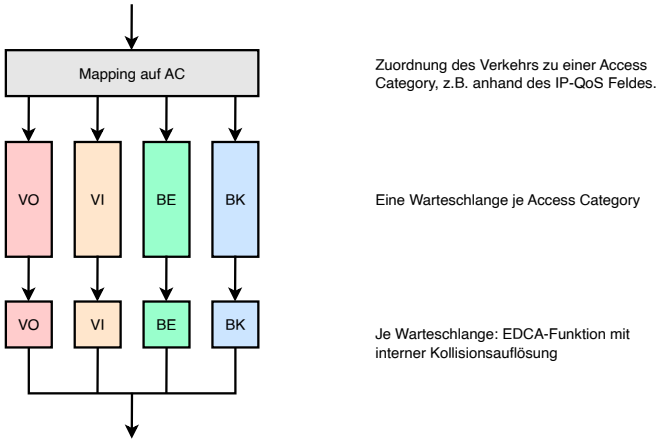


Abbildung 2.2: Modell der Implementierung der Access Categories

2.1.3 Enhanced Distributed Channel Access

Aufgrund der begrenzten Frequenzressourcen kann die Kommunikationsqualität in Situationen mit vielen Stationen beziehungsweise einer hohen Auslastung des drahtlosen Mediums stark beeinträchtigt werden [KIS+13]. So sinkt die Nutzbarkeit vor allem von Multimediaanwendungen. Etwa Internettelefonie oder das Streaming von Videos erfordert eine höhere Dienstgüte als die Übertragung einer Datei. Daher kann die Priorität der WiFi-Übertragung verschiedener Anwendungsklassen unterschieden werden.

Mit der Einführung von IEEE 802.11e im Jahr 2005 wurden zwei neue Modi des Zugriffs auf das Medium eingeführt [BC13].

- *Hybrid Coordination Function Controlled Channel Access (HCCA).* Dabei tritt ein zentraler Koordinator auf (der *Hybrid Coordinator (HC)*) [IEE16, Clause 10.22]. Innerhalb einer *Contention Free Period* vergibt der HC Möglichkeiten, als einziger Klient zu senden – sogenannte *Transmit Opportunities (TXOPs)* – an die Stationen.
- *Enhanced Distributed Channel Access (EDCA).* Das Medienzugriffsverfahren EDCA stellt eine Erweiterung der DCF dar. Rahmen können priorisiert werden, indem sie einer von vier Access Categories (ACs) zugewiesen werden. Auf die Mechanismen von EDCA wird nun im Detail eingegangen.

Access Categories

EDCA verwendet vier Access Categories, um für verschiedene Dienste unterschiedliche Dienstgüte liefern zu können. Der Benutzer kann dazu eine Benutzerpriorität zwischen 0 und 7 angeben, die gemäß Tabelle 2.2 auf die vier Zugriffskategorien „Voice“, „Video“, „Best Effort“ und „Background“ aufgeteilt werden [IEE16, Clause 10.2].

Abbildung 2.2 stellt dar, wie eine Implementierung von EDCA auf der Station aussehen könnte. Zunächst erfolgt eine Zuordnung der eingehenden Rahmen zu der jeweiligen Zugriffskategorie. Enthält der Rahmen einen VLAN (802.1Q) Tag, kann dazu der Wert des *Priority Code Point (PCP)*

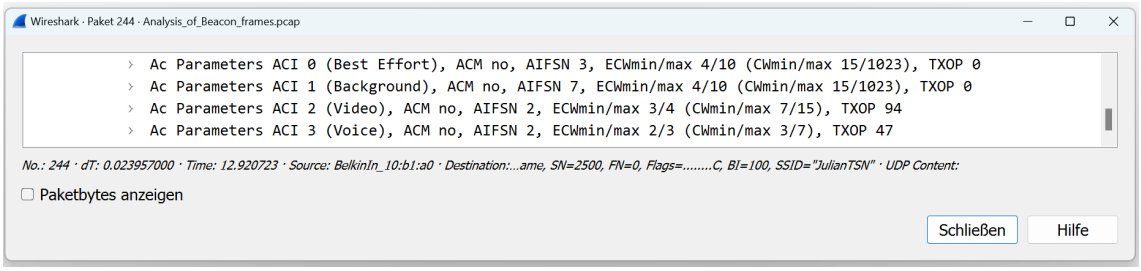


Abbildung 2.3: Ausschnitt der Parameter eines Beacon-Frames in Wireshark

verwendet werden. Möglich ist auch die Verwendung von Informationen aus dem IP-Header des enthaltenen Pakets, zum Beispiel der IP *Differentiated services code point* (DSCP) [IEE16, Clause 11.25]. Die Frames gelangen dann in die jeweilige FIFO-Warteschlange ihrer AC. Je Queue wird dann das Medienzugriffsverfahren durchgeführt; Konflikte der Medienzugriffsverfahren mehrerer Queues werde behoben, indem der Rahmen mit höherer Priorität zuerst gesendet wird.

Die Priorisierung zwischen den verschiedenen ACs erfolgt, indem zwei Parameter des Medienzugriffsverfahrens der DCF verändert werden:

- Statt des DIFS kommt das AIFS[i] zum Einsatz, wobei AIFS[i] bezeichnet, dass die Dauer des AIFS für jede Zugriffskategorie unterschiedlich sein kann [IEE16, Clause 10.2]. Mit sinkender Priorität steigt die Dauer des AIFS[i].
- Die Parameter CW_{min} und CW_{max} unterscheiden sich je Zugriffskategorie. Je höher die Priorität der Zugriffskategorie ist, desto niedriger sind CW_{min}, CW_{max}. Daraus ergeben sich probabilistisch niedrigere Werte des Random Backoff Counters für Zugriffskategorien höherer Priorität [RCAL12].

Die Parameter – AIFS[i] sowie CW_{min}, CW_{max} je AC – werden durch den Access Point als Bestandteil des periodischen Beacons mitgeteilt. Abbildung 2.3 zeigt dazu einen Auszug eines in Wireshark angezeigten Beacon-Rahmens. Zuletzt scheint erwähnenswert, dass auch bei Verwendung von EDCA die Möglichkeit besteht, eine TXOP anzufragen [IEE16, Clause 10.22].

Eine Teilmenge des WiFi 802.11e-Standards wird von der WiFi-Alliance als „WiFi Multimedia“ spezifiziert [Com12] und vermarktet. WiFi Multimedia basiert auf EDCA und lässt HCCA außen vor.

Priorität	Zugriffskategorie	Verkehrstyp
Niedrigste	AC_BK	Background
	AC_BE	Best Effort
	AC_VI	Video
Höchste	AC_VO	Voice

Tabelle 2.2: Prioritätszuordnung von EDCA

2.2 Time-Sensitive Networking

Ethernet findet sowohl bei Heimanwendern als auch in der Industrie bereits eine hohe Verbreitung – dennoch sind manche Anwendungsgebiete mit „klassischem“ Ethernet nicht immer umzusetzen, da hohe Anforderungen an die Güte der Kommunikation gestellt werden.

Time-Sensitive Networking (TSN) bezeichnet eine Reihe an Standards, die mit dem Ziel entwickelt werden, Zuverlässigkeit und Determinismus der Übertragung in Ethernet-Netzen zu verbessern, so dass auch echtzeitkritische Anwendungen realisiert werden können. Die TSN Task-Gruppe der IEEE erarbeitet dazu Standards, um deterministische Übertragung in Ethernet-Netzen zu ermöglichen [CCSR22]. Schlüsselkonzepte für TSN-Standards sind akkurate Zeitsynchronisierung zwischen Geräten, Scheduling zur Beschränkung der Latenz sowie die Reduktion des Jitters.

Rein kabelbasierte Systeme sind allerdings nicht immer möglich – beispielsweise für mobile Roboter, oder aufgrund der Notwendigkeit der Verlegung von Kabeln nicht flexibel oder skalierbar genug [KLG+21]. Daher fokussiert sich die Forschung nun auch auf TSN in drahtlosen Netzen, z.B. 5G oder WiFi 802.11be.

2.2.1 Kommunikationsklassen

Die Art und Weise der Kommunikation mehrere Geräte lässt sich in drei Klassen aufteilen, die sich hinsichtlich ihrer Periodizität und Anforderungen an die Dienstgüte unterscheiden. In den Standards des „European Telecommunications Standards Institute“ [3GP20], [3GP21] werden die folgenden Kategorien definiert:

1. *Deterministische periodische Kommunikation.* Die Periodizität bezieht sich auf festgelegte Zeitintervalle. In jedem Zeitintervall erfolgt eine Übertragung, diese hat strenge Anforderungen an die Zuverlässigkeit.
2. *Deterministische aperiodische Kommunikation.* Im Gegensatz zur deterministischen periodischen Kommunikation erfolgt die Übertragung nicht in im Vorhinein bekannten Zeitintervallen, sondern wird typischerweise durch externe Ereignisse ausgelöst. Die Übertragung hat strenge Anforderungen an die Zuverlässigkeit.
3. *Nichtdeterministische Kommunikation.* Unter nichtdeterministische Kommunikation fallen alle anderen Verkehrsklassen, die keine Anforderungen an die Dienstgüte haben. Die Periodizität ist hierfür irrelevant.

In Abgrenzung zu *echtzeitkritischem* Verkehr, beispielsweise Industrieprozessen wie der Steuerung von Robotern, bei denen der die Anforderungen streng sind und der Verkehr strikt innerhalb der vorgesehenen Zeitschranke zugestellt werden muss, gibt es auch *echtzeitsensiblen* Verkehr: Beispielsweise Anwendungen wie Onlinespiele oder Audio/Video-Streaming profitieren von geringer Latenz, erfordern aber nicht dieselben Garantien wie echtzeitkritischer Verkehr.

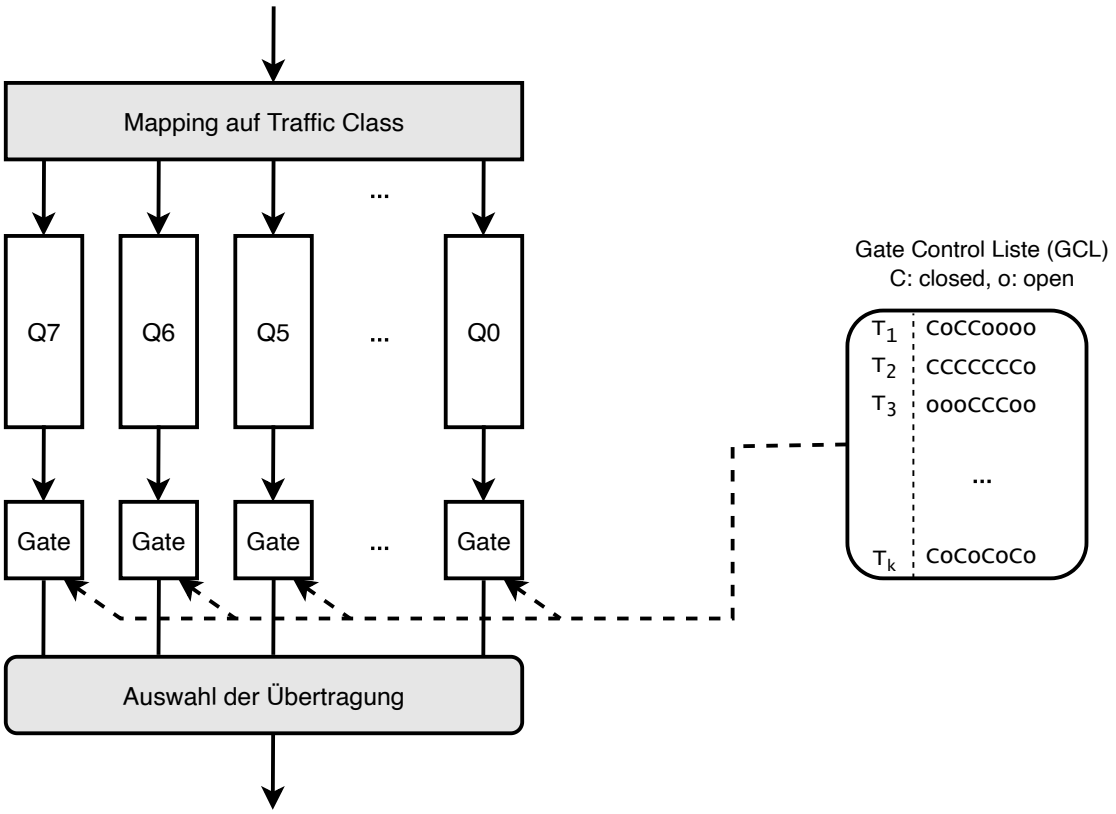


Abbildung 2.4: Architektur eines einzelnen Ports einer Bridge mit TAS

2.2.2 Time-Aware Shaper (TAS)

Um echtzeitkritischen Verkehr zu ermöglichen, wird zu jedem Zeitpunkt nur eine Verkehrsklasse oder eine definierte Menge an Verkehrsklassen übertragen [18, Annex Q] – so kann ein Kommunikationsfluss hoher Priorität nicht von Übertragungen niedrigerer Priorität gestört werden. Dieses Verfahren wird im 2016 erschienenen IEEE-Standard 802.1 Qbv spezifiziert.

Abbildung 2.4 fasst die funktionale Struktur des TAS für einen ausgehenden Port in Anlehnung an [18, Figure 8-14] zusammen.

Grundprinzip ist zunächst die Aufteilung der eingehenden Ethernet-Rahmen in acht Verkehrsklassen (engl: traffic classes) [KLG+21]. Eingehender Verkehr wird basierend auf seiner Priorität einer von acht Warteschlangen zugeordnet werden. Dazu kann der im VLAN-Tag des Rahmens enthaltene Priority Code Point (PCP) Wert verwendet werden, aber auch eine Auswertung des IP DSCP Werts (analog wie bei EDCA) oder schlicht eine Zuweisung der Priorität basierend auf dem einem eingehenden Port der Bridge ist möglich.

Wann genau der Verkehr einer Warteschlange übertragen wird, wird durch „Gates“ bestimmt. Jede Warteschlange wird durch ein Gate reguliert, ist dieses geschlossen, werden die enthaltenen Rahmen zurückgehalten und nicht über den ausgehenden Port übertragen.

Die Steuerung der Gates wird in einer Gate Control List (GCL) für jeden ausgehenden Port der Bridge in Form von absoluten Zeitpunkten spezifiziert. Die GCL enhält also Tupel aus Zeitpunkten und der Information „offen“ oder „geschlossen“ je Gate. Durch diese werden die Intervalle bestimmt, in denen die Gates offen bzw. geschlossen sind. Voraussetzung ist eine gemeinsame Ansicht der aktuellen Zeit aller Bridges und gegebenenfalls der Endsysteme bei sogenanntem isochronem Verkehr; diese muss z.B. durch das Precision Time Protocol (PTP) hergestellt werden [18, Annex Q].

Die Gate Control List (GCL) stellt einen Zeitplan dar, der periodisch abgearbeitet – also nach Ende wiederholt wird. Folglich eignet sich ein Netzwerk mit TAS besonders gut für periodische, deterministische Kommunikation. Die GCL wird durch einen zentralen Controller des Netzwerks bereitgestellt.

Im Zusammenhang mit dem TAS wird oft auch *Frame Preemption* verwendet: Wird bereits ein Rahmen niedrigerer Priorität übertragen, sobald ein Rahmen höherer Priorität übertragen werden soll, wird die Übertragung des Rahmens niedrigerer Priorität zugunsten des höher priorisierten Rahmens unterbrochen.

2.3 Mikrocontroller

Ein Mikrocontroller ist eine Einheit aus Prozessor, RAM und anderen Komponenten wie z.B. Timer-Module, Interrupt Controller oder Flash-Speicher, die auf einen Chip integriert werden [GW07]. Durch ihre geringe Baugröße und die Integration der Komponenten miteinander sparen sie dem Verwender Zeit und Geld gegenüber der Verwendung einzelner Komponenten. Sie sind dabei meist auf gewisse Anwendungsgebiete zugeschnitten, sodass die Auswahl des richtigen Mikrocontrollers entscheidend ist [GW07].

2.3.1 GPIO-Pins und Input Capture

Mithilfe von GPIO-Pins – *General Purpose Input/Output-Pins* – können Mikrocontroller beispielsweise mit Sensoren, Anzeigen oder auch anderen Mikrocontrollern interagieren. GPIO-Pins bieten damit eine flexible Möglichkeit, als Eingänge oder Ausgänge in einer Vielzahl an Projekten oder Anwendungen verwendet zu werden.

Ein für die Arbeit wichtiges Features ist *Input Capture*. Input Capture wird verwendet, um (externe) Ereignisse – ansteigende, abfallende oder wechselnde Taktflanken eines Eingangssignals – erfassen zu können. Sobald ein solches Ereignis auftritt, speichert der Mikrocontroller den Wert eines laufenden Timers in ein Hardwareregister, ein *Input Capture Register* [GW07]. Dies erlaubt im Allgemeinen hohe Präzision, da die Erfassung des Zeitpunktes des Ereignisses in Hardware geschieht und somit unabhängig vom gerade ausgeführten Code ist.

2.3.2 Verwendete Hardware

Im Verlauf der Arbeit kommen zwei verschiedene Mikrocontroller-Plattformen zum Einsatz, die im folgenden Abschnitt kurz vorgestellt werden.

1. Der Teensy 4.1 [PJR23] mit dem Mikroprozessor i.MX RT 1062 von NXP [NXP23]. Der Teensy wird im Verlauf der Arbeit als „Messinstrument“ eingesetzt.
2. Mikroprozessoren aus der ESP32-Serie von Espressif [Esp23b]. Aufgrund ihrer WiFi-Unterstützung eignen sie sich als drahtlos verbundener Endpunkt.

ESP32

Espressif bietet nach dem großen Erfolg der Mikrocontroller mit integrierter WiFi-Unterstützung ESP8266 und ESP32 eine ganze Reihe solcher Mikroprozessoren und Module an [Esp23c]. Aus diesem Produktportfolio werden in der Arbeit zwei Produkte verwendet: das ältere Modul ESP32-WROOM-32 und der neuere ESP32-C6.

Das Modul ESP32-WROOM-32 verwendet den Mikroprozessor ESP32-D0WDQ6 mit zwei Kernen der Xtensa-Architektur, die eine konfigurierbare Frequenz zwischen 80 MHz und 240 MHz bieten. Darüber hinaus bietet der Chip 34 programmierbare GPIO-Pins und enthält 448 kB ROM sowie 520 kB SRAM. Im Modul sind 4 MB per SPI verbundenem Flash-Speicher verbaut [23a], [23c]. Neben Bluetooth wird WiFi 802.11 b/g/n („WiFi 4“) mit bis zu 150 Mbit s^{-1} unterstützt.

Das neuere, auf dem ESP32-C6 basierende Modul ESP32-C6-WROOM-1 ist zum Zeitpunkt dieser Arbeit erst als Engineering Sample verfügbar. Es bietet mit 320 kB ROM sowie 512 kB SRAM leicht reduzierten Speicher, die RISC-V CPU bietet bis zu 160 MHz Taktfrequenz. Neben Bluetooth und IEEE 802.15.4 wird ebenfalls WiFi mit einer Datenrate bis 150 Mbit s^{-1} unterstützt – allerdings wird schon der neueste Standard 802.11ax („WiFi 6“) implementiert. Darüber hinaus unterstützt der ESP32-C6 WiFi Multimedia (WMM) [23b].

Beide funken ausschließlich im 2,4 GHz Frequenzband. Zur Programmierung der Mikrocontroller stellt Espressif die Plattform *Espressif IoT Development Framework (ESP-IDF)* bereit, welches auf dem Betriebssystem FreeRTOS aufbaut.

Teensy 4.1

Der Mikrocontroller Teensy 4.1 von PJRC bietet 7 936 kB Flash und 1 024 kB RAM sowie 55 Input/Output-Pins. Der verbaute Mikrochip, der i.MX RT 1062 von NXP basiert auf der ARM Cortex-M7 Architektur und bietet mit 600 MHz für einen Mikroprozessor viel Leistung. Integrierte Konnektivität bietet das 100 Mbit s^{-1} Ethernet Interface [PJR23].

Der i.MX RT 1062 ermöglicht hardwaregestütztes Erfassen von Sende- und Empfangszeitstempeln der Rahmen über das Ethernet-Interface sowie Input Capture / Output Compare an vier Pins ebenfalls über die Ethernet Clock [Sem21].

3 Verwandte Arbeiten

In diesem Kapitel werden die thematisch verwandten Arbeiten vorgestellt. Zunächst wird ein Überblick in den aktuellen Stand der Entwicklung von der nächsten Generation von WiFi (IEEE 802.11be) gegeben, sowie die aktuell diskutierten Vorschläge zur Weiterentwicklung zusammengefasst. Dies dient einer Einbettung der weiteren Arbeit in den aktuellen Stand der Forschung. Im Anschluss werden verschiedene thematisch verwandte Arbeiten vorgestellt, knapp zusammengefasst und zu dieser Arbeit abgegrenzt.

3.1 WiFi: Aktueller Stand und weitere Entwicklung

In den letzten WiFi-Generationen stieg die Datenrate immer weiter – dabei sind hohe Datenraten alleine nicht ausreichend, um echtzeitkritische Anwendungen zu ermöglichen: Der Kanalzugriff kann trotzdem eine lange Zeit in Anspruch nehmen, sodass die Latenz immer noch hoch ist [KLA20]. Viele Anwendungen erfordern dagegen jedoch enge Zeitschranken bei der maximalen Latenz der Übertragung. Unter Verwendung von WiFi sind solche Anwendungen noch nicht immer möglich. [3GP20] nennt verschiedene Industriebereiche, in denen ähnliche Produkte oder Dienste bereitgestellt haben – sogenannte „Vertical Domains“ und deren typische Anforderungen an Latenz und Zuverlässigkeit. So erfordern etwa videogesteuerte Roboter eine maximale Ende-zu-Ende Latenz zwischen 10 ms und 100 ms. Auch in [CCSR22] werden verschiedene Anwendungsbereiche aufgeführt, die beschränkte Latenz und hohe Zuverlässigkeit erfordern. Diese motivieren die Forschung an Verbesserungen des WiFi-Standards.

3.1.1 Arbeiten zur Evaluation des aktuellen Stands

Eine Vielzahl von Arbeiten beschäftigt sich mit den Entwicklungen des WiFi-Standards bis heute: Für geschichtlichen Hintergrund wird in [BC13] ein Überblick über die WiFi-Standards bis 802.11ae und der damals erwarteten Entwicklung bis 802.11aq gegeben; [KLA20] liefert einen kompakten Überblick über die bisherige Entwicklung des 802.11-Standards bis hin zu WiFi 6 (802.11ax). [MTM22] geht genauer auf die Anwendungsfälle und neuen Features von 802.11ax ein.

[KLG+21] nennt einige Gründe dafür, dass WiFi aktuell nicht in der Lage ist, deterministische Dienstgüte bei Übertragungen zu liefern:

- Aufgrund der instabilen Übertragungsumgebung, etwa unvorhergesehener Belegung des Mediums, kommt es zu Paketverlust und dadurch Neuübertragung des Pakets.
- Die Kollisionsvermeidung sorgt für eine hohe Wartezeit vor Beginn der Übertragung, was wiederum eine hohe Latenz verursacht.

Ergänzend nennt [KLA20], dass EDCA wie in Unterabschnitt 2.1.3 vorgestelt – der bislang einzige in der Praxis genutzte Mechanismus, eine Dienstgüte zu spezifizieren – keinen Determinismus erzielen kann.

Grundsätzlich ist es für WiFi-Netzwerke schwierig, bei der Übertragung feste Zeitschranken einzuhalten [KLA20] – schließlich arbeitet WLAN in den lizenzfreien Industrial, Scientific and Medical Band (ISM)-Bändern; die Freiheit des Übertragungskanals kann außerhalb fest kontrollierter Netze – etwa auf einem Firmencampus – nicht garantiert werden. Diese Abhängigkeit der Performanz des WiFi-Links von der Umgebung stellt eine große Herausforderung dar. Insbesondere bei überlappenden Basic Service Sets ist der Auslastungsgrad kaum kontrollierbar [CRC+18].

Zeitsensitive Anwendungen erfordern jedoch vorhersehbare, üblicherweise niedrige Latenz der Übertragung [CRC+18]. Die exakten Anforderungen an die Dienstgüte hängen von der jeweiligen Anwendung ab. Nachdem bisherige Standards sich vor allem auf einen höheren Durchsatz fokussiert haben, spielt in der aktuellen Generation der Drahtlosnetzwerke die Umsetzung von Anforderungen an eine deterministische, niedrige Latenz eine größere Rolle [KRD+18].

3.1.2 Entwicklungen für drahtloses Time-Sensitive Networking

Daher gibt es viele Vorschläge zur Weiterentwicklung des Standards, welche echtzeitkritische beziehungsweise echtzeitsensible Anwendungen ermöglichen sollen. Es wird erwartet, dass neuere WiFi-Standards eine bessere Unterstützung für TSN-Anwendungen liefern werden [CCSR22]. Etwa in [KLG+21] erwarten die Autoren, dass WiFi 802.11be periodische deterministische Kommunikation ermöglichen werde.

Eine Vielzahl an Autoren liefert dazu konkrete Vorschläge: [KLA20] wertet eine Vielzahl dieser Vorschläge aus und liefert eine Zusammenfassung der aktuellen Entwicklungen bei der Spezifikation von IEEE 802.11be. Beispielsweise [CRV+18] beschreibt zwei Lösungsvorschläge, den TAS aus 802.1Qbv (Unterabschnitt 2.2.2) in das Medienzugriffsverfahren von 802.11 (Unterunterabschnitt 2.1.2) zu integrieren. Einer der genannten Vorschläge baut dazu auf EDCA auf, indem dessen vier Warteschlangen zusätzlich mit Gates versehen werden. Dies wird auch in der Veröffentlichung [CCSR22] genannt, darüber hinaus fasst diese weitere Möglichkeiten des TSN und offene Herausforderungen zusammen. Auch [ACB21] erläutert TSN-Funktionalitäten, um Herausforderungen und Chancen sowie Einsatzmöglichkeiten dieser im kommenden Standard 802.11be zu diskutieren.

3.2 Latenzen und Anwendungsbeispiele

Bevor der Einfluss einzelner WiFi-Features bewertet werden kann, wird im Verlauf der Arbeit ein System entworfen, um präzise Latenzen messen zu können. In verschiedenen verwandten Arbeiten wird die Latenz von Übertragungen gemessen, gegebenenfalls spezifisch für ein Vertical oder eine konkretes Anwendungsbeispiel.

[MGS+12] analysiert Latenz und Paketverlust von WiFi-Broadcasts mit „off the shelf“ Hardware, die Autoren verwenden dabei die „Voice“-Zugriffskategorie von EDCA. In der Gegenwart von Cross-Traffic wird eine deutlich angestiegene Latenz beobachtet, und, da bei Broadcasts keine Retransmits erfolgen, erhöhter Paketverlust von 18%.

[PZC+16] analysiert die WiFi-Latenzen realer Installationen, indem 47 OpenWrt-kompatible Access Points bereitgestellt werden. An einem Universitätscampus werden durch Analyse der TCP-Handshakes die Latenzen der WiFi-Links gemessen. Im Gegensatz zu dieser Arbeit werden nur Round-Trip Zeiten betrachtet. Der Schwerpunkt liegt auf der Analyse der Ursachen für hohe Latenzen; es kommt ein Entscheidungsbaum zum Einsatz, der Verbindungen anhand ihrer Eigenschaften in „schnell“ und „langsam“ klassifiziert. Durch z.B. Umplatzierung der Access Points oder Kanalsauswahl konnten Verbesserungen erzielte werden, es werden keine Parameter des WiFi-Standards an sich analysiert.

Nicht nur Round-Trip Latenzen, sondern – wie in dieser Arbeit – Latenzen einer Übertragungsrichtung werden in [CRC+18] analysiert. Diese können bereits sehr gering sein – sofern das drahtlose Medium nicht hoch ausgelastet ist, also im Idealfall. Die Herausforderung sei es dagegen, die Zuverlässigkeit und Jitter bei hoher Auslastung des Netzwerks zu verbessern. Dazu wird unter anderem eine Adaption des TAS aus 802.1Qbv auf WiFi vorgeschlagen.

Einige Arbeiten betrachten die aktuelle oder zukünftige Ermöglichung spezifischer Anwendungsfälle über WiFi. In [KRD+18] werden mit einer LED-Matrix und einem kinematischen Kontrollsystem zwei Anwendungen demonstriert, die durch Verbesserungen des WiFi-Stack profitieren. In [IJ18] wird der Anwendungsfall analysiert, Online-Spiele über WiFi zu konsumieren. Mit denselben drahtlosen Mikrocontrollern, dem ESP32, wie in dieser Arbeit misst [WMW20] mit variierender Anzahl an Geräten und Rahmen pro Sekunde. Es wird ebenfalls die Latenz einer Übertragungsrichtung gemessen; allerdings ist der Anwendungsfall zugeschnitten auf digitale Musikinstrumente, die Latenz der Anwendung wird mitgemessen. Die Anwendung in [SMBC21] ist die deterministische Kommunikation kollaborativer Roboter, welche über ein Publish/Subscribe-System miteinander kommunizieren. Durch geeignete Zuordnung der Topics innerhalb des „Robot operating system 2“ auf Prioritäten innerhalb des Wireless TSN-Netzwerks sowie der Nutzung des TAS werden wichtige Datenpakete mit deutlich geringerer Latenz und Jitter zugestellt; die Zusammenarbeit der Roboter wird so deutlich verbessert.

4 Systemmodell und Problemstellung

In diesem Kapitel werden zunächst die Bestandteile des Systems erläutert, im Anschluss werden die grundlegenden Annahmen der betrachteten Netzwerke geschildert und die Problemstellung erneut erläutert. Zuletzt folgen einige Definitionen der *Key Performance Indicators (KPI)*, also Kennzahlen der später zu messenden Charakteristika der Informationsübermittlung.

4.1 Systemmodell

Beteiligt an der Informationsübermittlung sind jeweils ein Sender und ein Empfänger als Endpunkte, die durch ein Netzwerk verbunden sind. Diese Informationsübermittlung von Sender zu Empfänger bezeichnen wir auch als *Informationsfluss* oder kurz *Fluss*. Dieser Fluss beginnt beim Sender, und wird durch das Netzwerk zum Empfänger übermittelt. Insbesondere kann eines der Geräte – Sender oder Empfänger – ein WiFi-Klient sein, dann ist auch ein WiFi Access Point als Gegenstück des drahtlosen Links Teil des Netzwerks.

Ziel ist, die Zeitdifferenz zwischen Empfangszeitpunkt und Sendezeitpunkt, jeweils an den beiden Endpunkten des Netzwerks, zu messen.

4.1.1 Auswahl der Hardware

In vielen Fällen kommen für derlei Messungen spezialisierte Netzwerkkarten oder Field Programmable Gate Arrays (FPGAs) zum Einsatz. In zukünftigen Industrieanwendungen werden – insbesondere auf Seiten der WiFi-Klienten – häufig keine spezialisierten Netzwerkkarten zum Einsatz kommen, sondern gewöhnliche und kostengünstige Mikrocontroller. In diesem Sinne kommen in dieser Arbeit als Endpunkte ebenfalls solche Mikrocontroller zum Einsatz.
Dabei handelt es sich nicht um spezialisierte Messinstrumente, sondern um „off-the-shelf“ Hardware, die sich so auch nah an Praxisanwendungen befindet. Insbesondere die Wahl der WiFi-Klienten soll durch den Messaufbau so wenig wie möglich eingeschränkt werden, sodass der Messaufbau geeignet ist, Messungen mit WiFi-fähigen Mikrocontrollern so flexibel wie möglich durchzuführen.

Beispiele hierfür sind die in Unterabschnitt 2.3.2 vorgestellten Mikroprozessoren i.MX RT 1062 von NXP als Ethernet-fähiger Endpunkt sowie die Mikrocontroller der ESP32-Reihe von Espressif als WiFi-fähiger Endpunkt.

Im folgenden Abschnitt wird auf den Aufbau des übermittelnden Netzwerks eingegangen.

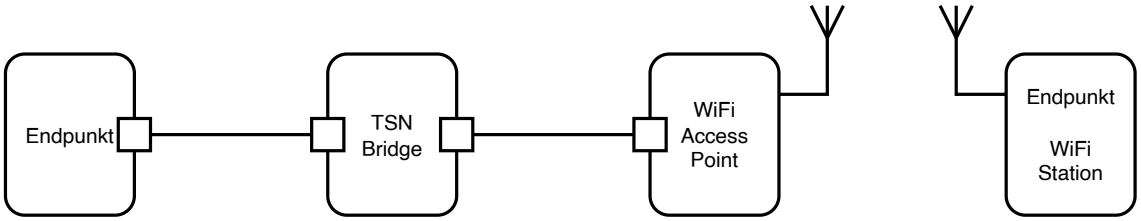


Abbildung 4.1: Exemplarische Darstellung eines simplen Netzwerks mit einem WiFi Link

4.1.2 Aufbau des Netzwerks

Time Sensitive Networking ist eine Technologie der Schicht zwei des ISO/OSI-Referenzmodells – daher wird lediglich die Übermittlung in Netzwerken der Schicht zwei betrachtet.

Im Laufe der Arbeit wird ausschließlich Unicast-Kommunikation betrachtet: Je Fluss ist also genau ein Sender sowie genau ein Empfänger beteiligt. Auf dem Kommunikationspfad können sich mehrere Bridges befinden, außerdem kann ein drahtloser Link beteiligt sein, konkret eine Übertragung per WiFi.
Konzeptionell sind die folgenden Bestandteile Teil des Netzwerks:

- *Endpunkte* sind entweder per Ethernet oder per WiFi mit dem Netzwerk verbunden, sie nehmen eine Rolle als Sender oder Empfänger wahr. WiFi-Geräte können optional Unterstützung für Enhanced Distributed Channel Access bieten (siehe Unterabschnitt 2.1.3).
- *Bridges* arbeiten auf der Schicht zwei – sie sind Netzwerkgeräte, die zwei oder mehrere Netzwerkgeräte miteinander verbinden, indem sie Datenrahmen zwischen verschiedenen Netzsegmenten weiterleiten.
  Eine spezielle Form davon sind *TSN-Bridges*, sie unterstützen bei dieser Weiterleitung besondere Funktionalität zur Steuerung des zeitlichen Verhaltens, allen voran den Time Aware Shaper (IEEE 802.1Qbv) (siehe Unterabschnitt 2.2.2).
- *WiFi Access Points (APs)* agieren als zentraler Punkt für WiFi-Geräte und können Datenrahmen zwischen Drahtlosgeräten oder zwischen Drahtlosgerät und drahtgebundenem Netzwerk übermitteln.

Verkehr, der sich Ressourcen – etwa die selbe Warteschlange der Bridge oder das Frequenzband der Drahtlosübertragung – mit einem betrachteten Fluss teilt, bezeichnen wir als *Cross-Traffic*. Dieser belegt Bandbreite und verzögert anderen Verkehr, sodass er möglicherweise negative Auswirkungen auf die Übertragungseigenschaften des eigentlichen Flusses hat. Durch Wahl geeigneter Schedules der Endpunkte sowie der TSN-Bridges können die Übertragungseigenschaften verbessert werden.

Für die betrachteten Netzwerke gelten jeweils die folgenden Annahmen:

1. Die Netzwerktopologie ist vollständig bekannt.
2. Die Netzwerktopologie ist für jede Messung statisch. Innerhalb eines spezifischen Versuchsaufbaus kommen keine neuen Sender oder Empfänger hinzu.
3. Für verschiedene Versuchsaufbauten lassen sich verschiedene Netzwerktopologien aufbauen.
4. Es besteht vollumfängliche Kontrolle über die Schedules der TSN-Bridges.

Da in erster Linie Latenz und Jitter des drahtlosen Links betrachtet werden sollen, kommen neben einem drahtgebundenen und einem drahtlosen Endpunkt meist nur eine (TSN-)Bridge und ein WiFi Access Point zum Einsatz. Abbildung 4.1 zeigt exemplarisch ein solches Netzwerk.

4.1.3 WiFi

Die Messungen der Latenz erfolgen in realen Systemen in realen Umgebungen. Daher muss die folgende, der Realität entsprechende, Annahme getroffen werden.

Annahme: Interferenz

Es kann nicht garantiert werden, dass das vom WiFi-Kanal genutzte Frequenzband ausschließlich für die Nutzung durch WiFi-Geräte zur Verfügung steht. Einerseits kann eine Nutzung durch andere, nicht zum Versuchsaufbau gehörende, aber in Empfangsreichweite liegende WLAN-Stationen bzw. WLAN-Access-Points nicht ausgeschlossen werden. Andererseits ist auch eine Belegung des Kanals durch Drahtloskommunikation von Geräten, die andere Standards als 802.11 verwenden, nicht auszuschließen.

4.2 Problemstellung

Zusammengefasst ist die Problemstellung der Entwurf und die Implementierung eines Systems zur präzisen Messung von Ende-zu-Ende-Latenz und Jitter von Kommunikationspfaden, die eine Übertragung per WiFi beinhalten.

Derzeit wird in der IEEE Task Group for 802.11be (TGbe) am Standard der nächsten Generation von WiFi, 802.11be – im Markennamen WiFi 7 – gearbeitet. WiFi 7 soll eine deutlich bessere Unterstützung für Echtzeitanwendungen liefern.

Innerhalb dieses Kontexts werden im Verlauf der Arbeit zunächst mithilfe einer vergleichenden Messung der Latenz rein über Ethernet ein Vergleich zu den gemessenen Werten der WiFi-Übertragung hergestellt.

Eine oft getroffene Annahme lautet, dass die Ende-zu-Ende-Latenz für beide Übertragungsrichtungen – also von Sender zu Empfänger, und dann der umgekehrte Weg durch das Netzwerk – identisch sind. Diese Annahme wird im Verlauf der Arbeit überprüft.

Schließlich wird die Effektivität des im technischen Hintergrund ausführlich vorgestellten Verfahrens EDCA (siehe Unterabschnitt 2.1.3) im Hinblick auf die Priorisierung wichtiger Kommunikationsflüsse bei hoher Auslastung des drahtlosen Mediums analysiert.

Und zuletzt wird aktuelle Forschung zur Weiterentwicklung des WiFi-Standards aufgegriffen, indem einerseits eine Kompensation der durch den WiFi-Link stark erhöhten variablen Paketverzögerung mithilfe des Time-Aware Shapers (siehe Unterabschnitt 2.2.2) und andererseits die Möglichkeit des Schedulings der Übertragungen mehrerer Stationen evaluiert wird.

Wo Lösungen erst vorgeschlagen, aber noch nicht in kommerziell verfügbaren Produkten integriert sind, werden möglichst vergleichbare Annäherungen konstruiert.

Um diese Messungen durchzuführen, besteht der Bedarf, die Sende- und Empfangszeitpunkte der versendeten Pakete präzise zu messen. Wie zu Beginn der folgenden Evaluation gezeigt wird, bewegt sich die Ende-zu-Ende Latenz, sobald eine Übertragung per WiFi durchgeführt wird, im Bereich einiger hundert Mikrosekunden; die Ende-zu-Ende Latenz bei Übertragung rein über Ethernet kann sich gar im Bereich weniger Mikrosekunden bewegen.
Daher ist für aussagekräftige Messungen eine Messauflösung im Bereich von Nanosekunden erforderlich. Bei Verwendung von *Off-the-Shelf* Hardware und einer reinen Implementierung des Versuchs in Software ist die Messgenauigkeit nicht ausreichend.

Zusammengefasst gestaltet sich also in erster Linie die Anfertigung dieser Messungen zur späteren Evaluierung als zu lösendes Problem.

4.3 Key Performance Indicators

Im folgenden Abschnitt werden die für den Verlauf der Arbeit relevanten Begrifflichkeiten definiert. Diese dienen als Key Performance Indicators (KPIs) für die Beurteilung der Versuchsergebnisse.

4.3.1 Latenz

Laut Definition in IEEE 802.1Q-2014 [14] bezeichnet die Latenz (im englischen auch *frame delay*) die Verzögerung, die ein Frame im Verlauf der Ausbreitung durch das Netzwerk erfährt. Diese sei zu messen als die Zeitdifferenz zwischen dem Zeitpunkt, an dem der Frame einen bekannten Referenzpunkt des Frames das erste Mal passiert bis zu dem Zeitpunkt, an dem dieser Referenzpunkt das zweite Mal passiert wird.

Für den Verlauf der Arbeit wird diese durch Definition (4.3.1) etwas vereinfacht, vor allem, um eine bessere Messbarkeit der Latenz zu erzielen.

Definition 4.3.1 (Latenz)
Die Latenz der Übertragung einer Information bezeichnet die Zeitdifferenz zwischen der dem Zeitpunkt der Übergabe an eine spezifizierte Protokollschicht p_1 auf dem Sender bis zum Zeitpunkt der Übergabe derselben Information an die spezifizierte Protokollschicht p_2 auf dem Empfänger.

In Abbildung 4.2 ist exemplarisch die Latenz veranschaulicht: Möchte man die Zeitdifferenz zwischen Zeitpunkt der Übergabe an die Sicherungsschicht (p_1) beim Sender mit dem Zeitpunkt der Übergabe an die Anwendungsschicht (p_2) auf dem Empfänger betrachten, ergibt sich die Latenz l als $l = t_2 - t_1$ wie dargestellt.

Die gewählte Definition der Latenz ermöglicht eine Messung dieser auf einer Vielzahl von Systemen: Es genügt, die Zeitpunkte t_1, t_2 erfassen zu können. (Dabei muss darauf geachtet werden, dass dies mit der selben Uhr oder mit synchronisierten Uhren geschieht – dieses Problem wird im Rahmen des Entwurfs angegangen.) Hat man also Sender und Empfänger zur Verfügung, deren Netzwerkstack nicht quelloffen ist, ist eine Erfassung des Zeitpunktes eines Referenzpunktes eines Frames fast ausgeschlossen – wählt man dann aber beispielsweise p_1, p_2 jeweils als Punkt zwischen Anwendungs- und Transportschicht und erfasst die Zeitpunkte diese Übergaben, kann die so definierte Latenz gemessen werden.

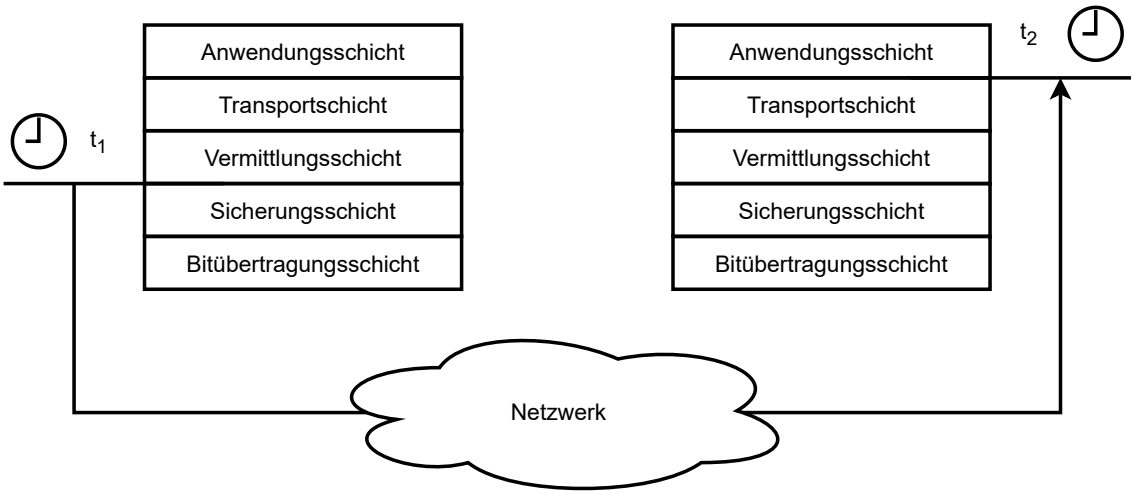


Abbildung 4.2: Exemplarische Darstellung der Latenz

4.3.2 Latenzvarianz

Die Varianz σ^2 bezeichnet allgemein die Summe der quadrierten Abweichungen zum Erwartungswert μ der Ergebnisse eines Zufallsexperiments.

Definition 4.3.2 (Latenzvarianz)
Für eine Sequenz L mit k Latenzmessungen $L = (m_0, m_1, ..., m_{k-1})$ *bezeichnet die* Latenzvarianz *die Varianz der Latenz. Sie berechnet sich als*

$$\text{Latenzvarianz}(L) = \sigma^2 = \frac{1}{k} \sum_{m_i \in L} (m_i - \mu)^2$$

unter Zuhilfenahme des Erwartungswertes μ*, für den gilt:*

$$\mu = \frac{1}{k} \sum_{m_i \in L} m_i$$

Die Latenzvarianz σ^2 gibt schnell Auskunft darüber, wie „vorhersehbar“ sich die Kommunikation über den gemessenen Kommunikationspfad verhält: Sie ermöglicht es, die Streuung der Latenzzeiten um den Mittelwert zu quantifizieren. Wenn die Latenzvarianz hoch ist, bedeutet dies, dass die Latenzzeiten der Messreihe stark um den Mittelwert streuen; eine niedrige Latenzvarianz dagegen weist auf eine geringe Streuung hin. Eine hohe Heterogenität der Latenzen geht also einher mit einer hohen Latenzvarianz.

Die Standardabweichung σ ergibt sich aus der Wurzel der Varianz; sie gibt die durchschnittliche Differenz der Latenzen zum Mittelwert wieder. Für die Liste $L = (m_0, m_1, ..., m_{k-1})$ berechnet sie sich als:

$$\text{Standardabweichung}(L) = \sigma = \sqrt{\frac{1}{k} \sum_{m_i \in L} (m_i - \mu)^2}$$

4.3.3 Jitter

Jitter beschäftigt sich mit der Schwankung der Zeitintervalle zwischen Signalen. Für die konkrete mathematische Definition von Jitter gibt es verschiedene Ansätze, das Wort wird in anderen Kontexten mit unterschiedlichen Bedeutungen eingesetzt – im Kontext von Computernetzen bezeichnet Jitter oft einfach die Latenzvarianz. Der IEEE Standard Jitter and Phase Noise [21b] liefert verschiedene Definitionen. Die im Verlauf der Arbeit verwendeten Definitionen werden nun eingeführt.

Definition 4.3.3 (1-Jitter)
Der 1-Jitter *bezeichne die Differenzen aufeinanderfolgender Latenzmessungen.*

Für k Latenzmesungen $L = (m_0, m_1, m_2, ..., m_{k-2}, m_{k-1})$ ergeben sich $k - 1$ 1-Jitter-Werte $J_1 = ((m_1 - m_0), (m_2 - m_1), ..., (m_{k-1} - m_{k-2}))$.

Beispiel. Für $L = (1\,\text{ms}, 2\,\text{ms}, 1\,\text{ms}, 1\,\text{ms})$ ergibt sich $J_1 = (1\,\text{ms}, -1\,\text{ms}, 0\,\text{ms})$.

Definition 4.3.4 (Min-Max-Jitter)
Der Min-Max-Jitter bezeichne die Spannweite der gemessenen Latenzen, berechnet als $J_{min-max} = max(L) - min(L)$.

Beispiel. Für $L = (1\,\text{ms}, 5\,\text{ms}, 2\,\text{ms})$ ergibt sich $J_{min-max} = 5\,\text{ms} - 1\,\text{ms} = 4\,\text{ms}$.

4.3.4 Zuverlässigkeit

Oft möchte man nicht die reine Latenz betrachten, sondern die Wahrscheinlichkeit, dass eine Information den Empfänger rechtzeitig erreicht. Dazu eignet sich die Zuverlässigkeit.

Definition 4.3.5 (Zuverlässigkeit)
Zuverlässigkeit bezeichnet den Anteil der übermittelten Pakete, die innerhalb der vorgegebenen zeitlichen Beschränkung erfolgreich an die Zielschicht des Empfängers zugestellt wurden, an allen gesendeten Paketen. [3GP21, sinngemäß zu engl. reliability]

Für eine untere Zeitschranke t_{min} und eine obere Zeitschranke t_{max} berechnet sich die Zuverlässigkeit Z also als

$$Z = \frac{\text{Anzahl der Pakete mit Ankunftszeitpunkt } t_{min} < t < t_{max}}{\text{Anzahl aller Pakete}}$$

In diesem Kapitel werden zunächst die Bestandteile des Systems erläutert, im Anschluss werden die grundlegenden Annahmen der betrachteten Netzwerke geschildert. Zuletzt folgen einige Definitionen der *Key Performance Indicators (KPI)*, also Kennzahlen der später zu messenden Charakteristika der Informationsübermittlung.

5 Entwurf

In diesem Kapitel wird der Entwurf eines Messsystems zur Erfassung der Latenzen L gemäß Definition 4.3.1 beschrieben. Die beschriebenen Key Performance Indicators 1-Jitter, Min-Max-Jitter sowie Latenzvarianz und die Standardabweichung können aus den Latenzmessungen berechnet werden. Das Kapitel beginnt zunächst mit einem Entwurf eines Basisaufbaus, der in der Lage ist, die Ende-zu-Ende Latenzen einer Übertragung über das Netzwerk zu erfassen. Dies allein genügt jedoch nicht, um auch die Veränderung der Übertragungseigenschaften durch die Verwendung des Time-Aware Shapers quantifizieren zu können.

Daher wird im weiteren Verlauf des Entwurfs ein Verfahren zur Synchronisierung mehrerer Mikrocontroller eingeführt, um mehrere gleichzeitige Sendevorgänge auslösen zu können. Darauf aufbauend wird eine Verfahren entworfen, die Mikrocontroller ohne die Notwendigkeit einer PTP-Implementierung mit der Periode einer TSN-Bridge zu synchronisieren.

Zuletzt werden die entworfenen Funktionen des Systems jeweils in verschiedenen Kombinationen zusammengefügt. Dazu werden mehrere konkrete Versuchsaufbauten motiviert und erläutert. Diese dienen jeweils der experimentellen Ermittlung einer der in der Problemstellung beschriebenen Fragestellungen.

5.1 Einleitung

Zur Berechnung der Latenz $l = t_2 - t_1$ gemäß Definition 4.3.1 sind nur die beiden Zeitstempel t_1, t_2 notwendig. Die Schwierigkeit liegt darin, diese präzise erfassen zu können. Eine erste Herangehensweise an die Problematik wäre, den Sendezeitpunkt auf dem Sender und den Empfangszeitpunkt auf dem Empfänger zu erfassen; die Latenzen könnten nach Übertragung der Zeitstempel an einen zentralen Server berechnet werden.

Abbildung 5.1 demonstriert das grundlegende Problem, ausgelöst durch nicht oder nicht genau synchronisierte Uhren: Zum gleichen Betrachtungszeitpunkt ist nach Stand des Senders die Systemzeit t_s = 9:41:299, aus Sicht des Empfängers t_e = 9:41:300. Es ergibt sich eine Zeitdifferenz von $\delta t = 1$ ms. Folglich wären alle Latenzmessungen ebenfalls falsch, eine wahre Latenz $l = 5$ ms würde gemessen als $l' = 6$ ms. Die Genauigkeit der Messergebnisse wäre also immer abhängig von der Genauigkeit der Synchronisierung der Uhren. Eine Zeitsynchronisierung mittels Network Time Protokol (NTP) führt zu einer Zeitabweichung δt in Höhe von mehren hundert Mikrosekunden bis etwa einer Millisekunde [Mil94]. Ein kurzer Vorausblick auf die Evaluierung in Kapitel 7 zeigt, dass sich die gemessene Ethernet-Latenz im Bereich einiger 10 µs, die Latenz der Übertragung über WiFi im Bereich einiger 100 µs bewegt. So erhaltene Messungen hätten folglich keine Aussagekraft.

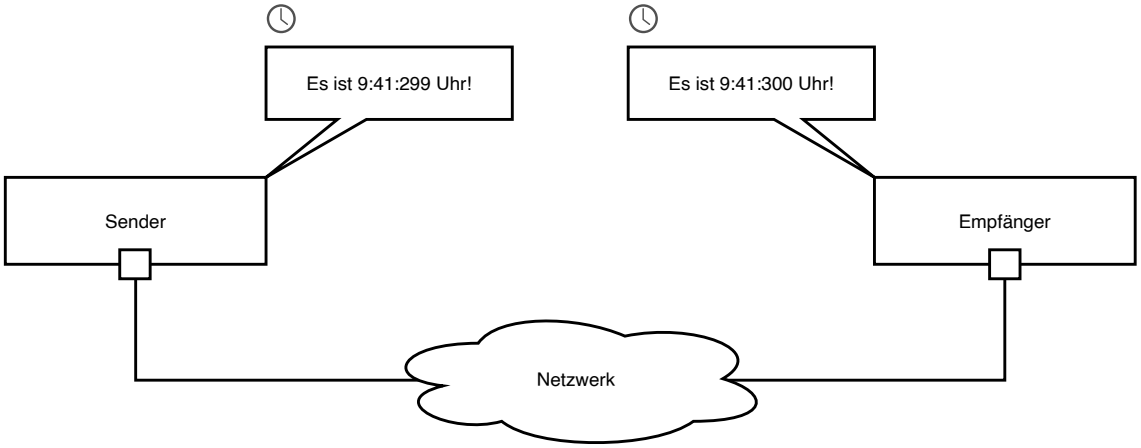


Abbildung 5.1: Problematik unterschiedlicher Systemzeiten

5.1.1 Ansätze

Um präzise Messungen anfertigen zu können, gibt es grundsätzlich zwei Herangehensweisen.

1. *Synchronisierung der Uhren.* Das Problem der unterschiedlichen Systemuhrzeiten wird durch eine möglichst genaue Synchronisierung derselben mitigiert. Die Zeitstempel werden dann auf den beiden verschiedenen Endpunkten erfasst.
2. *Messung mit nur einer Uhr.* Ein gängiger Ansatz zur Betrachtung von Latenzen ist die Betrachtung der Round-Trip Latenz. Beispielsweise das weit verbreitete Tool „ping“ misst Hin- und Rückweg der Kommunikation, so dass beide Zeitstempel – Sendezeitstempel der Anfrage und Empfangszeitstempel der Antwort – mithilfe der selben Systemzeit auf dem selben Gerät erfasst werden können.

Eine Synchronisierung der Uhren hat dabei einige Schwierigkeiten. Zwar lässt sich durch Verwendung des Precision Time Protocols (PTP) eine genauere Zeitsynchronisation als durch NTP herstellen, allerdings besteht dafür ein hoher Implementierungsaufwand. Weiterhin sind besonders die WiFi-Plattformen nicht für PTP geeignet, da sie kein Hardware-Timestamping unterstützen; die Zeitstempel müssten folglich in Software erfasst werden.

In [HJA+21] wird die Synchronisationsgenauigkeit einer Zeitsynchronisation mithilfe von PTP über WiFi evaluiert. Der Synchronisationsfehler zwischen zwei Endpunkten liegt dabei in nur 90% der Fälle unter 50 µs, bei einer Standardabweichung von $\sigma = 20{,}12$ µs. Insgesamt würde diese Methode also keine zufriedenstellende Messgenauigkeit erzielen.

Darüber hinaus basiert die Synchronisation mithilfe des Precision Time Protocols auf der Annahme, dass Sende- und Empfangslatenz identisch sind [WAA+15] – diese Annahme müsste also bereits verwendet werden. Sofern diese nicht erfüllt wäre, würden die Messwerte dadurch bereits verfälscht.

Aus den genannten Gründen wird der Entwurf so konstruiert, dass keine Zeitsynchronisierung mehrere Geräte erforderlich ist. Messungen mit nur einer Uhr sind dagegen ebenfalls schwierig – sofern nicht nur die Round-Trip Zeit gemessen werden soll, da Sender und Empfänger im

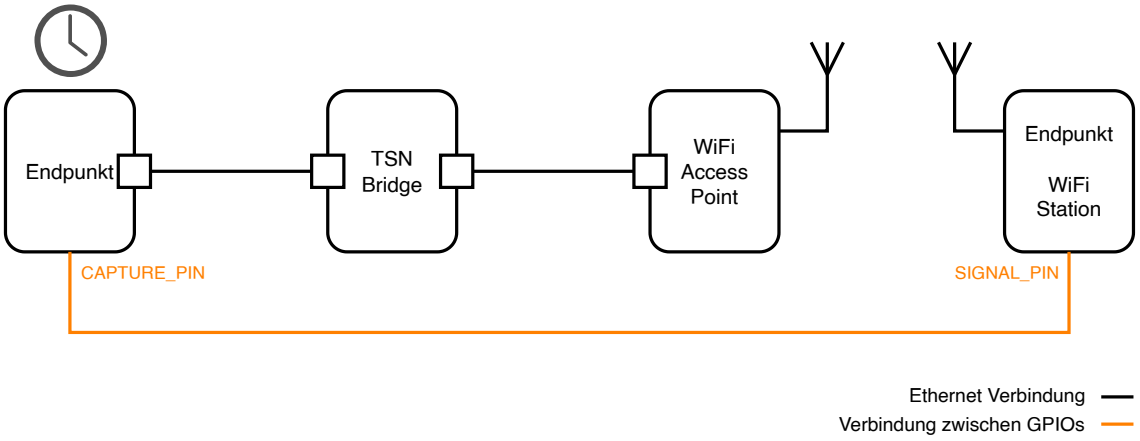


Abbildung 5.2: Basisentwurf mit 1 Sender, 1 Empfänger

Allgemeinen keine gemeinsame Uhr besitzen. Im folgenden Abschnitt wird ein Ansatz eingeführt, der die Zeitstempel mit Verwendung von einer Uhr erfasst, dabei aber nicht auf die Messung von Round-Trip-Latenzen beschränkt ist.

5.2 Basisentwurf

Die Kernidee des Entwurfes wird in Abbildung 5.2 dargestellt: Beide Zeitstempel werden auf dem selben Mikrocontroller mithilfe der selben Uhr erfasst. Der Mikrocontroller „Teensy 4.1“ mit dem verwendeten Mikrochip „i.MX RT1062“ von NXP ist als einer der Endpunkte – der per Ethernet verbundene – gleichzeitig Messinstrument und für die Erfassung beider Zeitstempel zuständig. Von zwei zu erfassenden Zeitstempeln wird einer ohnehin auf dem Teensy erfasst. Der zweite Zeitstempel wird nun erfasst, indem der zweite Endpunkt einen GPIO-Pin umschaltet. Der Teensy speichert den Zeitpunkt des Umschaltens dieses Signals. Dieser Ansatz ist aufgrund der geringen physischen Distanz zwischen Sender und Empfänger möglich, sodass diese per Kabel verbunden werden können.

Kommt der Teensy als Sender zum Einsatz, erfasst er zunächst den Sendezeitpunkt t_1. Nach Ankunft der Übertragung an der Protokollschicht p_2 auf dem (WiFi-)Empfänger ändert dieser den Pegel eines verbundenen General Purpose Input/Output (GPIO)-Pins. Dieses Ereignis wird auf dem Teensy erfasst, mit der selben Uhr wird so t_2 registriert.

Ist der Teensy Empfänger, zeigt zunächst der Sender direkt vor Beginn der Übergabe an die Protokollschicht p_1 den unmittelbar bevorstehenden Zeitpunkt t_1 durch Wechsel des Signals an, dieser Zeitpunkt wird durch den Teensy erfasst. t_2 ist der Empfangszeitstempel auf dem Teensy, der wiederum mit der selben Uhr wie t_1 erfasst werden kann.

5.2.1 Festlegung der Messpunkte

Die gewählte Definition der Latenz in 4.3.1 lässt Flexibilität bei der Auswahl der konkreten Messpunkte – im folgenden Unterabschnitt wird daher festgelegt, welche Zeitdifferenz gemessen wird.

Teensy

Der Teensy hat die Fähigkeit, Sende- und Empfangszeitstempel in Hardware zu erfassen (siehe Unterabschnitt 2.3.1). Diese Möglichkeit wird genutzt, da sie eine höhere Genauigkeit bieten kann, als die Zeitstempel softwaregestützt zu erfassen.

Für ausgehende Rahmen kann ein Bit zur Anforderung eines Sendezeitstempels im Sendepuffer gesetzt werden. Für alle Rahmen, bei denen dieses Bit gesetzt ist, speichert erfasst die Sicherungsschicht den Sendezeitstempel im Register ENET_ATSTMP. Für alle eingehenden Rahmen erfasst die Sicherungsschicht den Zeitpunkt des *Start of Frame Delimiters*, und speichert diesen im Empfangspuffer. Einige Implementierungsdetails dazu werden im Kapitel Implementierung (Kapitel 6) erläutert.

ESP32

Die Auswahl des Zeitpunktes auf Seiten des ESP32 gestaltet sich etwas vielschichtiger. Die Produkte der ESP32-Serie von Espressif lassen sich mithilfe des *ESP IoT Development Frameworks (ESP-IDF)* programmieren. Darin enthalten sind bereits modifizierte Verianten des Betriebssystems FreeRTOS sowie des IP-Stacks lwIP (lightweight Internet Protocol). Anwendungen für den ESP32 bauen auf diesen Komponenten auf. Darüber hinaus ist sind die WiFi-Treiber (die WiFi-Treiber unterscheiden sich für verschiedene ESPs) nicht quelloffen.

Daher wird an dem Messpunkt angesetzt, an dem auch Anwendungen für den ESP aufbauen würden: Die Empfangszeitstempel werden direkt vor- bzw. nach Aufruf der Socket-API gemessen. In Senderichtung wird der GPIO-Pin umgeschaltet, unmittelbar danach erfolgt der Aufruf der `sendto`-Funktion. In Empfangsrichtung erfolgt der Aufruf der blockierenden `recvfrom`-Funktion, unmittelbar nach deren Ende wird der Signalpegel des GPIO-Pins umgeschaltet.

Das hat zur Folge, dass die Verzögerung, die die versandte Nutzlast zwischen Aufruf der Socket-API bis zur Übergabe an die Radioschnittstelle erfährt, Teil der gemessenen Latenz sind. Dazu zählen beispielsweise Kontextwechsel zwischen FreeRTOS-Tasks, Zeitbedarf für die Allokation von Buffern bzw. die Kopie der Rahmen in selbige, oder die Dauer der Verarbeitung im lwIP-Task. Das hat Vor- und Nachteile: Einerseits wird die Aussagekraft über die reine Netzwerklatenz der WiFi-Übertragung geschmälert. Andererseits gestaltet sich die Implementierung simpel, so dass der ESP nur Stellvertreter für eine Kategorie an WiFi-fähigen Mikrocontrollern ist – der Aufruf einer Socket-API sollte auf einer Vielzahl von Mikrocontrollern möglich sein, so dass sich mit dem gewählten Ansatz später mehrere Mikrocontroller vergleichen ließen. Im Verlauf der Arbeit wird beispielsweise eine Kompensation des Jitters betrachtet – dort ist dann auch der durch den Netzwerkstack verursachte Jitter enthalten. Zuletzt spricht dafür, dass die Messungen dann näher an den Anwendungen liegen, die mit einem ESP32 umgesetzt würden – also beispielsweise die Erfassung der Latenz bis zu dem Zeitpunkt, zu dem frühestens mit der Steuerung einer Maschine

begonnen werden könnten. Somit ist diese Latenz auch die relevante für die Performanz der Anwendung. Einordnend in die Definition der Latenz (4.3.1) wird also der Zeitpunkt zwischen Transportschicht und Anwendungsschicht gemessen.

5.2.2 Input Capture

Essentiell für das beschriebene Verfahren ist, dass der Teensy in der Lage ist, den Zeitpunkt des durch den ESP32 ausgelösten Signalwechsels genau zu erfassen. Die i.MX RT 1060-Prozessoren ermöglichen eine hardwaregestützte – und dadurch präzise – Erfassung externer Ereignisse. Entscheidend ist dabei, dass die selbe Uhr verwendet wird, die auch die Sende- bzw. Empfangszeitstempel der Ethernet-Rahmen erfasst.

Unter dem Begriff *Input Capture* wird im Technical Reference Manual (TRM) die Funktionalität des Mikrocontrollers aufgeführt, den Zeitpunkt der steigenden, fallenden oder wechselnden Taktflanke eines Eingangssignales erfassen zu können [Sem21, S. 2118]. Die Erfassung der Zeitstempel wird im sogenannten *Timer Compare Status Register* (ENET_TCSR) konfiguriert, der erfasste Zeitstempel kann aus dem *Timer Compare Capture Register* (ENET_TCCR) ausgelesen werden. Der Teensy verfügt über eine Ethernet-Uhr mit einer Frequenz von 25 MHz, was einer Messauflösung von 40 ns entspricht.

Im Entwurf des Messsystems an sich wird noch genauer auf die Datenstruktur zum Speichern der Zeitstempel eingegangen. Bevor jedoch die konkreten Programmabläufe entworfen werden, wird zunächst noch ein Konzept zur Synchronisierung mehrere Mikrocontroller eingeführt.

5.3 Synchronisation

Der folgende Entwurfsschritt stellt zuerst eine Synchronisation mehrerer Mikrocontroller untereinander her, so dass gleichzeitig Sendevorgänge ausgelöst werden können. Mithilfe der Synchronisation lässt sich aber auch umgekehrt vermeiden, dass mehrere Mikrocontroller gleichzeitig senden. Im zweiten Teil des Abschnitts wird zusätzlich vorgestellt, wie die Synchronisation an die Periode der TSN-Bridge ohne Notwendigkeit einer PTP-Implementierung erfolgen kann.

5.3.1 Gleichzeitige Informationsübermittlung

In der Realität werden Vorgänge der Informationsübermittlung oft durch ein externes Ereignis ausgelöst. Dies hat zur Folge, dass mehrere Geräte zum gleichen Zeitpunkt auf Anwendungsebene einen Sendevorgang auslösen. Beispiele für deterministische aperiodische Kommunikation sind etwa mehrere drahtlose Bewegungsmelder, die den selben Bereich abdecken – wird dieser durch eine Person betreten, beginnt die Übertragung auf allen Geräten zum selben Zeitpunkt. Auch für periodische deterministische Kommunikation lässt sich ein solches Beispiel leicht finden: Etwa Publish / Subscribe-Systeme, auf denen mehrere Geräte zum selben Zeitpunkt eine Nachricht empfangen, und dann ebenfalls zum selben Zeitpunkt die Antwort veranlassen.

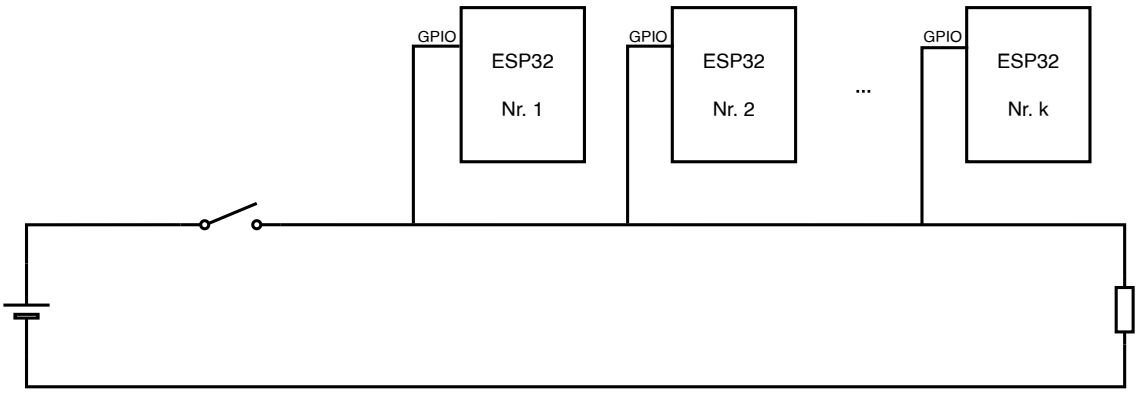


Abbildung 5.3: Basis der Synchronisierung der Mikrocontroller

Die Mikrocontroller teilen sich jedoch das drahtlose Medium, so dass vermutet werden kann, dass die Latenz dadurch ansteigt. Der Effekt eines solchen Szenarios soll gemessen werden können. Zu diesem Zwecke ist ein Ziel des Entwurfs, mehrere Mikrocontroller derart zu synchronisieren, dass die Anwendungsebene zum gleichen Zeitpunkt mit der Informationsübermittlung beginnt.

Grundsätzlich wäre es auch hier möglich, eine gemeinsame Uhrzeit aller Mikrocontroller herzustellen; dann könnten beispielsweise alle Anwendungen zu einem gemeinsam vereinbarten Zeitpunkt die Socket-API aufrufen. Für die zukünftige Entwicklung von Wireless TSN ist die Herstellung einer gemeinsamen Uhrzeit zwar unabdingbar, liegt aus den in Abschnitt 5.1 diskutierten Gründen aber außerhalb des Rahmens dieser Arbeit. Im Laufe der Evaluation wird außerdem festgestellt, dass eine Software-Implementierung mit der beschriebenen Erfassung der Zeitstempel eine sehr schlechte Genauigkeit liefern würde.

Stattdessen erfolgt die Synchronisation mithilfe der GPIO-Pins des Mikrocontroller. Abbildung 5.3 zeigt diesen Ansatz: Alle Mikrocontroller sind mit dem selben physischen Signal verbunden. Betätigt man nun den Schalter, registrieren alle Mikrocontroller gleichzeitig diese steigende Taktflanke. Etwa indem eine Interrupt Service Routine bei steigender Taktflanke dieses GPIO-Pins registriert wird, kann die Anwendung dann z.B. den Versand des Pakets veranlassen.

5.3.2 Time-Aware Shaper

Für den Empfänger kann es wünschenswert sein, den Ankunftszeitpunkt eines Rahmens möglichst genau zu kennen – also einen möglichst hohen Anteil der Rahmen innerhalb eines eng festgelegten Zeitraumes zu erhalten. Zur Erinnerung: Die Zuverlässigkeit gemäß Definition 4.3.5 bezeichnet den Anteil an allen Paketen, die zum Zeitpunkt t mit $t_{min} \leq t \leq t_{max}$ zugestellt werden.

Die Übertragung per WiFi führt aufgrund von Kollisionen und dem hohen Einfluss des Random Backoff Timers jedoch zu einem kaum eng eingrenzbaren Ankunftszeitpunkt. Der Time-Aware Shaper lässt sich daher einsetzen, um die hohe Latenzvarianz des WiFi-Links zu kompensieren. So lässt sich eine hohe Zuverlässigkeit für einen kleiner gewählten Zeitraum erreichen. Dazu wird angenommen, dass sich alle Rahmen eines betrachteten Flusses in der selben Verkehrsklasse befinden.
Beispiel. Das Gate dieser Verkehrsklasse ist innerhalb einer Periodendauer von 50 ms nur ein

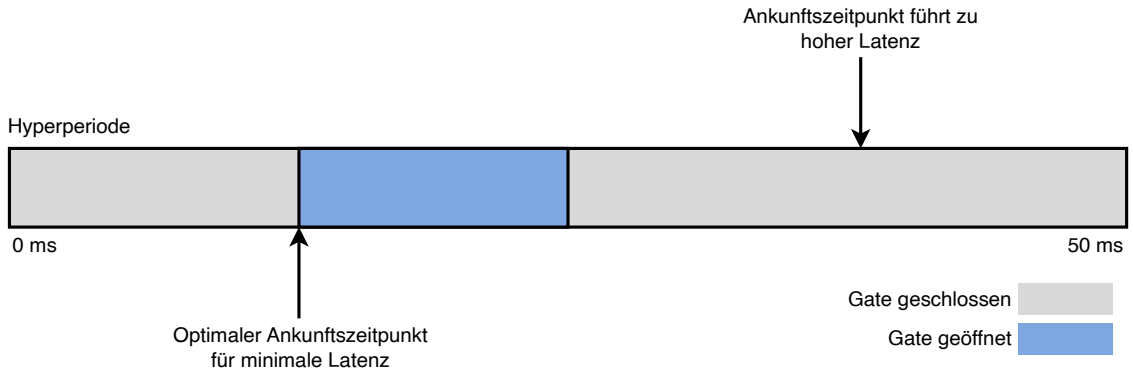


Abbildung 5.4: Problematik hoher Latenz bei zufälligem Ankunftszeitpunkt innerhalb der Periodendauer

Mal für 2 ms geöffnet. Der Jitter des WiFi-Links wird dadurch kompensiert, der TAS versendet Pakete dieser Verkehrsklasse nur innerhalb des vorgegebenen Zeitraumes. Da die Latenzvarianz der nachfolgenden Übertragung per Ethernet bedeutend kleiner ist als die des WiFi-Links, wird die Latenzvarianz auf diese Art verringert.

So kann eine verbesserte Zuverlässigkeit erzielt werden – allerdings zu Lasten einer erhöhten Latenz. Abbildung 5.4 demonstriert dieses Problem: Ein Rahmen, der erst nach der Phase des geöffneten Gates ankommt, verweilt bis zur erneuten Öffnung des Gates in der folgenden Periode in der Warteschlange der Bridge. So entsteht eine hohe Latenz. Ideal für die eine geringe Latenz wäre es dagegen, wenn der Rahmen so versandt wird, dass er unmittelbar zum Zeitpunkt der Öffnung des Gates weitergeleitet werden kann – der Rahmen kann die TSN-Bridge also nicht zufällig innerhalb der Periodendauer erreichen. Es bedarf einer Synchronisation, so dass der Sender angepasst auf die Gate Control List des Time-Aware Shapers senden kann.

Zu diesem Zwecke wird ein zweiter, per Ethernet verbundener Mikrocontroller eingeführt, der im folgenden als „Trigger“ bezeichnet wird. Der Trigger löst ein periodisches, an die Periode der TSN-Bridge synchronisiertes, Triggersignal aus. Dies basiert auf dem folgenden Ansatz: Der Teensy sendet periodisch – der Sendezeitpunkt liegt beliebig innerhalb der Periodendauer – einen Rahmen an den Trigger (dieser Rahmen wird nun Triggerrahmen genannt). Der Triggerrahmen wird jedoch bis zum Beginn der Periode durch die TSN-Bridge zurückgehalten. Die Gate Control List ist also so konfiguriert, dass nur zu Beginn jeder Periode für einen sehr kurzen Zeitraum (z.B. 15 µs) das Gate der Warteschlange, in der sich der Triggerrahmen befindet, geöffnet ist. So wird erreicht, dass der Trigger den Triggerrahmen genau zu Beginn einer jeden neuen Periode erhält. Sobald der Trigger diesen erhält, wird dies unverzüglich durch Signalwechsel an einem GPIO-Pin angezeigt. Insgesamt schaltet der Trigger also genau zu Beginn jeder Periode einen GPIO-Pin um. Die ESPs sind mit diesem GPIO-Pin verbunden und werden dadurch an die Periode der Bridge synchronisiert.

Diese Art der Synchronisation ermöglicht im Verlauf der Versuchsaufbauten, die Auswirkung des Time-Aware Shapers zu messen.

5.4 Versuchsaufbauten ohne Time-Aware Shaper

In den vorangegangenen Abschnitten wurden die Grundlagen gelegt: Zunächst ist es möglich, Latenzen einer Übertragungsrichtung mit einer Uhr zu messen. Im Anschluss wurde die Synchronisation mehrere Mikrocontroller untereinander sowie mit der Periode einer TSN-Bridge eingeführt. Mithilfe dieser Grundbausteine werden nun konkrete Versuchsaufbauten entworfen, die jeweils geeignet sind, Messdaten zu einer Fragestellung zu generieren.
In diesem Abschnitt werden drei Szenarien ohne Verwendung des Time-Aware Shapers eingeführt.

5.4.1 Szenario 0: Messung der Latenz über Ethernet

Im Verlauf der Arbeit wird grundsätzlich die Charakteristik von Latenz und Jitter der Übertragung über einen drahtlosen Link ermittelt. Um dies in Verhältnis zu Latenz und Jitter einer Übertragung rein über Ethernet zu setzen, wird mithilfe desselben Grundprinzips wie im Basisentwurf eingeführt – der Messung der Latenz einer Übertragungsrichtung mithilfe einer Uhr – ein Szenario untersucht, in dem zwei per Ethernet verbundene Mikrokontroller miteinander kommunizieren.

Abbildung 5.5 zeigt die Topologie des Netzwerks dieses Versuchsaufbaus sowie den Fluss der übertragenen Rahmen.

5.4.2 Szenario 1: Basismessungen der Übertragung über einen WiFi-Link

Zu Beginn werden die Latenzen eines Basisszenarios erfasst. Die Netzwerktopologie ist wie in Abbildung 5.2 dargestellt. Für einen Sender und einen Empfänger werden mit deaktiviertem Time-Aware Shaper sowie bei möglichst freiem Medium – soweit in realer Büroumgebung möglich – die Sende- und Empfangszeitstempel erfasst. So kann die typische Latenz sowie die Key Performance Indikatoren 1-Jitter und Min-Max-Jitter dieses Basisversuchs erfasst und charakterisiert werden. Die Messungen werden für beide Übertragungsrichtungen – von ESP32 zu Teensy, sowie von Teensy zu ESP32 – durchgeführt.

Dazu werden die beiden Szenarien 1S, 1E eingeführt; sie unterscheiden sich nur darin, wer die Rolle von Sender bzw. Empfänger einnimmt. Abbildung 5.6 zeigt Szenario 1S, in dem der ESP die Rolle des Senders einnimmt, und Rahmen an den Teensy überträgt. Abbildung 5.7 zeigt Szenario 1E, in dem die Übertragungsrichtung umgekehrt wurde; der ESP empfängt nun durch den Teensy versendete Rahmen.

Beeinflussende Parameter

Bei gleich bleibender Netzwerktopologie (Betrachtung des Szenarios 1S) lässt sich auch der Einfluss verschiedener Parameter analysieren. Ein Einflussfaktor ist die Konfiguration des WiFi-Treibers des drahtlosen Endpunkts. In der Standardkonfiguration des ESP32-C6 durch Espressif ist das Features *Aggregate MAC Protocol Data Unit (AMPDU)* aktiviert. In einem weiteren Versuch wird AMPDU deaktiviert und die Veränderung von Latenz und Jitter beobachtet.

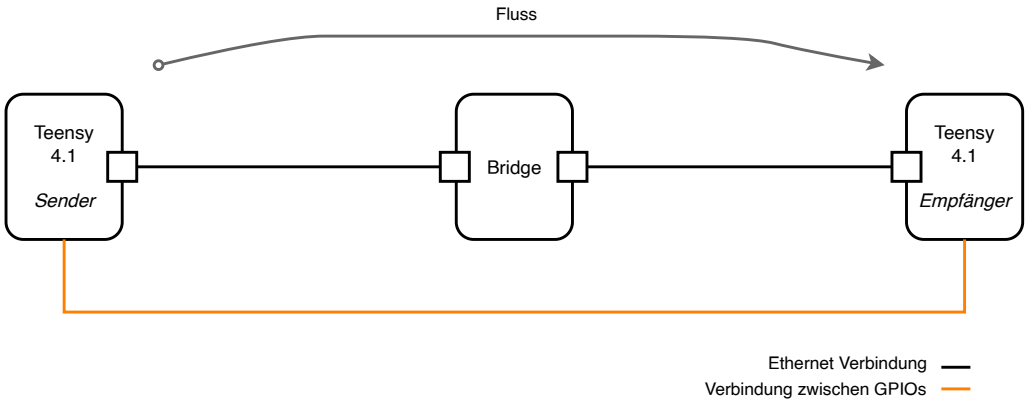


Abbildung 5.5: Szenario 0: Übertragung per Ethernet

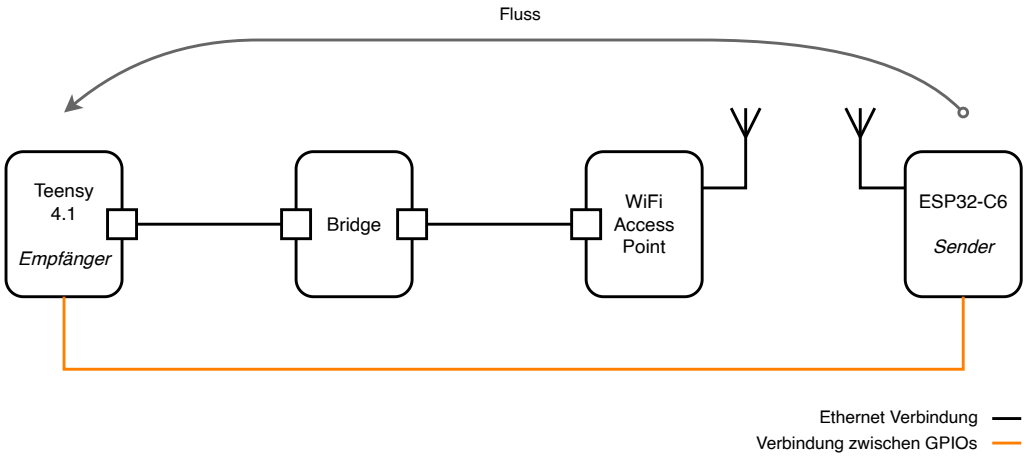


Abbildung 5.6: Szenario 1S: Übertragung per WiFi, ESP32-C6 sendet an Teensy.

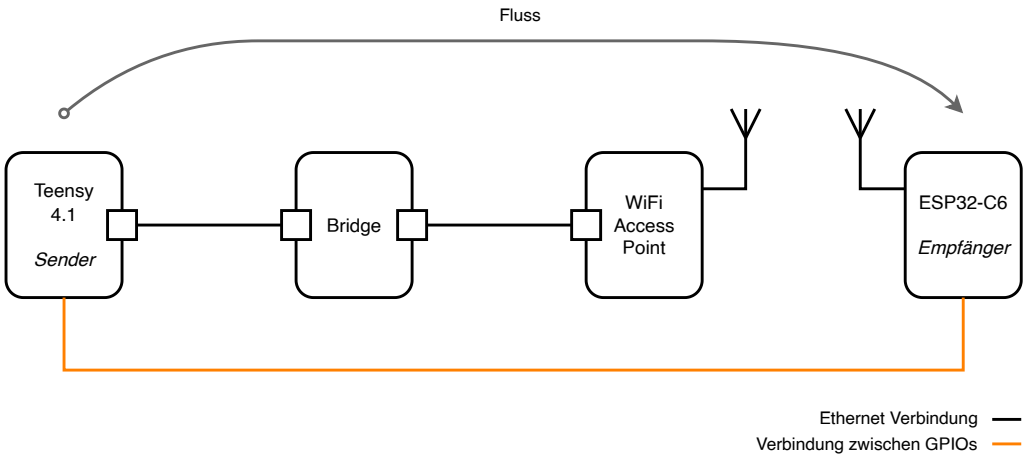


Abbildung 5.7: Szenario 1E: Übertragung per WiFi, ESP32-C6 empfängt vom Teensy.

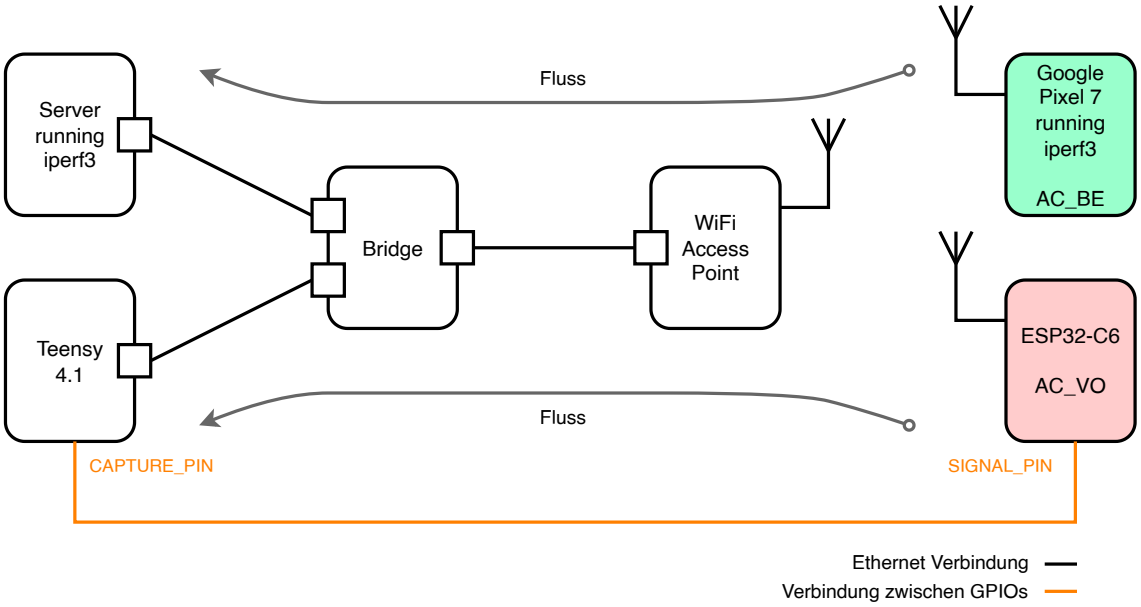


Abbildung 5.8: Szenario 2: Erfassung der Auswirkung von der EDCA-Zugriffskategorien auf die Latenz, in der Gegenwart von Cross-Traffic.

Ein weiterer Einflussfaktor, eine hohe Auslastung des drahtlosen Mediums, wird im Zuge einer Bewertung der Effektivität der Priorisierung der Zugriffskategorien von Enhanced Distributed Channel Access untersucht.

5.4.3 Szenario 2: Verwendung von Enhanced Distributed Channel Access in der Gegenwart von Cross-Traffic

Im technischen Hintergrund in 2.1.3 wurde bereits ausführlich auf das derzeit einzige weit verbreitete Verfahren zur Spezifikation einer Dienstgüte – Enhanced Distributed Channel Access – eingegangen. Während der ältere ESP32 noch kein EDCA unterstützt, bietet der neuere ESP32-C6 von Espressif Unterstützung für WiFi Multimedia (WMM) (Unterunterabschnitt 2.3.2).

Ziel ist es, den Einfluss der vier Zugriffskategorien – Voice, Video, Best Effort und Background – bei einem hoch ausgelasteten Medium, also in der Gegenwart von Cross-Traffic, auf die Latenz zu untersuchen. Um hohe Last auf dem Medium zu erzeugen, kommt mit dem Google Pixel 7 ein handelsübliches Android-Smartphone zum Einsatz, welches Teil des selben Basic Serivce Sets ist und mit iperf3[1] im UDP-Modus einen Geschwindigkeitstest startet. So wird mit einfachen Mitteln eine realistische – da reale – Last auf dem Medium erzeugt.

Abbildung 5.8 zeigt den Versuchsaufbau konzeptionell. Ein ESP32-C6 agiert als periodischer Sender. Innerhalb jedes festgelegten Zeitintervalls überträgt die Anwendung einen Rahmen, der ein UDP-Datagramm enthält, an den Teensy. Die Latenz dieser Übertragung wird mit der bekannten Methode gemessen.

[1] Die Android-Anwendung PingTools enthält iperf3. https://pingtools.org/

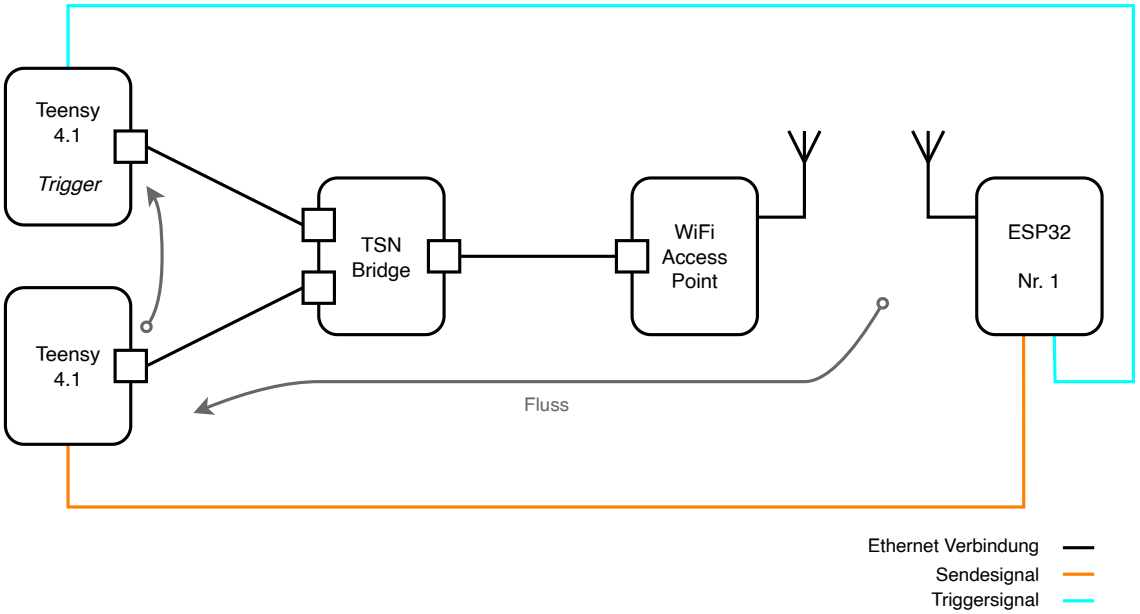


Abbildung 5.9: Versuchsaufbau zur Messung des Effekts des Time-Aware Shapers zur Reduktion der Latenzvarianz

Im Programmcode des ESP32-C6 kann angegeben werden, mit welcher Zugriffskategorie der Versand der Rahmen erfolgen soll. Die Darstellung in Abbildung 5.8 zeigt exemplarisch die Zugriffskategorie AC_VO. Durchgeführt wird die Messung aber für jede der vier Zugriffskategorien.

5.5 Versuchsaufbauten mit Time-Aware Shaper

In diesem Szenario werden zwei Szenarien eingeführt, die sich Funktionalität des Time-Aware Shapers zunutze machen oder emulieren. Im ersten Unterabschnitt wird der Time-Aware Shaper auf der TSN-Bridge eingesetzt, um die variable Paketverzögerung der Übertragung des WiFi-Links zu kompensieren (Packet Delay Compensation). Im zweiten Unterabschnitt werden mehrere sendende Geräte zeitlich zueinander positioniert, sodass die Übertragungslatenz insgesamt verringert werden kann.

5.5.1 Szenario 3: Kompensation der Latenzvarianz des WiFi-Links

Zunächst kommt der Time-Aware Shaper zum Einsatz, um die Latenzvarianz der Übertragung zu minimieren, beziehungsweise eine hohe Zuverlässigkeit in einer kurzen Zeitspanne $T = t_{max} - t_{min}$ zu ermöglichen (siehe Unterabschnitt 5.3.2). Abbildung 5.9 zeigt diesen Versuchsaufbau. Die blaue Verbindung stellt das Triggersignal dar, welches den Beginn einer neuen Periode signalisiert und den Sendevorgang des ESPs auslöst. Der Einfluss des Einsatzes des TAS zur Kompensation des variablen Paketverzögerung auf Jitter und Zuverlässigkeit wird analysiert.

5.5.2 Szenario 4: Reduktion der Latenz durch Scheduling der Übertragungen

In Situationen, in denen mehrere Stationen gleichzeitig senden möchten, kommt es zu einer Verlängerung der Latenz: Nur eine Station sendet als erstes, die übrigen müssen die Übertragung (und ggf. Acknowledgements) anderer Stationen abwarten; möglicherweise kommt es zu Kollisionen. Positioniert man die Übertragungen der sendenden Geräte stattdessen anhand eines geschickt gewählten Zeitplans, lassen sich Situationen vermeiden, in denen mehrere Geräte gleichzeitig eine Übertragung anstoßen und dann aufgrund des Medienzugriffsverfahrens warten müssen. So lässt sich die Ende-zu-Ende Latenz verringern.

Das vorgestellte Synchronisationsverfahren wird genutzt, so dass alle sendenden ESPs gleichzeitig ausgelöst werden. Nun können zwei Konfigurationen verglichen werden:

1. Alle ausgelösten Sender (ESPs) rufen unmittelbar den Socket-Code auf, so dass alle (annähernd) gleichzeitig senden möchten. Die WiFi-Implentierung des ESPs führt dann die DCF aus.
2. Die ausgelösten Sender bekommen jeweils eine Rückhaltezeit zugewiesen, in welcher sie die Ausführung des Anwendungscodes pausieren (bzw. „busy-waiting“). Erst nach Ablauf der konfigurierten Zeit – die sich je Sender unterscheidet – wird der Socket-Code aufgerufen.

Der Aufbau dieses Versuchs ist in Abbildung 5.10 dargestellt.

Im Kapitel „Verwandte Arbeiten“ (Abschnitt 3.2) wurde ein Vorschlag knapp aufgeführt, den Time-Aware Shaper aus dem Standard 802.1Qbv in das Medienzugriffsverfahren EDCA zu integrieren. Das Inter Frame Spacing sowie das Rückzahlen des Random Backoff Timers würden durch Öffnen des Gates begonnen bzw. durch Schließen des Gates pausiert werden [CRV+18], [CCSR22]. Abbildung 5.11 stellt diesen Entwurf dar. Dieser Vorschlag aus der Literatur wird durch den hier vorgestellten Versuchsaufbau angenähert: Der Effekt, hohe Latenzen aufgrund des Medienzugriffsverfahren durch geschickte Wahl eines Zeitplans zu entzerren, wird in beiden Fällen erreicht. Da der WiFi-Stack der ESP32-Plattformen nicht quelloffen ist, ist eine Implementierung der Gates in Integration mit dem Medienzugriffsverfahren – wie eigentlich vorgeschlagen – nicht möglich; stattdessen erfolgt die zeitliche Koordination bereits auf Anwendungsebene.

5.6 Umsetzung des Messsystems

Um die beschriebenen Versuche durchführen zu können, müssen eine Vielzahl unterschiedlicher Komponenten implementiert werden und ineinander greifen.

Die komplexeste Codebasis ist für den in 5.5.2 geschilderten Versuch (Abbildung 5.10) erforderlich. Daher folgt an dieser Stelle ein knappe Auflistung der zum Einsatz kommenden Komponenten, die implementiert werden müssen:
Der in der Abbildung untere Teensy ist zuständig für die Erfassung der Sendezeitstempel mittels Input Capture sowie der Empfangszeitstempel, die durch die Hardware in den Empfangspuffer geschrieben werden. Außerdem versendet er periodische „Trigger“-Rahmen an den in der Abbildung oberen Teensy. Dieser auslösende Teensy – der in der Abbildung obere, zur Differenzierung der

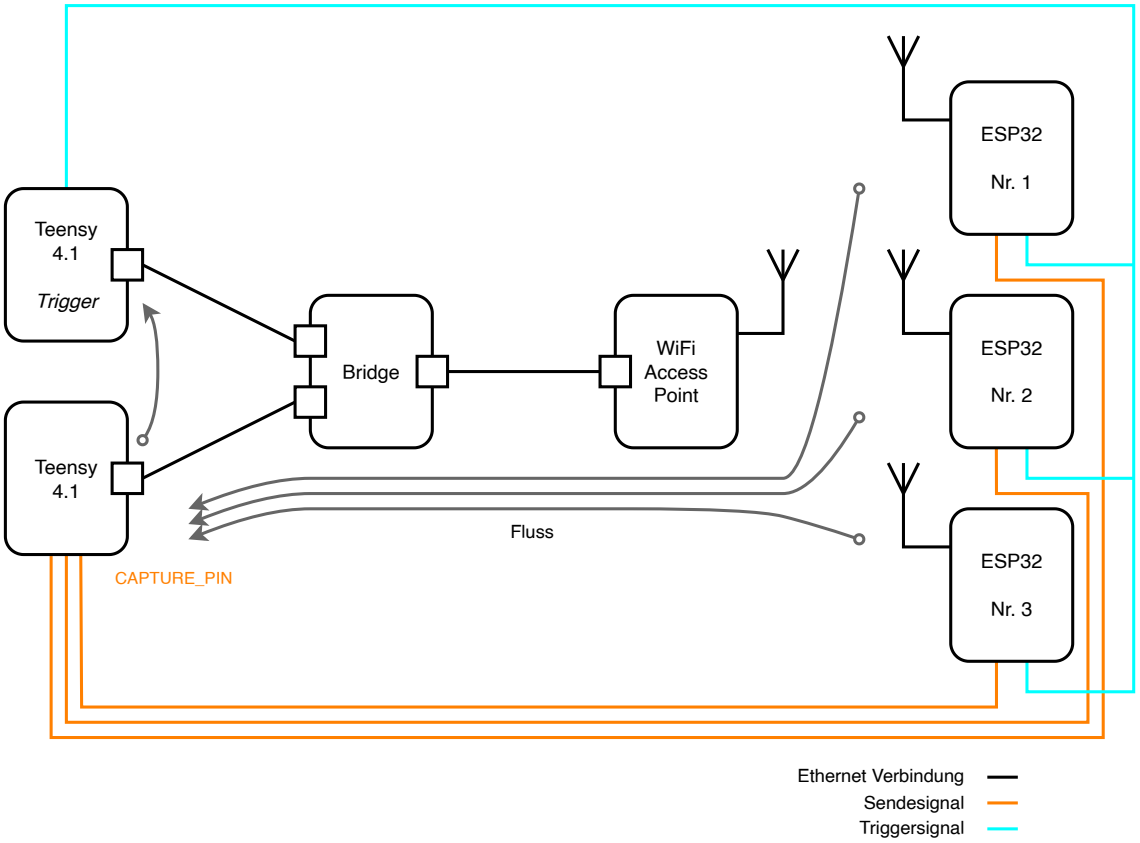


Abbildung 5.10: Veranschaulichung des Versuchsaufbaus

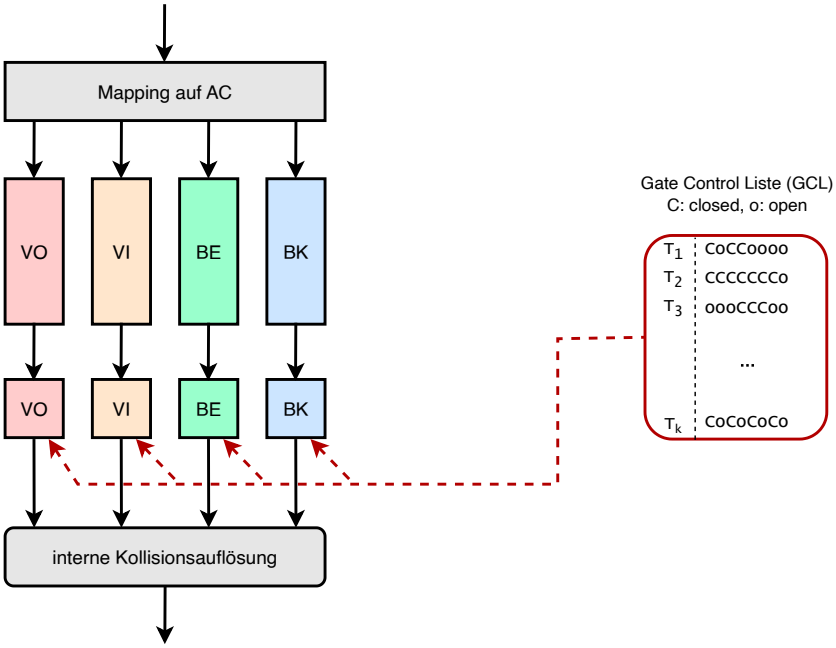


Abbildung 5.11: Vorschlag, das bisherige EDCA-Verfahren um einen TAS zu erweitern

beiden im folgenden Abschnitt als Trigger bezeichnet – hat die Aufgabe, nach Erhalt des Rahmens durch die TSN-Bridge das Triggersignal auszulösen. Die ESP32s sind mit dem Triggersignal verbunden, und lösen nach Erhalt des Signals den Versand des Datagramms aus.

Nun wird festgehalten, dass durch Wiederverwendung dieser Bestandteile auch weitere Versuchsaufbauten realisiert werden können, bei denen ein oder mehrere ESPs als Sender auftreten und Rahmen an den Teensy senden. So muss kein mehrfacher Implementierungsaufwand betrieben werden.

Beispielhaft an der Basismessung erläutert: Gemessen werden dort Latenzen bei deaktiviertem Time-Aware Shaper und möglichst freiem Medium. Die Umsetzung dieses Szenarios wird nun so gestaltet, dass der Sendevorgang ebenfalls durch den Trigger ausgelöst wird. Es spielt für den Versuch schließlich keine Rolle, *wodurch* dieser Sendevorgang intern ausgelöst wurde: Ob innerhalb einer simplen `for`-Schleife, oder durch ein externes Triggersignal. Letzteres ermöglicht aber, beinahe die selbe Codebasis für alle Versuchsaufbauten zu verwenden, statt unterschliedliche Programme für jede der Versuchsaufbauten zu implementieren. Auf welche Art und Weise der ESP implementiert ist, bevor schließlich der GPIO-Pin umgeschaltet wird (was den Sendezeitpunkt markiert) und der Aufruf der Socket-API erfolgt, verändert die Resultate der Messungen nicht. Auch für die Umsetzung des Szenarios EDCA müssen nur wenige Zeilen Code hinzugefügt werden, um den Versand der Rahmen in der jeweiligen Zugriffskategorie zu ermöglichen. Einzige Ausnahme ist die Basismessung, bei der der ESP32 als Empfänger, nicht wie in den übrigen Szenarien als Sender auftritt. Diese erfordern einen höheren Grad der Anpassung.

5.6.1 Überblick

Die Umsetzung der Funktionalität der ESPs gestaltet sich bedingt durch den Entwurf am einfachsten, da sie sich sehr nah an tatsächlichen Anwendungen befindet. Etwa für Basisfunktionalität wie die Herstellung der Verbindung zum Access Point kann auf bestehenden Code zurückgegriffen werden, so dass nur die darauf aufbauende Funktionalität implementiert wird.

Im Gegensatz zur Implementierung des ESP32 – die auf dem IoT-Development-Framework sowie dem dort verwendeten Netzwerkstack lwIP[2] aufbaut – wird für die Implementierung des Teensy-Codes auf keinem bestehenden Framework aufgebaut, sondern Versand und Empfang von Ethernet-Rahmen auf niedrigster Ebene selbst implementiert, bzw. auf bestehendem Source-Code[3] aufgebaut.

Damit das Projekt später leicht wieder verwendet werden kann, soll die Codebasis mit Arduino kompatibel sein, so dass man ohne größeren Anpassungsbedarf den veröffentlichten Code mittels Arduino-Toolchain kompilieren und auf den Teensy übertragen kann. Es gibt mit NativeEthernet[4] und QNEthernet[5] bereits zwei Arduino-kompatible Netzwerkstacks für den Teensy, QNEthernet unterstützt sogar die Erfassung von Sende- und Empfangszeitstempeln. Die Verwendung einer bereits bestehenden Netzwerkbibliothek brächte neben einer erhöhten Einarbeitungszeit einige Nachteile mit sich, so müssten etwa weitere Features (wie die Erfassung der verschiedenen externen Ereignisse) in die Codebasis integriert werden. Zur Anfertigung der Messungen genügt es, direkt

[2] Dokumentation zu lwIP: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/lwip.html
[3] Quellcode, auf dem z.T. aufgebaut wurde: https://github.com/PaulStoffregen/teensy41_ethernet
[4] NativeEthernet auf GitHub: https://github.com/vjmuzik/NativeEthernet
[5] QNEthernet auf GitHub: https://github.com/ssilverman/QNEthernet

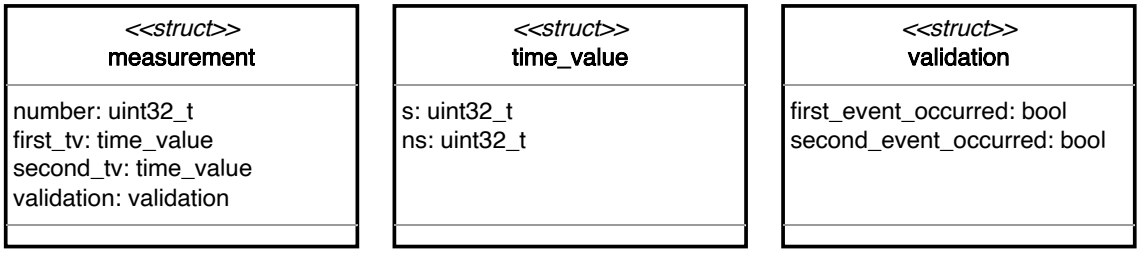


Abbildung 5.12: Datenstrukturen zum Speichern der Messwerte

mit Sende- und Empfangsringspeicher zu interagieren, statt zwischengeschaltete Schichten eines Netzwerkstacks zu verwenden. So bleibt die Codebasis so klein (und damit übersichtlich) wie möglich, außerdem können Hintergrundprozesse – etwa die Beantwortung von ARP- oder mDNS-Anfragen – die Messungen nicht verfälschen. Eine möglichst hohe Kontrolle über jeden einzelnen versendeten Rahmen wird als wichtig angesehen, sodass für die Implementierung des Teensy-Codes Low-Level-Code mit einzeln angefertigten Rahmen und direkter Interaktion mit den Registern der Hardware gewählt wird. Insgesamt erfolgt die Implementierung des Teensys also in einem anderen Stil als die des ESP32, auf ihr liegt im weiteren Verlaufs dieses Abschnitts auch der Fokus.

5.6.2 Erfassen von Messungen

Die Erfassung der Messungen geschieht in einem mehrphasigen Aufbau, bestehend aus Messphase und Übertragungsphase. Der Teensy ist zwar für einen Mikrocontroller mit viel RAM ausgestattet ist, trotzdem können für große Messreihen nicht alle Werte lokal gespeichert werden. Daher werden in einer Messphase jeweils eine vordefinierte Anzahl an Messungen (meist 1000) angefertigt. Diese werden in der Übertragungsphase über die serielle Schnittstelle an einen Computer / Server übertragen, bevor die nächste Messphase beginnt.

Zentrales Element der Erfassung der Messwerte ist die Datenstruktur zur Zwischenspeicherung auf dem Teensy. Abbildung 5.12 zeigt die drei wichtigsten Datenstrukturen.

Eine Messung für einen Rahmen enthält dessen Sende- und Empfangszeitpunkt, die Information, um welchen Rahmen es sich handelt (dazu wird als UDP-Nutzlast eine eindeutige Sequenznummer versendet) und eine weitere Datenstruktur zur Validierung. Es wird dabei in „erstes“ und „zweites“ Ereignis abstrahiert, bezogen auf die zeitliche Ordnung.

Da es möglich ist, dass ein durch den ESP übertragener Rahmen beispielsweise gar nicht ankommt, gibt es für jede Messung das Validierungsfeld, welches lediglich aus zwei boolschen Werten besteht. Vor Beginn jeder Messphase werden alle Validierungsfelder auf `false` zurückgesetzt. So können ungültige Messungen – etwa, wenn ein Rahmen nie ankam, die Speicherstelle aber noch den Empfangszeitstempel aus einer vergangenen Messphase enthält – erkannt werden.

Ein Zeitpunkt (`time_value`) besteht aus einem Wert für die Sekunde und einem Wert für die Nanosekunde.

In jeder Messphase füllt sich so ein Array aus Messungen, beispielhaft erklärt für den Teensy als Empfänger: Wechselt das Signal am Input-Capture fähigen Pin, wird dieser Zeitpunkt durch die Hardware in ein Register gespeichert. Der folgende Interrupt speichert diesen `time_value`

in `first_tv` und markiert durch `validation.first_event_occurred=true`, dass ein Versandzeitpunkt angezeigt wurde. Wird der versendete Rahmen empfangen, wird der Empfangszeitpunkt in `second_tv` gespeichert und die „Flagge“ `validation.second_event_occured` auf `true` gesetzt.

Nach Ende der Messphase werden diese Daten an einen PC zur Auswertung übertragen. Ein Python-Skript liest die über die serielle Schnittstelle übertragenen Informationen, und erstellt daraus Objekte einer Klasse `Measurement`. Die Daten können dann in Python ausgewertet oder in Form einer .csv-Datei gespeichert werden.

Bisher ausgelassen wurde, dass der Teensy in der Lage ist, die Messungen als Empfänger für drei Sender parallel an drei verschiedenen Input-Capture fähigen Pins durchzuführen. Die Messungen werden also eigentlich in einem Array gespeichert, das für jeden Index drei Messungen (`.chan0`, `.chan1`, `.chan2`) enthält.

5.6.3 Timekeeping

Für die Erfassung von Sendezeitstempel, Empfangszeitstempel sowie die Aktualisierung der globalen Variable „second“ sind drei verschiedene Bits im Interrupt Event Register (ENET_EIR) von Relevanz [Sem21, S. 2150]. Das exakte Timing der unterschiedlichen Interrupts zueinander gestaltet sich schwierig, auf einige Details wird in der Implementierung in Unterabschnitt 6.3.3 eingegangen.

5.6.4 Mitschnitt und Analyse der WiFi-Pakete

Im Rahmen der Evaluation wird untersucht, wie oft z.B. Retransmits auftreten – diese Information wird erhalten, indem der Netzwerkverkehr mitgeschnitten wird. Auch für die Entwicklung ist es praktisch, Einblick in die Übertragung der WiFi-Pakete zu erhalten, so können eventuelle Fehlerquellen besser eingegrenzt werden.

Die Kommunikation per WiFi erfolgt „Over the Air“ – damit also über ein Medium, auf das auch Dritte Zugriff haben. Manche Netzwerkkarten unterstützen den sogenannten Monitor Mode: In diesem können alle Funkpakete empfangen werden, die in ihrem Funkbereich übertragen werden – also auch Pakete, die nicht an das mitschneidende Gerät adressiert sind. Dies wird verwendet, um mithilfe eines Laptops die Kommunikation zwischen Access Point und ESP32 aufzuzeichnen. Ist das WiFi-Netzwerk verschlüsselt und der Schlüssel unbekannt, sind nur Informationen bis ISO-OSI Ebene 2 ersichtlich, etwa MAC-Adressen von Sender und Empfänger oder die Information, ob es sich um eine erneute Übertragung handelt. Ist der Schlüssel bekannt, bestehen Möglichkeiten, diesen in die Analysewerkzeuge zu importieren und somit auch Informationen über die übertragenen Daten bis hin zur Anwendungsebene zu erhalten. Zur Vereinfachung kann die Verschlüsselung schlicht deaktiviert werden, dann sind alle übertragenen Daten zugänglich, und auch die IP-Adressen oder die übertragene Nutzlast ist einsehbar.

Dabei ist zu beachten, dass die Daten, die ein drittes Gerät empfängt, nicht genau die selben sein müssen, die der eigentliche Empfänger erhält. So kann es vorkommen, dass der eigentliche Empfänger – z.B. der ESP32 – den übertragenen Rahmen durch lokale Interferenz mit Bitfehlern und somit falscher Prüfsumme empfängt; der Rahmen wird daher verworfen. Das mitschneidende Gerät erhält den Rahmen jedoch korrekt. Die Paketaufzeichnung liefert wertvolle Einblicke in

die Vorgänge auf dem Übertragungsmedium – es ist aber festzuhalten, dass die Aufzeichnung des Netzwerkverkehrs auf einem dritten Gerät nicht genau den Datenerhalt auf dem eigentlichen Empfänger widerspiegelt.

6 Implementierung

Im folgenden Kapitel wird die Implementierung des Messsystems vorgestellt. Aufgrund des hohen Umfangs der Codebasis kann nicht auf alle Aspekte eingegangen werden, stattdessen werden einzelne, jeweils thematisch zusammengehörige Aspekte der Implementierung vorgestellt.

Im ersten Abschnitt dieses Kapitels wird die Implementierung der ESP32-Mikrocontroller vorgestellt. Dabei werden einzelne Ausschnitte des erstellten Codes gezeigt. Der zweite Abschnitt des Kapitels erläutert die Implementierung des Codes des Teensys. Ein Fokus liegt dabei auf einer aufgetretetenen Schwierigkeit, dem Timing verschiedener Interrupts zueinander. Zuletzt wird vorgestellt, wie Paketaufzeichnungen angefertigt wurden.

6.1 ESP32

Die ESP32-Geräte werden in zwei Rollen verwendet: Einerseits als sendendes Gerät, andererseits als empfangendes Gerät. Daher werden für diesen Mikrocontroller zwei Programme entwickelt, eines für die Rolle als Sender (esp_sender_x), und eines für die Rolle als Empfänger (esp_receiver_x).

6.1.1 WiFi-Verbindung

Aufbau der Verbindung zum Access Point

Beide Programme haben gemeinsam, dass zunächst eine Verbindung zum Access Point aufgebaut werden muss. Espressif empfiehlt, Anwendungen, die auf ESP-IDF basieren, auf deren Beispielprogrammen aufzubauen. Daher basiert der Code zum Aufbau der Verbindung mit dem Access Point auf dem *getting_started*-Programm[1] für Stationen. In diesem bezieht der ESP seine IP-Adresse per Dynamic Host Configuration Protocol (DHCP). Im verwendeten Aufbau dagegen ist jedoch kein DHCP-Server aktiv, sodass eine statische Internet Protocol (IP)-Adresse verwendet werden muss. Dazu wird mithilfe der in Espresifs IoT Development Framework enthaltenen Funktion `esp_netif_dhcpc_stop` der DHCP-Client gestoppt und mit `esp_netif_set_ip_info` eine neue, geeignete Konfiguration gesetzt.

[1] Espressifs Beispielprogramm für WiFi-Stationen: https://github.com/espressif/esp-idf/tree/master/examples/wifi/getting_started/station

Konfiguration der Energiespareinstellungen für WiFi

Ein wichtiger Faktor für niedrige Latenz besonders beim Empfang der WiFi-Pakete sind die Energieeinstellungen des Mikrocontrollers; in Foren beklagen Nutzer sich häufiger über sehr hohe mittlere Latenzen (in der Region von 50 ms). Dies wird ausgelöst durch die Default-Einstellung der WiFi-Energiekonfiguration, deren Konsequenzen aus der Dokumentation zunächst nicht klar hervorgehen. [Esp23a] schildert drei mögliche Energiespareinstellungen:

1. WIFI_PS_NONE. In diesem Modus erfolgt keine Energieeinsparung, das Modem ist ständig aktiv.
2. WIFI_PS_MIN_MODEM. In diesem Modus geht der ESP in einen Energiesparmodus und deaktiviert das Modem zeitweise. Mindestens zu jedem Beacon-Intervall wird das Medium abgehört, um das Beacon zu empfangen. Sofern der Access Point Rahmen für den ESP gepuffert hat, können diese dann empfangen werden. Dies ist auch die Standardeinstellung.
3. WIFI_PS_MAX_MODEM. Ähnlich zum vorherigen Modus wird das Modem deaktiviert, es wird aber nicht jedes Beacon empfangen, sondern nur jedes k-te (k konfigurierbar).

Belässt man es nun bei der Standardeinstellung, ist die Folge eine hohe Latenz, da der Mikrocontroller nur selten überhaupt sein Radio aktiviert. Daher ist es wichtig, den Energiesparmodus WIFI_PS_NONE zu wählen. Mithilfe der nachfolgenden Methode in Listing 6.1 ist es nach Aufruf von `esp_wifi_init()` möglich, die Energiespareinstellung zu setzen.

```
esp_wifi_set_ps(WIFI_PS_NONE);
```

Listing 6.1: ESP32 – Deaktivieren des Stromsparmodus des WiFi-Moduls

6.1.2 Implementierungsdetails des Programms esp_sender_x

Die grundlegende Funktionalität des Programmes *esp_sender_x* ist wie folgt: Die steigende Taktflanke eines „Trigger“-Signals signalisiert den Beginn einer neuen Hyperperiode. Innerhalb dieser wird – dies ist konfigurierbar – entweder sofort, oder nach einer vordefinierten Verzögerung eine Information über WiFi versandt. Direkt vor Versand des Pakets wird ein GPIO-Pin umgeschaltet, um den Versandzeitpunkt des Pakets anzuzeigen. Dann wird das Paket per `sendto`-Funktion über einen Socket versendet.

```
void IRAM_ATTR gpio_isr_handler(void *arg)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR(xSemaphore, &xHigherPriorityTaskWoken);
    // Request a context switch if needed
    if (xHigherPriorityTaskWoken == pdTRUE)
    {
        portYIELD_FROM_ISR();
    }
}
```

Listing 6.2: ESP32 – Interrupt Service Routine nach Triggersignal

```
uint32_t packet_counter = 0;
char payload[128];
bool signal_pin_high = false;

for (;;)
{
    sprintf(payload, "ESP-%02d#%06lu\n", CONFIG_DEVICE_NUMBER, packet_counter++);
    size_t payload_len = strlen(payload);

    if (xSemaphoreTake(xSemaphore, portMAX_DELAY) == pdTRUE)
    {
#if CONFIG_TSN == 1
        // Blocking delay, vTaskDelay() is not accurate enough.
        ets_delay_us(CONFIG_DELAY_US);
#endif

        signal_pin_high = !signal_pin_high;
        gpio_set_level(SIGNAL_PIN, signal_pin_high);
        sendto(sock, payload, payload_len, 0, (struct sockaddr *)&dest_addr, sizeof(dest_addr)
);
    }
}
```

Listing 6.3: ESP32 - Ausschnitt des Tasks, der UDP Datagramme versendet.

Dazu werden zwei verschiedene Tasks und ein Interrupt genutzt. Einstiegspunkt in die Ausführung des Programmes *esp_sender_x* ist der Task `app_main`; dort erfolgt beispielsweise die Konfiguration des WiFi-Interfaces und der beschriebenen Stromspareinstellung. Außerdem wird ein GPIO-Pin so konfiguriert, dass bei steigendem Signalpegel die Interrupt Service Routine (ISR) in Listing 6.2 ausgelöst wird.

Die Semaphore, die in Listing 6.2 gegeben wird, wird im Task `udp_sender_task` genommen. Dieser Task erstellt einen Socket, der Datagramme an eine konfigurierte IP-Adresse mit konfiguriertem Port versenden kann. Nach der Erstkonfiguration befindet sich der Task in einer Endlosschleife, in der nach Geben der Semaphore Daten gesendet werden. Listing 6.3 zeigt einen Ausschnitt dieses Tasks. Noch bevor auf die Semaphore gewartet wird, wird der zu sendende Text vorbereitet. Nach erfolgtem Trigger-Signal – also gegebener Semaphore – erfolgt die ggf. konfigurierte Wartezeit. Dann wird der Signal-Pin, der den Versand des Pakets anzeigt, umgeschaltet und `sendto` aufgerufen.

6.1.3 Implementierungsdetails des Programms esp_receiver_x

Kommt der ESP32 als Empfänger zum Einsatz, ist der Ablauf des Codes noch einfacher. Nach Aufbau der Verbindung zum Access Point und setzen der statischen IP-Adresse befindet sich Task in einer Endlosschleife. Der Aufruf von `recvfrom` blockiert, bis auf dem angegebenen Port Daten erhalten werden; der GPIO-Pin wird unmittelbar danach umgeschaltet. Ein Ausschnitt dieses Tasks ist in Listing 6.4 zu sehen.

```
int len = recvfrom(sockfd, buf, MAX_UDP_PACKET_SIZE, 0, (struct sockaddr *)&src_addr, &socklen
);
if (len < 0)
{
    ESP_LOGE(TAG, "recvfrom failed: %s", strerror(errno));
}
else
{
    gpio_set_level(MEASURE_PIN, high);
    high = !high;
}
```

Listing 6.4: ESP32 - Ausschnitt des Tasks, der UDP Datagramme empfängt.

6.1.4 Entwicklung für ESP32 und ESP32C6

Die Implementierung wird so gestaltet, dass sie sowohl mit dem älteren ESP32 mit WiFi 4 als auch dem neueren ESP32C6 mit WiFi 6 kompatibel ist. Bei der Konfiguration der beiden Mikrocontroller entstehen leichte Unterschiede, beispielsweise kann für den ESP32C6 – der nur einen CPU-Kern hat – im Unterschied zum ESP32 keine Affinität zu CPU-Kernen für den Netzwerkstack spezifiziert werden. Die Codebasis lässt sich abseits der Unterschiede in der Konfiguration ohne weitere Veränderungen für beide Mikrocontroller kompilieren und verwenden.

```
1    $ cd working_dir
2    $ idf.py set-target esp32c6
3    $ idf.py menuconfig
4    $ idf.py build flash monitor
```

Listing 6.5: Übertragung eines Projekts auf ESP32C6

Sofern die zur Kompilierung erforderlichen Programme[2] installiert sind, kann ein Projekt wie in Listing 6.5 auf den Mikrocontroller übertragen werden. Zunächst setzt man das Ziel (esp32 oder esp32c6) und erstellt die Konfigurationsdatei. Dies öffnet einen interaktiven Dialog, in dem die Konfigurationsparameter eingetragen werden können. Der letzte Befehl kompiliert den Code, überträgt in auf den Mikrocontroller und öffnet die serielle Konsole.

[2] Installationsanleitung für die Toolchain von ESP-IDF: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html

6.2 Teensy

6.2.1 Programmierung des i.MX RT 1062

Für einige verwendete Funktionalität, etwa die Konfiguration des I/O-Multiplexers des i.MX RT 1062 für Hardware-Timestamping verschiedener externer Ereignisse und der entsprechenden Kontrollregister konnte kein vorhandener Code oder keine Softwarebibliothek gefunden werden, die bereits das gewünschte leistet. Daher wurde diese Funktionalität mithilfe des Handbuches des Prozessors [Sem21] selbst implementiert, indem in verschiedenen Registern entsprechende Bits gesetzt werden. So geschriebener Code wirkt schnell unübersichtlich und kann später durch Dritte kaum nachvollzogen werden. Da der Programmcode als Bestandteils des Messsystems eine der Beiträge dieser Arbeit ist, soll der Code jedoch lesbar gehalten werden.

Beispiel

Im Verlauf der Implementierung werden etwa zur Aktivierung bestimmter Funktionalität gewisse Bits in Registern des i.MX RT 1062 gesetzt.
Möchte man die PTP-Timestamping-Uhr des Chips aktivieren, müssen im entsprechenden Kontrollregister (*Adjustable Timer Control Register*, ENET_ATCR) die folgenden drei Bits gesetzt werden:

- Bit 5 muss laut Handbuch immer mit eins beschrieben werden [Sem21, S. 2240]
- Bit 4, um periodischen Interrupt zu generieren (PEREN)
- Bit 0, um den Timer zu aktivieren. (EN)

Dabei sind die folgenden drei Implementierungen äquivalent:

```
1    ENET_ATCR = 49;
```

```
1    ENET_ATCR = (1 << 5) | (1 << 4) | (1 << 0);
```

```
1    /* Adjustable Timer Control Register Reserved: Has to be one
2    pg 2240 */
3    #define ENET_ATCR_RSVD (1 << 5)
4    /* Adjustable Timer Control Register Peren: Enable Periodical Event
5    pg 2240*/
6    #define ENET_ATCR_PEREN (1 << 4)
7    /* Adjustable Timer Control Register Enable: Enable Timer
8    pg 2240*/
9    #define ENET_ATCR_ENABLE (1 << 0)
10   ENET_ATCR = ENET_ATCR_RSVD | ENET_ATCR_PEREN | ENET_ATCR_ENABLE;
```

Die zuerst genannte Variante ist später nur schwer verständlich, die zweite Variante ist schon etwas lesbarer. In der Implementierung wird aufgrund der noch besseren Wartbarkeit die dritte Variante gewählt; die `#define` Anweisungen sind dabei in eine separate Datei ausgelagert. In modernen Entwicklungsumgebungen ist somit die Erläuterung des Wertes bei darüberfahren der Maus sichtbar. Weiterhin werden im Code `#define`-Anweisungen verwendet, um beispielsweise „ALT5“ statt „0b101“ verwenden zu können, so dass auch die Auswahl von Optionen bei Registern konsistent mit der Schreibweise im technischen Handbuch [Sem21] ist.

6.2.2 Konfiguration und Interrupt bei Input Capture

Der i.MX RT1062 besitzt jeweils vier TCSR- bzw. TCCR-Register [Sem21, S. 2118, S. 2114, S. 2150, S. 2248, S. 2246], so dass zunächst der Anschein erweckt wird, man könne die Zeitstempel vierer Signale an verschiedenen Pins erfassen - es sind jedoch nur drei der internen Pins des i.MX RT 1062 bei der Teensy-Entwicklerplatine auch herausgeführt. Es handelt sich dabei um den internen Pin GPIO_B1_12 auf Pin 35 des Teensy 4.1-Boards, GPIO_AD_B0_13 auf Pin 25 sowie GPIO_AD_B1_03 auf Pin 15.

Ein einzelner physischer Pin auf dem Chip kann mehrere Funktionalitäten inne haben, und die selbe Funktionalität kann gegebenenfalls durch verschiedene physische Pins ausgeübt werden. In diesem Fall erfolgt die Konfiguration des physischen Pins durch Konfigurationskommandos des Multiplexers und des *Daisy Chain Select* [Sem21, S. 322]. Das Setup des Pin-Multiplexings, die

```
1  // Reset Timer Capture Status Registers to enable configuration.
2  // After boot, the Timer Capture Status Registers are already 0, this is just to be
   sure.
3  ENET_TCSR0 = RESET;
4  ENET_TCSR1 = RESET;
5  ENET_TCSR2 = RESET;
6
7  // Configure Pin Multiplexing.
8  IOMUXC_SW_MUX_CTL_PAD_GPIO_B1_12    = ALT3; // Pin 35, ENET_1588_EVENT0_IN
9  IOMUXC_SW_MUX_CTL_PAD_GPIO_AD_B0_13 = ALT6; // Pin 25, ENET_1588_EVENT1_IN
10 IOMUXC_SW_MUX_CTL_PAD_GPIO_AD_B1_03 = ALT4; // Pin 15, ENET_1588_EVENT2_IN
11
12 // Configure Daisy Chain. (pg 322 rev 3)
13 IOMUXC_ENET0_TIMER_SELECT_INPUT = ALT2; // Pin 35, ENET_1588_EVENT0_IN
14
15 // Configure Timer Capture Registers.
16 // Set registers to Input Capture on both edges (IC_BE), enable interrupts (TIE) and
   clear the Timer Flag (TF).
17 ENET_TCSR0 = ENET_TCSR_TF | ENET_TCSR_TIE | ENET_TCSR_TMODE_IC_BE;
18 ENET_TCSR1 = ENET_TCSR_TF | ENET_TCSR_TIE | ENET_TCSR_TMODE_IC_BE;
19 ENET_TCSR2 = ENET_TCSR_TF | ENET_TCSR_TIE | ENET_TCSR_TMODE_IC_BE;
```

Listing 6.6: Code zur Konfiguration von Pin-Multiplexiung und Kontrollregister des Input-Capture

Konfiguration der Daisy Chain sowie die Initialisierung der *Timer Capture Status Register* geschieht in der in der Datei *teensy41_ethernet.cpp* definierten Methode *void ptpclk_init*. Listing 6.6 zeigt einen Ausschnitt dieser Methode.

Nach diesem Konfigurationsschritt wird bei Wechsel des Pegels des Eingangssignals der Zeitpunkt des Wechsels erfasst - jedenfalls der Wert der Nanosekunde seit letztem Zurücksetzen des Timers. Somit ist zusätzlich die Information notwendig, in welcher Sekunde das Ereignis auftrat. Wie in Listing 6.6 zu sehen ist, werden die Bits im Kontrollregister so gesetzt, dass mit Wechsel des Signalpegels auch ein Interrupt ausgelöst wird. In der deshalb ausgeführten Unterbrechungsdienstroutine wird die Sekunde des *Input Capture*-Ereignisses in die Datenstruktur der Messungen gespeichert.

Damit der Interrupt nicht direkt erneut auslöst, muss die *Timer Flag* für das jeweilige Kontrollregister zurückgesetzt werden. Eine Besonderheit ist dabei die Art und Weise, wie Bits zurückgesetzt werden: Ist das „Timer Flag“-Bit 1, kann es zu 0 zurückgesetzt werden, indem man eine 1 hineinschreibt [Sem21, S. 2247].

Dabei ist zu berücksichtigen, dass die Bits „Timer Interrupt Enable“ sowie „Timer Mode“ gesetzt bleiben müssen, um weitere Input Captures zu ermöglichen. Wie in Listing 6.7 zu sehen, werden also bei jedem Interrupt Bit 7 zum zurücksetzen der Timer Flag (ENET_TCSR_TF), Bit 6, um weiterhin Interrupts zuzulassen (ENET_TCSR_TIE) sowie Bits 2, 3 (ENET_TCSR_TMODE_IC_BE) um auf beiden Flanken einen Input Capture auszulösen geschrieben.

```
1  void enet_timer_isr()
2  {
3      if (ENET_TCSR0 & ENET_TCSR_TF)
4      {
5          // Clear the timer flag; don't unset other settings.
6          ENET_TCSR0 = ENET_TCSR_TF | ENET_TCSR_TIE | ENET_TCSR_TMODE_IC_BE;
7
8          // Store the current time, increase the index.
9          volatile uint32_t index = interrupt_counts.chan0++;
10         rcv_measurements[index].chan0.first_tv.s = seconds;
11         rcv_measurements[index].chan0.first_tv.ns = ENET_TCCR0;
12         rcv_measurements[index].chan0.validation.first_event_occurred = true;
13     }
14
15     /* Procedure is analog for TCSR1 with chan1 and TCSR2 with chan2. */
16     /* ... */
17
18     asm("dsb");
19 }
```

Listing 6.7: Teensy – Interrupt Service Routine für Input Capture

6.3 Implementierung des Timekeepings auf dem Teensy

Die Timings der verschiedenen Interrupts zueinander können eine Schwierigkeit darstellen. Im folgenden Abschnitt wird zunächst auf die Implementierungsdetails des Timekeepings eingegangen und dann die Problematik geschildert.

6.3.1 Setup der Timestamping-Clock

Essentiell für das Timestamping ist der Free-Running Counter (FRC), welcher die Zeitstempel generiert. Er bietet auch Möglichkeiten zur Korrektur der Zeit, die etwa für eine PTP-Implementierung essentiell wären, hier aber nicht weiter benötigt werden. Das Handbuch des Mikroprozessors beschreibt, wie der FRC typischerweise verwendet wird: Der Hardware-Zähler deckt das Intervall einer Sekunde ab (also Werte zwischen 0 und 10^9, denn $10^9\,\text{ns} = 1\,\text{s}$ – gezählt werden stets Nanosekunden). Erreicht der Zähler den Wert 10^9, wird ein periodisches Ereignis ausgelöst – ein Interrupt. Die Software verwaltet dann einen Sekundenzähler [Sem21, S. 2115 ff.].

Dieser Vorschlag wurde auch in der vorgestellten Implementierung so umgesetzt. Die verwendete Uhr hat eine Frequenz von $f = 25\,\text{MHz}$, also eine Periodendauer von $T = \frac{1}{f} = 40\,\text{ns}$. Die Konfiguration erfolgt mithilfe der folgenden Register:

- Time-Stamping Clock Period Register (ENET_ATINC): Enthält den Wert, der jeden Taktzyklus auf den FRC addiert wird, also 40 (alle Werte geben jeweils Nanosekunden an).
- Timer Period Register (ENET_ATPER): Erreicht der FRC diesen Wert, wird der Interrupt ausgelöst und der Timer beginnt wieder bei 0. Dieser Wert wird auf 10^9 gesetzt.
- Adjustable Timer Control Register (ENET_ATCR): Erlaubt die Konfiguration des FRC wie in Unterunterabschnitt 6.2.1 gezeigt. In diesem Konfigurationsschritt werden die periodischen Interrupts aktiviert.

Gemessen seit Programmstart enthält der FRC nun also immer die aktuelle Nanosekunde, die Anzahl der Sekunden seit Programmstart wird durch die periodischen Interrupts aktualisiert. Beide Informationen zusammen werden dann zum Erfassen von Sende- und Empfangszeitstempeln genutzt.

6.3.2 Zeitstempel

Die Hardware-gestützte, präzise Erfassung der Zeitstempel eingehender und ausgehender Rahmen ist eines der wichtigsten Features des Mikroprozessors, welches ihn zu einem geeigneten Messinstrument macht.

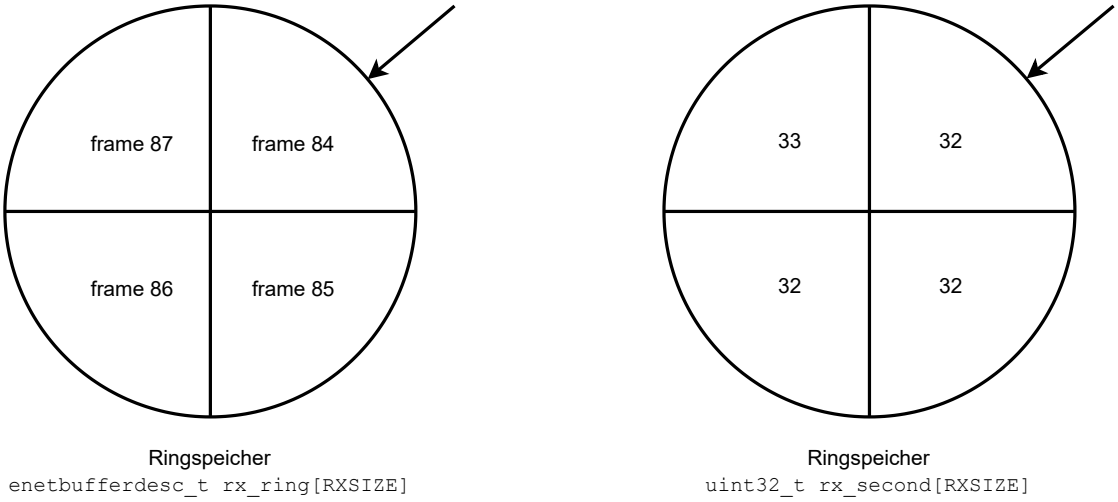


Abbildung 6.1: Beispiel für Inhalte der beiden Ringpuffer

Sendezeitstempel

Der Versand aller Ethernet-Rahmen auf dem Teensy erfolgt durch die Methode send_frame, deren Signatur in Listing 6.8 gezeigt wird.

```
1    void send_frame(void* frame, unsigned int len, bool timestamp);
```

Listing 6.8: Signatur der Methode zum Versand von Rahmen

Durch den Boolean timestamp kann angegeben werden, ob für den zu versendenden Rahmen ein Sendezeitstempel angefordert werden soll. So können für die für die Messunge relevanten Rahmen Zeitstempel angefordert werden, beispielsweise ARP– oder Ping-Antworten dagegen werden mit timestamp=false versandt.

Die Methode `send_frame` kopiert die in `len` spezifizierte Anzahl Bytes in den Ringspeicher für ausgehende Rahmen. Sofern durch den aufrufenden Code ein Sendezeitstempel angefordert wird, wird das entsprechende Bit im *Enhanced Transmit Buffer Descriptor* [Sem21, S. 2127 f.] gesetzt. Durch setzten eines „Ready“-Bits im Buffer Descriptor wird der Hardware signalisiert, dass der Rahmen bereit zum Versenden ist.

Nach Versand ist der Zeitstempel des letzten versandten Rahmens (mit gesetzten Timestamp-Bit) im Register ENET_ATSTMP zu finden; das Bit TS_AVAIL im *Event Interrupt Register* signalisiert, dass ein neuer Zeitstempel vorhanden ist und löst einen Interrupt aus, im dem dieser ausgelesen werden kann. ENET_ATSTMP enthält nur den Wert des FRC, also die Nanosekunde des Versands, der Wert der Sekunde muss mittels Software im Interrupt ermittelt werden.

Empfangszeitstempel

Sobald ein Rahmen empfangen wird, erfasst die Sicherungsschicht den Wert des Timers bei Erkennung des *Start of Frame Delimiters*. Dieser wird im Empfangsringspeicher als Bestandteil der eingegangenen Frames gespeichert. Der Wert des Timers gibt allerdings nur die aktuelle Nanosekunde an. Der Wert der aktuellen Sekunde muss seperat erfasst werden, im Empfangsringspeicher `rx_ring` ist diese Information nicht enthalten! Daher wird ein paralleler Ringspeicher `rx_second`, der die Sekunden des Empfangs von Paketen erfasst, eingerichtet. Abbildung 6.1 zeigt einen Möglichen Inhalt dieses Ringspeichers. Die Ringspeicher sind dargestellt mit einer Größe `RXSIZE` von 4, dies dient nur der Übersichtlichkeit der Darstellung, in der Implementierung ist `RXSIZE=12`. Direkt nach Empfang eines Rahmens wird ein Interrupt ausgelöst. Dieser wird genutzt, um die aktuelle Sekunde in den parallelen Ringspeicher `rx_second` zu schreiben.

Der Code in Listing 6.9 zeigt die Methode, die empfangene Messungen analysiert und die relevanten Informationen in der Datenstruktur `rcv_measurements` speichert. Zeilen 3, 4 zeigen das Auslesen der Werte für Sekunde und Nanosekunde des Empfangszeitpunktes.

```
1  void handle_incoming_measurement(int rxnum, char *udp_payload, rcv_measurement_t
   *rcv_measurements)
2  {
3      uint32_t rx_s = rx_second[rxnum];
4      uint32_t rx_ns = rx_ring[rxnum].timestamp;
5
6      char *sending_device = strtok(udp_payload, ", -#");
7      char *sending_device_number = strtok(NULL, ", -#");
8      char *measurement_number = strtok(NULL, ", -#");
9
10     /* Some checks to ensure the received measurement is valid. */
11
12     uint8_t sending_device_number_int = (uint8_t)atoi(sending_device_number);
13     uint32_t measurement_number_int = atoi(measurement_number);
14     uint32_t index_in_measurements = measurement_number_int % MEASUREMENTS_PER_REPORT;
15
16     measurement *current_measurement;
17
18     if (sending_device_number_int == 1)
19     {
20         current_measurement = &rcv_measurements[index_in_measurements].chan0;
21     }
22     /* Procedure is analog for 2, 3. Returns for other values. */
23
24     (*current_measurement).number = measurement_number_int;
25     (*current_measurement).second_tv.s = rx_s;
26     (*current_measurement).second_tv.ns = rx_ns;
27     (*current_measurement).validation.second_event_occurred = true;
28 }
```

Listing 6.9: Teensy - Handling eingehender Messungen

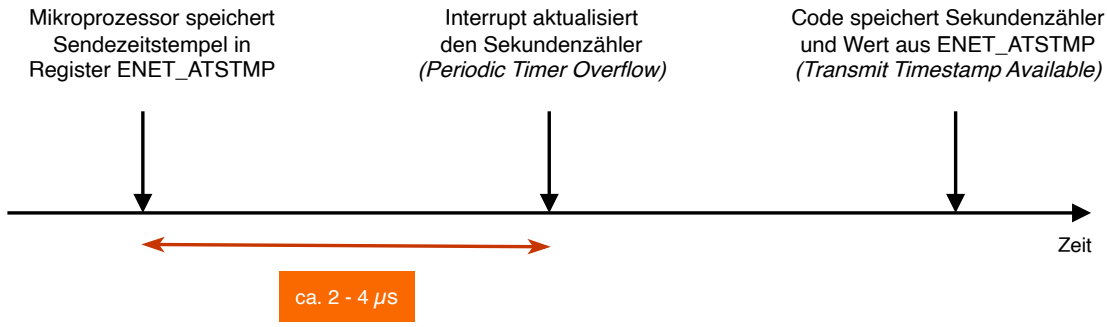


Abbildung 6.2: Zeitlicher Verlauf der Interrupts in Senderichtung, die zu falschem Wert des Sekundenzählers führen.

6.3.3 Timing verschiedener Interrupts

Die folgenden Ereignisse werden alle in der selben Interrupt Service Routine behandelt, da sie alle die *Interrupt Request* 115 auslösen [Sem21, S. 48 f.]:

1. *Periodic Timer Overflow.* Der Überlauf des Nanosekundenzählers, also der Beginn einer neuen Sekunde.
2. *Receive Frame Done.* Ein Rahmen wurde empfangen und der dazugehörige Buffer Descriptor aktualisiert.
3. *Transmit Timestamp Available.* Zeigt an, dass der Zeitstempel für den letzten übertragenen Rahmen im Register ENET_ATSTMP verfügbar ist.

Nun kann es vorkommen, dass der Wert des Sekundenzählers zwischen dem Zeitpunkt, an dem der Mikrocontroller den FRC in ENET_ATSTMP latcht, und dem Auslesen von Sekunde und Nanosekunde der Sekundenzähler bereits um eins erhöht wurde. Abbildung 6.2 zeigt den zeitlichen Ablauf der Ereignisse bei Versand eines Frames. Problematisch wird dies, wenn der Zähler der Nanosekunde bereits kurz davor ist, zurückgesetzt zu werden (wenige Mikrosekunden).

Ein beobachtetes Beispiel war: In Sekunde 2 535 s wird der Wert 999 997 240 ns in das Register ENET_ATSTMP gespeichert. Fälschlicherweise wird aber der Zeitstempel 2 536 s, 999 997 240 ns gemeldet; da der Sekundenzähler bis zum Zeitpunkt der Ausführung des speichernden Codes bereits um eins erhöht wurde.

Analog kann das Problem in Empfangsrichtung auftreten. In diesem Fall wird zuerst die Nanosekunde des Empfangs in den Empfangsringspeicher geschrieben, dann erhöht sich der Sekundenzähler um eins. Erst danach erfolgt der Interrupt *Receive Frame Done*, in welchem der Zeitwert bestehend aus Sekunde und Nanosekunde in der Datenstruktur `rcv_measurements` gespeichert wird.

Dies ließe sich auf Seite des Teensy beispielsweise durch genaue Kenntnis der Interrupt-Timings auf Hardwareebene lösen, die aufwendig zu ermitteln wären. Alternativ könne man Ressourcen sperren, und das Problem so vermeiden – besonders in Empfangsrichtung gestaltet sich dies aber kompliziert, da viele verschiedene Speicherstellen mit Zugriff teils in Hardware, teils in Software beteiligt sind. Stattdessen wird ausgenutzt, dass fälschlicherweise erhöhten Sekundenwerte sehr

einfach zu erkennen sind: Der empfangende Code prüft die übertragenen Werte auf Plausiblität. Ist eine Latenz negativ oder größer als eine Sekunde, wird der Wert zur manuellen Prüfung ausgegeben – was unproblematisch ist, da nur etwa $3 \cdot 10^{-3}\%$ der Werte betroffen sind.

6.4 Mitschnitt und Analyse der WiFi-Pakete

In Unterabschnitt 5.6.4 wurde die Anfertigung von Paketaufzeichnungen vorgestellt. In diesem Abschnitt wird die Umsetzung der Aufzeichnung knapp beschrieben.

Zur Erfassung der WiFi-Pakete wird ein Lenovo ThinkPad T460s verwendet, dieses enthält den Chipsatz „Intel Corporation Wireless 8260 (rev 3)“ . Für das Debian-Betriebssystem enthält das Paket „firmware-iwlwifi“ die notwendigen Treiber; für die Installation des Pakets muss der „non-free“-Bereich des Paketarchivs in den Paketquellen enthalten sein. Die Drahtlosfunktionalität ist zunächst deaktiviert, das Gerät muss mit `rfkill unblock` aktiviert werden.

Die Aufzeichnung der WLAN-Pakete ermöglichen die zwei folgenden Tools:

1. airmon-ng. Mithilfe dieses Tools kann der Monitor-Modus einer WLAN-Schnittstelle (sofern durch die WLAN-Schnittstelle unterstützt) aktiviert werden. Der Befehl airmon-ng <Schnittstellenname> <Kanal> aktiviert den Monitor-Modus und erzeugt eine virtuelle Schnittstelle, mit der der angegebene Kanal abgehört werden kann. Diese hat den Suffix „mon“, aus der Schnittstelle wlp4s0 wird also die zusätzliche virtuelle Schnittstelle wlp4s0mon erzeugt.
2. netsniff-ng. Das Tool netsniff-ng ist ein freies Sofwarewerkzeug für Linux, mit dem der eingehende Netzwerkverkehr einer Netzwerkschnittstelle in Form einer PCAP-Datei gespeichert und ggf. wiederholt werden kann [23e].

Listing 6.10 zeigt die notwendigen Befehle, um eine Aufzeichnung des WLAN-Verkehrs in die Datei „demo.pcap“ zu speichern. Zunächst wird das Radio-Interface mit Index 1 – beim verwendeten Rechner die WiFi-Schnittstelle – aktiviert. Dann wird ein virtuelles Gerät mit Monitor-Modus für das WLAN-Interface wlp4s0 angelegt, welches Kanal 11 abhört. In der letzten Zeile wird die Aufzeichnung gestartet, diese kann mit der Tastenkombination Ctrl + C beendet werden. Dabei werden die Pakete auch auf der Konsole ausgegeben, dies kann durch die Flagge „--silent“ unterdrückt werden.

```
1        $ sudo rfkill unblock 1
2        $ sudo airmon-ng start wlp4s0 11
3        $ sudo netsniff-ng --in wlp4s0mon --rfraw --out demo.pcap
4
```

Listing 6.10: Aufzeichnung per WiFi-Sniffer

Nach Ausführung der in Listing 6.10 genannten Befehle liegt eine .pcap-Datei vor. Diese kann mithilfe des Programms Wireshark [23d] betrachtet werden. Wireshark (früher Ethereal) ist ein freies Programm zur Analyse von Netzwerkprotokollen. Neben der Anzeige der aufgezeichneten .pcap-Dateien bietet es umfangreiche Möglichkeiten, diese zu filtern oder einzelne Pakete hervorzuheben.

7 Evaluation

Das folgende Kapitel 7 beginnt mit einer knappen Betrachtung der Genauigkeit des entworfenen Messsystems an sich. Diese wird anhand der gelieferten Messdaten von zwei der vorgestellten Szenarien durchgeführt.

In den darauf folgenden Abschnitten wird das Messsystem verwendet, um für die im Entwurf vorgestellten Szenarien jeweils Latenzen zu erfassen. So werden unterschiedliche Schlüsse in Hinsicht auf Zuverlässigkeit, Reduktion der Latenz kritischer Rahmen, der Effektivität des Einsatzes von EDCA und zuletzt des Schedulings mehrerer Sender gezogen. Für jeden der durchgeführten Versuche werden die Ergebnisse anhand von Diagrammen vorgestellt und diskutiert.

7.1 Evaluation der Latenz über Ethernet und Messgenauigkeit

In diesem Abschnitt werden Auflösung und Genauigkeit des Messsystems diskutiert. Dies geschieht anhand der Basismessung des Szenarios zweier per Ethernet verbundener Mikrocontroller (Szenario 0, Unterabschnitt 5.4.1) sowie einer Betrachtung des zeitlichen Abstands der Triggersignale (Szenario 3, Unterabschnitt 5.5.1).

7.1.1 Auswertung der Übertragung rein über Ethernet

Abbildung 7.1 zeigt die gemessenen Ende-zu-Ende-Latenzen der Übertragung rein per Ethernet als Histogramm sowie die empirische kumlative Verteilungsfunktion. Gemessen wurden die Latenzen für 100.000 versandte Rahmen, auf 100 Mess- und Übertragungsphasen aufgeteilt. Der Median der Latenzen beträgt 32,24 µs, die Standardabweichung nur 0,08 µs.

Die Ethernet-Uhr taktet mit einer Frequenz von 25 MHz, hat also eine Periodendauer von 40 ns. Diese Messauflösung ist im Histogramm gut zu erkennen. Auch die gemessenen Latenzen treten somit nur als ganzzahlige Vielfache von 40 ns auf. Dadurch ist in der empirischen kumulativen Verteilungsfunktion die Stufenbildung zu erkennen – diese sind nicht der Darstellung geschuldet, sondern resultiert aus der gegebenen Messauflösung. Etwa eine Latenz von 32 300 ns wird nie gemessen – da 32300 kein ganzzahliges Vielfaches von 40 ist. Dieser Wert hat im Histogramm somit die Häufigkeit 0. Diese Messergebnisse bestätigen, dass die zur Erfassung der Zeitstempel eingesetzte Ethernet-Uhr tatsächlich mit der gewünschten Frequenz in Betrieb ist, und zeigen nebenbei, dass die Latenz der Übertragung über Ethernet sehr gering ist.

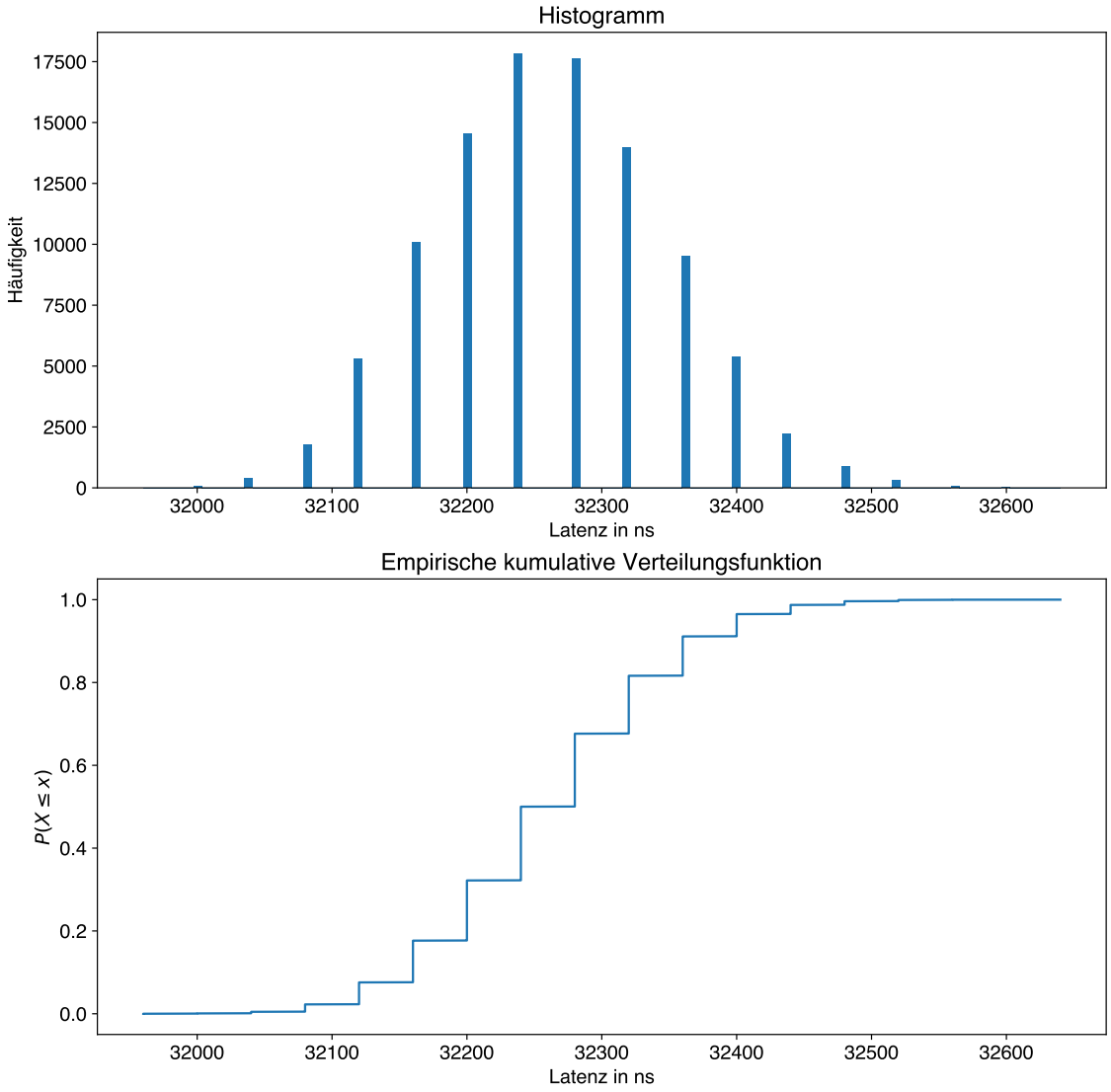


Abbildung 7.1: Auswertung der Übertragung von Rahmen rein über Ethernet

7.1.2 Genauigkeit der Messungen anhand der Auswertung des Triggersignals

Eine Quelle für Ungenauigkeit der Messungen ist nicht nur die reine Auflösung, sondern auch die Genauigkeit der taktgebenden Uhr an sich. Verwendete Quarze vergleichbarer Mikrocontroller haben eine Genauigkeit von 30ppm, verschiedene Versionen des Teensy 4.1 wurden auf -7 ppm bis -11 ppm vermessen[1].

[1] Auf GitHub veröffentlichte Messdaten: https://github.com/manitou48/crystals

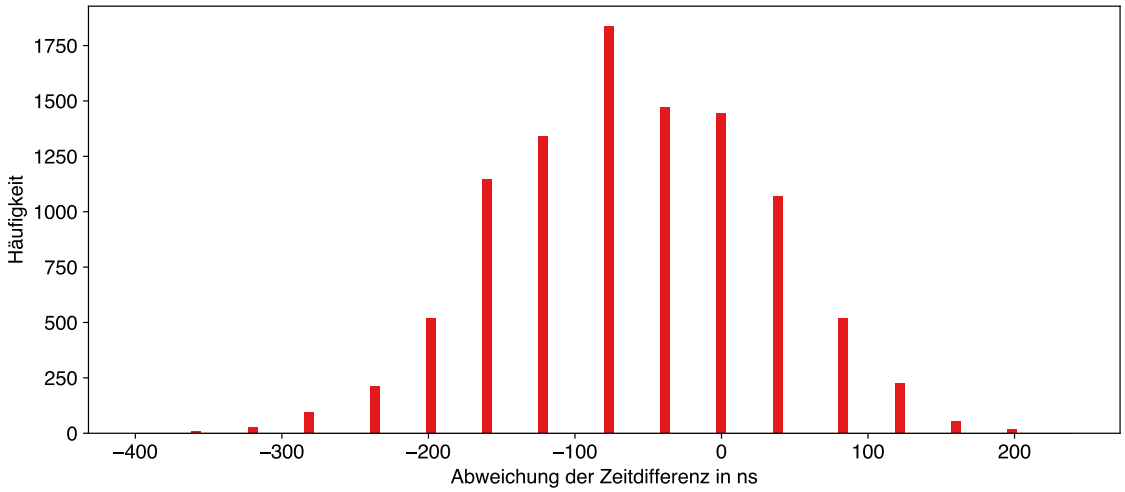


Abbildung 7.2: Histogramm der Abweichung der Differenz aufeinanderfolgender Triggersignale von der konfigurierten Periodendauer der TSN-Bridge

In diesem Abschnitt wird die Genauigkeit der Uhr durch eine Analyse des Triggersignals betrachtet. Kommt der ESP32 als Sender zum Einsatz, ist das Triggersignal das Signal, welches den Versand auslöst. Es wird im Entwurf eingeführt, um Mikrocontroller mit der Periode einer TSN-Bridge zu synchronisieren.

Gemessen wird nun die zeitliche Differenz aufeinanderfolgender Triggersignale für eine Periodendauer des Schedules der TSN-Bridge von 35 ms. Dazu kommt der Versuchsaufbau aus Szenario 4 (Unterabschnitt 5.5.2) zum Einsatz – nur diesmal nicht mit dem Ziel die Latenzen der Übertragung der Rahmen zu messen; stattdessen wird mittels Input Capture der Zeitpunkt der Triggersignale erfasst. Ohne jegliche Form der Ungenauigkeit könnte man erwarten, dass innerhalb jeder Messphase das $k + 1$. Triggersignal exakt 35 000 000 ns nach dem k-ten Triggersignal erfolgt. Abbildung 7.2 zeigt das Histogramm der Abweichung der gemessenen Zeitdifferenzen von dieser Erwartung. Die Beobachtung deckt sich mit den verfügbaren gemessenen Daten, dass das Quarz des Teensys bei Zimmertemperatur ein wenig zu langsam läuft – auch wenn die gezeigte Abweichung nicht allein auf die Uhr des Teensys zurückzuführen ist, sondern mehrere Auslöser hat:

1. Die Ungenauigkeit der Ethernet-Uhr des Teensys, mit der die Zeitstempel erfasst werden.
2. Die Ungenauigkeit der Uhr der TSN-Bridge, sodass die Periodendauer nicht exakt eingehalten wird.
3. Jitter der Dauer zwischen Öffnen des Gates der TSN-Bridge und Anzeige des Erhalts des Rahmens am GPIO-Pin des Triggers.

Mithilfe dieser Daten lässt sich trotzdem die Ungenauigkeit der Ethernet-Uhr an sich nach oben abschätzen. Alle Ungenauigkeiten zusammenaddiert führen zu den dargestellten Werten. Deren Median liegt bei $-62{,}01$ ns, die Standardabweichung bei 89,14 ns. Kernaussage der Analyse dieser Daten ist: Alle Ungenauigkeiten zusammen resultieren in einer Standardabweichung von weniger als einem Zehntel einer Mikrosekunde. Gegenüber den gemessenen WiFi-Latenzen, die sich im Bereich mindestens mehrerer hundert Mikrosekunden befinden, ist die Ungenauigkeit des Erfassungssystems also vernachlässigbar.

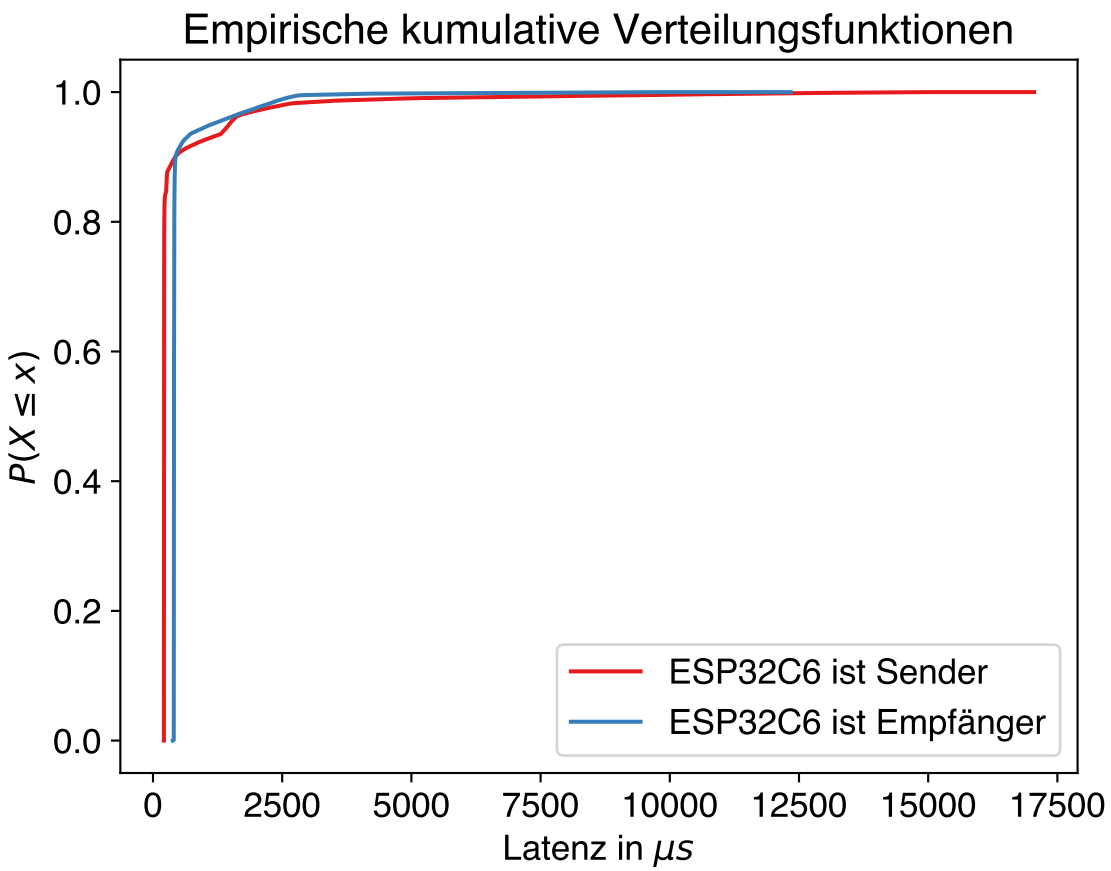


Abbildung 7.3: Empirische kumulative Verteilungsfunktionen beider Übertragungsrichtungen der Basismessung

7.2 Basismessungen

Im folgenden Abschnitt werden die Basismessungen diskutiert; also die Übertragung über einen WiFi-Link ohne TAS wie in den Szenarien 1S, 1E (Unterabschnitt 5.4.2) aufgeführt. Dazu werden zwei Datensätze je 100.000 Messungen angefertigt.

Abbildung 7.3 zeigt zu Beginn die empirischen kumulativen Verteilungsfunktionen der Latenzen beider Übertragungsrichtungen. Auf den ersten Blick sind sich beide ähnlich, auf die Verschiedenheiten und deren Ursachen wird im Verlauf des Abschnitts noch eingegangen. Die Begriffe „Senderichtung“ und „Empfangsrichtung“ beziehen sich im Folgenden jeweils auf den ESP32-C6, das heißt, Senderichtung bezeichnet die Übertragung von ESP32-C6 zu Teensy (Szenario 1S), Empfangsrichtung die Übertragung von Teensy zu ESP32-C6 (Szenario 1E).

Zunächst wird aber deutlich, was bereits in den verwandten Arbeiten genannt wurde: Ist das Übertragungsmedium nur zu geringem Grade ausgelastet – die Messungen wurden nach Ende des Arbeitstages auf möglichst freiem Kanal angefertigt – kann die Latenz der Übertragung per

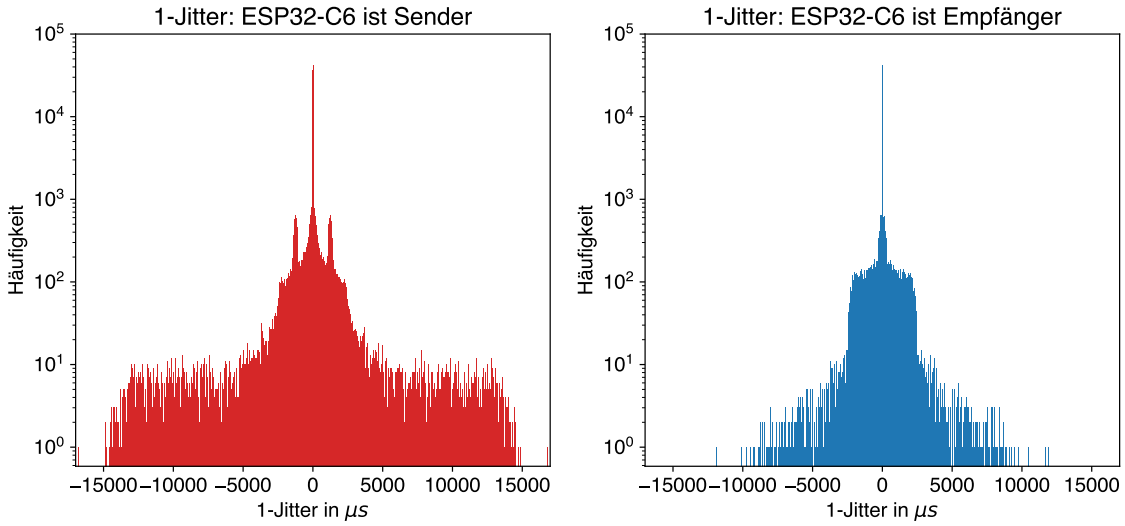


Abbildung 7.4: Darstellung der 1-Jitter Sende- und Empfangsrichtung

WiFi bereits sehr gering sein. In Senderichtung sind 92.643% der Latenzen kleiner als 1 ms, in Empfangsrichtung 94.574%. Kleiner als 5 ms sind in Senderichtung dann bereits 99.04%, in Empfangsrichtung 99.837% der Latenzen.

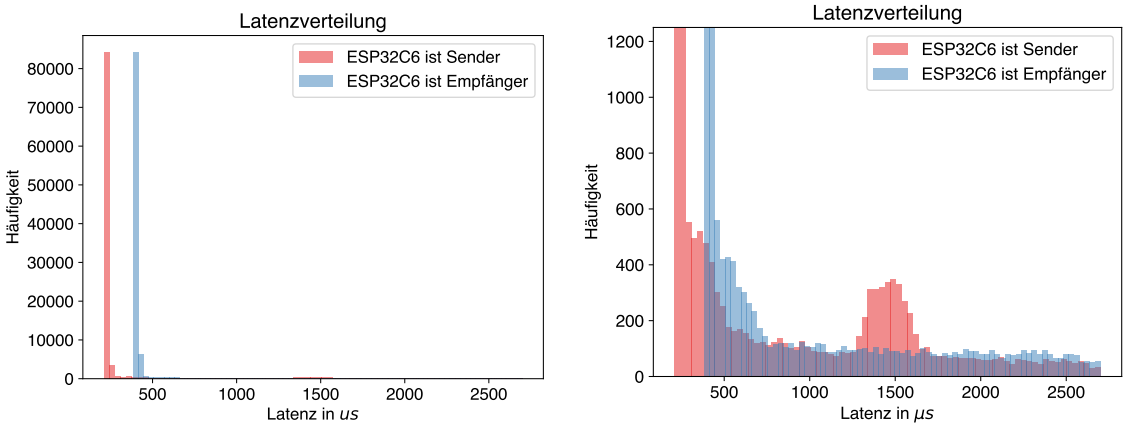
Das in [CCSR22] genannte Anwendungsgebiet von Multi-Player Onlinespielen erfordert bei einer beschränkten Latenz von maximal 10 ms eine Zuverlässigkeit von 99.9%. In Empfangsrichtung wird dies mit 99.995% erfüllt, in Senderichtung mit 99.589% knapp nicht erfüllt.

Insgesamt wird aber auch deutlich, dass die gemessenen Ende-zu-Ende Latenzen viel höher sind als in der zuerst gemessenen Übertragung per Ethernet. Die durchschnittliche Latenz liegt dort bei etwa 32 µs - allein die Dauer des DIFS und einer mittleren Random Backoff Time (Random() = 8 bei $CW_{min} = 15$) liegt mit DIFS + 8 · aSlotTime = SIFS + 10 · aSlotTime = 10 µs + 10 · 9 µs = 100 µs weit darüber, hinzu kommt die Zeit für die eigentliche Übertragung der Daten sowie die Verarbeitungszeiten auf Sender und Empfänger.

Abbildung 7.4 stellt die Werte des 1-Jitters (J_1) in einem Histogramm mit logarithmischer Skalierung der Y-Achse dar, so erhält man eine kompakte Darstellung des Jitters. Tabelle 7.1 zeigt die Werte für Standardabweichung σ der Latenz und Min-Max-Jitter $J_{min-max}$. Anhand dieser Messwerte wird deutlich, dass es bei Verwendung von WiFi deutlich schwieriger ist, deterministische Kommunikation zu erzielen. Es ist deutlich zu erkennen, dass die variable Paketverzögerung bei Verwendung des drahtlosen Links sehr viel höher wird.

	Ethernet	**Senderichtung**	**Empfangsrichtung**
σ	0,08 µs	1 030,49 µs	465,67 µs
$J_{min-max}$	0,68 µs	16 838,92 µs	11 957,96 µs

Tabelle 7.1: Standardabweichung und Min-Max-Jitter der Latenz bei Übertragung per WiFi



(a) Histogramm der Latenzen... **(b)** ...mit anderem Ausschnitt

Abbildung 7.5: Histogramme der Latenzen

Abbildung 7.5 zeigt das Histogramm der Latenzverteilung in zwei verschiedenen Ausschnitten. Beide zeigen nur Latenzen bis maximal 2 700 µs, auch, wenn, wie Abbildung 7.3 bereits dargestellt hat, ein Teil der Latenzen noch größer ist. Abbildung 7.5a hebt deutlich den Unterschied der Latenzen in Sende- und Empfangsrichtung hervor. Der Median der Latenzen in Senderichtung liegt bei 211 µs, der Median in Empfangsrichtung beträgt 406 µs – ein Unterschied fast von Faktor 2. Eine oft getroffene Annahme lautet, dass die Latenz in Sende- und Empfangsrichtung identisch ist. Daher ist ein so hoher gemessener Unterschied zumindest überraschend. Eine Analyse der parallel zur Latenzmessung angefertigten Paketaufzeichnungen des WiFi-Verkehrs deckt auf, dass die Asymmetrie der Latenzen nicht durch die WiFi-Übertragung an sich verursacht wird. In beide Richtungen wird für den Großteil der Pakete das HT PHY mit dem *Modulation and Coding Scheme (MCS) 7* verwendet. Daraus ergibt sich eine Datenrate von 72,2 Mbit s^{-1}. Die in Wireshark angezeige Übertragungsdauer des Rahmens auf der Bitübertragungsschicht (Attribut wifi_radio.duration) beträgt in beide Senderichtungen 47 µs.

Stattdessen lässt sich ein Teil der Ursache der Asymmetrie in der Implementierung des Netzwerkstacks des ESP32 vermuten. Wie in Kapitel 5 geschildert wurde, wird die Latenz bis hin zur Anwendungsebene gemessen. Das heißt insbesondere, dass die Latenz aller darunter liegenden Schichten mitgemessen wird. Die WiFi-Dokumentation des ESPs [Esp23a] erläutert den Fluss der Daten, dieser ist in Abbildung 7.6 dargestellt. Nun liegt die Vermutung nahe, dass sich die Ausführungszeit des Codes in Sende- und in Empfangsrichtung unterscheidet. Eine Zeitdifferenz von 173 µs entspricht bei der vorliegenden CPU-Frequenz von 160 MHz 27680 Taktzyklen. Weiterer möglicher Einflussfaktor ist die interne Implementierung des Access Points, möglicherweise führt auch diese zu Unterschieden zwischen Sende- und Empfangslatenz.

Dies wäre beispielsweise für eine zukünftige PTP-Implementierung relevant: Das Precision Time Protocol basiert auf der Annahme, dass die Latenzen beider Übertragungsrichtungen identisch sind. Diese Annahme ist hier offensichtlich nicht erfüllt – eine PTP-Implementierung basierend auf der vorliegenden Zeiterfassung hätte also absehbar eine schlechte Performanz.

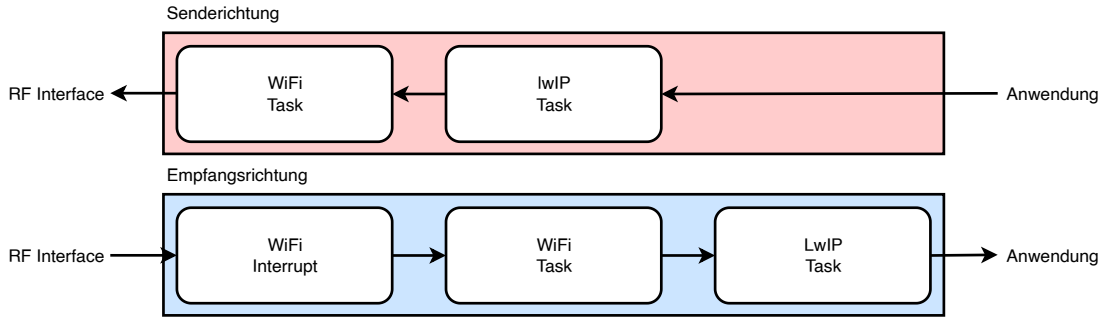


Abbildung 7.6: Darstellung des Datenpfades in Sende- und Empfangsrichtung

Eine zweite Beobachtung wird in der Darstellung der selben Daten mit anderem Ausschnitt in Abbildung 7.5b deutlich: In Senderichtung tritt eine erneute Häufung der Latenzen um ca. 1 500 µs auf, die in Empfangsrichtung nicht zu beobachten ist.

Die Paketaufzeichnung legt offen, dass es sich in Empfangsrichtung nur bei ca. 0.08% der Rahmen um einen Retransmit handelt, während es sich in Empfangsrichtung bei 2.54% der Rahmen um einen Retransmit handelt. Da diese Daten auf einem dritten Gerät erfasst wurden, können sie allerdings nur eine Abschätzung der tatsächlichen Werte des Prozentsatzes an Retransmits liefern. Eine mögliche Erklärung für dieses Verhalten wäre, dass der Access Point – da er weniger Beschränkungen in Bauform und beispielsweise Stromverbrauch hat – eine bessere Performanz beim Abhören des Mediums liefert, ein belegtes Medium so besser erkennt und Kollisionen effektiver vermeiden kann.

Die Paketaufzeichnung legt weiterhin offen, dass der ESP32-C6 in der Standardkonfiguration des WiFi-Treibers Frames als *Aggregate MAC Protocol Data Unit (AMPDU)* Frames versendet. AMPDU dient eigentlich einer Erhöhung des Durchsatzes, indem mehrere Rahmen in einen zusammengefasst werden – im vorliegenden Fall erhöht es jedoch die durchschnittliche Latenz. Abbildung 7.7 zeigt, dass AMPDU auch einen Einfluss auf die Acknowledgements hat. *Block Acknowledgements (Block Acks)* ermöglichen, mehrere Rahmen mit nur einem Block Ack zu bestätigen. Wie sichtbar ist, wird trotzdem jeder einzelne Rahmen mit einem Block Ack bestätigt. Kommt ein Rahmen jedoch nicht an – wie in der Abbildung derjenige, mit UDP-Payload „ESP-01#005432“ – fragt der ESP zunächst ein Block Ack an (*Block Ack Request*). Die Antwort enthält dann ein Block Ack, aus dem hervorgeht, dass dieser Rahmen nicht empfangen wurde. Erst dann wird der verloren gegangen Rahmen neu übertragen. Die Block Ack Request und die darauf folgende Block Ack werden mit einer Datenrate von nur 1 Mbit s^{-1} versandt, so dass dieser Vorgang einen langen Zeitraum in Anspruch nimmt. Die erst nach Abschluss dieses Vorgangs aus Block Ack Req und Block Ack neu übertragenen Rahmen sind diejenigen, die nach etwa 1 500 µs nach Aufruf der Socket-API schließlich ankommen und so die sichtbare Häufung um diese Latenz in Abbildung 7.5b auslösen. Auch ohne die Details von AMPDU bzw. Block Acks im Detail behandelt zu haben, verdeutlicht dies exemplarisch, dass einzelne verwendete Features des WiFi-Standards sich spürbar auf die Latenz auswirken können. Vergleichend wird im nachfolgenden Unterabschnitt die empirische kumulative Verteilungsfunktion ebenfalls des ESP32-C6 als Sender, aber mit AMPDU deaktiviert gezeigt.

Zuletzt stellt Tabelle 7.2 die Werte der Latenzen in Sende- und Empfangsrichtung für einige Quantile dar.

Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005430
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C
Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005431
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C
Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005432
Espressi_40:43:38 …	BelkinIn_10:b1:a0 …	802.11	802.11 Block Ack Req, Flags=........C
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C
Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005432
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C
BelkinIn_10:b1:a0	Broadcast	802.11	Beacon frame, SN=3860, FN=0, Flags=........C, BI=100, S…
Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005433
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C
Espressi_40:43:38	10.1.100.88	UDP	ESP-01#005434
BelkinIn_10:b1:a0 …	Espressi_40:43:38 …	802.11	802.11 Block Ack, Flags=........C

Abbildung 7.7: Ausschnitt der Paketaufzeichnung

7.2.1 Basismessung unter Variation von Parametern

Im vorangegangenen Versuch stellte sich heraus, dass ein Teil des Unterschiedes zwischen Senderichtung und Empfangsrichtung auf den Versand von AMPDU-Frames bzw. der Verwendung von Block Acknowledgements entstand. Daher wird im anschließenden Versuch AMPDU deaktiviert.

Die Abbildung 7.8 zeigt zunächst anhand der beiden empirischen kumulativen Verteilungsfunktionen, dass sich die Latenz durch die Deaktivierung des Features tatsächlich senkt. Die durchschnittliche Latenz sinkt von 424,04 µs auf 288,91 µs, das 95%-Quantil von 1 473,56 µs auf 630,20 µs.

Interessant ist auch, dass im gewählten Ausschnitt des Histogramms nun die Häufung der Latenzen der Retransmits zu sehen ist. Die einzelnen Peaks sind jeweils 9 µs voneinander entfernt – genau die Slotzeit des HT PHYs (siehe Technischer Hintergrund, Unterunterabschnitt 2.1.2). Der Access Point hat ein minimales Contention Window $CW_{min} = 15$, d.h. beim ersten Retransmit ist $CW = 31$. Das Histogramm zeigt auch genau 31 Peaks – was zumindest einen Teil der Erklärung offen lässt, da nach der Definition der Random()-Funktion gemäß [IEE16, Gleichung 10-1] eine Zufallszahl

Latenzen für Sende- und Empfangsrichtung					
Quantil	0%	10%	20%	30%	40%
Latenz in Senderichtung (μs)	210.04	211.04	211.16	211.24	211.48
Latenz in Empfangsrichtung (μs)	383.36	404.84	405.16	405.32	405.88
Quantil	50%	60%	70%	80%	90%
Latenz in Senderichtung (μs)	211.72	212.08	213.48	218.08	430.84
Latenz in Empfangsrichtung (μs)	406.44	407.44	410.12	412.44	435.84
Quantil	95%	99%	99.9%	99.99%	100%
Latenz in Senderichtung (μs)	1473.56	4817.12	13145.96	14606.16	17048.96
Latenz in Empfangsrichtung (μs)	1123.93	2542.04	6197.57	9208.85	12341.32

Tabelle 7.2: Quantile der Latenzen für Szenario 1S, 1E

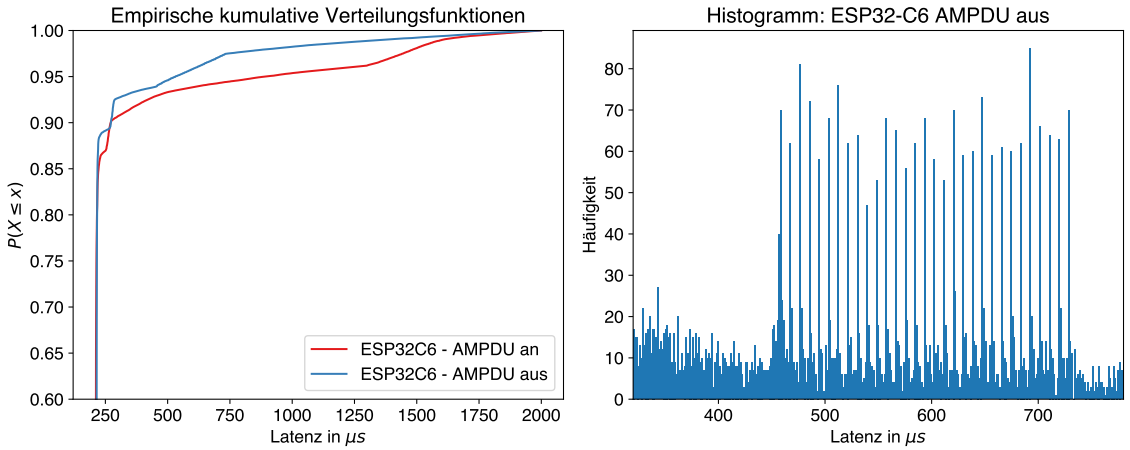


Abbildung 7.8: Szenario 1S, AMPDU deaktiviert

aus dem Intervall [0, CW] gezogen wird; somit müssten eigentlich 32 Peaks zu sehen sein. Es ist darauf hinzuweisen, dass beide Diagramme speziell gewählte Ausschnitte zur Demonstration der genannten Besonderheiten darstellen, man beachte also die Beschriftung der Achsen.

7.3 Enhanced Distributed Channel Access (EDCA)

In diesem Abschnitt erfolgt die Evaluierung des in Unterabschnitt 5.4.3 vorgestellten Versuchs zur Bewertung des Einflusses der vier Zugriffskategorien von EDCA (Szenario 2). Da die Messung für die vier Zugriffskategorien nacheinander stattfinden muss, werden, um externe Einflüsse zu minimieren, alle Versuchsreihen unmittelbar nacheinander nach Ende des Arbeitstages angefertigt. Für jede Zugriffskategorie werden 20.000 Rahmen übertragen.

Abbildung 7.9 zeigt die Histogramme der Latenzen für jede Zugriffskategorie. Bereits die Histogramme lassen vermuten, dass die vier Zugriffskategorien tatsächlich das Gewünschte leisten und eine Verbesserung der Dienstgüte für Rahmen, die mit einer Zugriffskategorie höherer Priorität übertragen wurden, erzielt wird. Um die Histogramme erkennbar zu halten werden nur Latenzen bis maximal 100 ms dargestellt, auch wenn einige Latenzen noch höher sind.

Dies ist für sich alleine gesehen schon bemerkenswert – lag doch die höchste von 100.000 gemessenen Latenzen für die Basismessung des ESP32-C6 als Sender ohne Cross-Traffic bei 17,05 ms (ohne explizite Spezifikation einer Zugriffskategorie, d.h. „Best Effort“), liegt die maximale von 20.000 beobachtete Latenzen für die Best-Effort-Zugriffskategorie nun bei 184,731 ms; eine Erhöhung um mehr als den Faktor 10.
Die durchschnittliche Latenz erhöht sich von 0,424 ms auf 5,403 ms, was einen Anstieg um mehr als den Faktor 12 bedeutet – all das, bei einem einzigen parallel sendenden Gerät.

Die gemessenen Latenzen zeigen, dass durch Verwendung der EDCA-Zugriffskategorien der Effekt des Cross-Traffics auf die Latenz abgeschwächt werden kann. Abbildung 7.10 zeigt die empirischen kumulativen Verteilungsfunktionen je Zugriffskategorie.

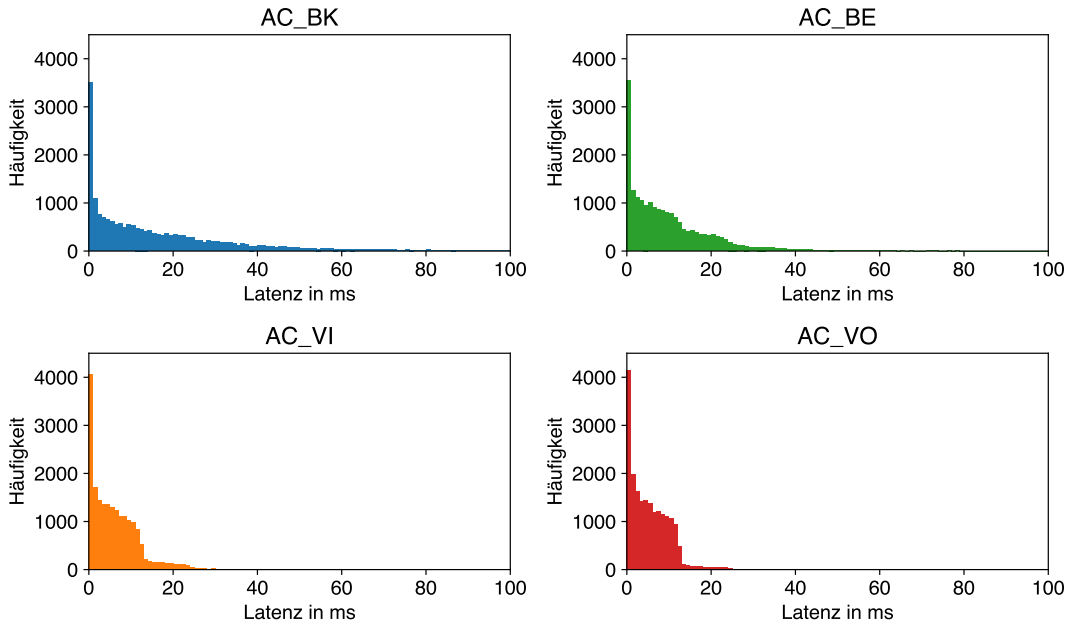


Abbildung 7.9: Histogramme der Latenzen der vier Zugriffskategorien

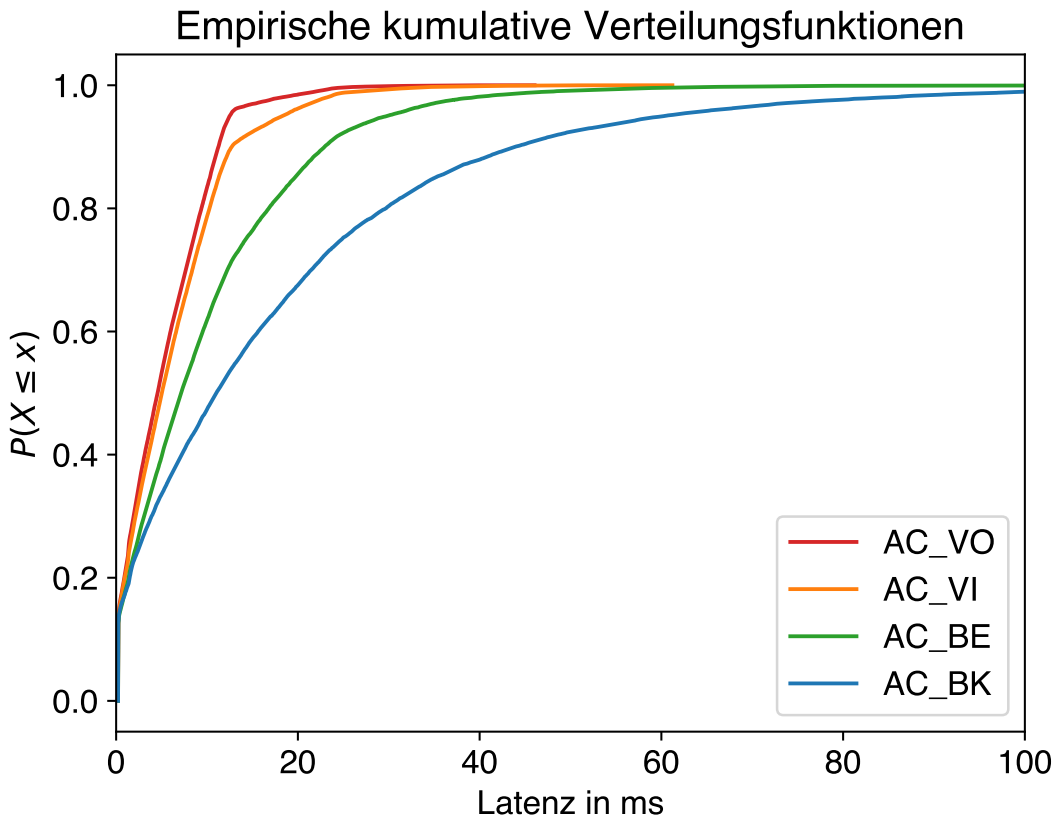


Abbildung 7.10: Empirische kumulative Verteilungsfunktion der Latenzen der vier Zugriffskategorien

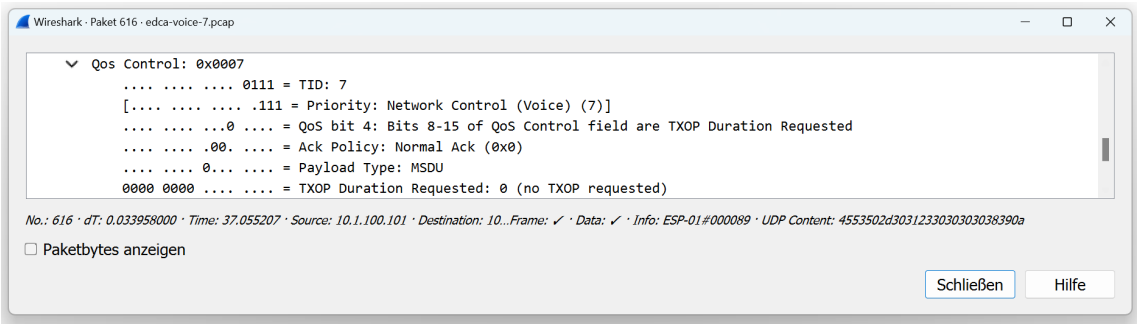


Abbildung 7.11: Ausschnitt der Parameter eines mit Zugriffskategorie Voice versandten Rahmens des ESP in Wireshark

Abbildung 7.11 zeigt einen Ausschnitt eines durch den ESP32-C6 übertragenen Rahmens der Zugriffskategorie Voice. Dies liefert Einblick in das Verhalten des ESP: Dieser scheint nur IFS sowie die Parameter CW_{min}, CW_{max} anzupassen, eine Anfrage von Transmit Opportunities (TXOP) konnte im Rahmen des durchgeführten Versuchs nicht beobachtet werden. (Es ist jedoch nicht auszuschließen, dass in anderen Versuchen, z.B. bei Verwendung von TCP, TXOPs angefragt werden würden. Dies wird in dieser Arbeit nicht untersucht.)

Bereits in Abbildung 2.3 wurden die EDCA-Parameter, die der verwendete Access Point in den Beacons ausstrahlt, gezeigt. Zur Erinnerung: Für die Zugriffskategorie Voice ist darin $CW_{min} = 3, CW_{max} = 7$. Das hat zur Folge, dass bereits nach der ersten Kollision die maximale Größe des Contention Windows erreicht ist. Der Random Backoff Timer wird folglich aus dem Intervall $[0, CW_{max}]$ – also 8 Werten – pseudozufällig gewählt.

Liegt nun ein Netz vor, in dem k Geräte das Priorisierungsverfahren EDCA verwenden und dabei $CW_{max} = 7$ für die verwendete Zugriffskategorie spezifiziert ist, ergibt sich für alle weiteren Retransmits eine Kollisionswahrscheinlichkeit P zwischen diesen Geräten von

$$\begin{aligned} P &= 1 - \frac{(CW_{max}+1)!}{(CW_{max}+1-k)! \cdot (CW_{max}+1)^k} \\ &= 1 - \frac{8!}{(8-k)! \cdot 8^k} \end{aligned}$$

Das ergibt bereits für 2 Sender eine Kollisionswahrscheinlichkeit von 12.5%, 34.375% für 3 Sender und schon knapp 56% für 4 Sender – das ist nur die Kollisionswahrscheinlichkeit der EDCA verwendenden Geräte untereinander, hinzu kommen Einflüsse durch andere Geräte oder die Umgebung. Da ja bereits $CW = CW_{max}$ gilt, haben auch alle weiteren darauf folgenden Retransmits die gleiche, hohe Kollisionswahrscheinlichkeit!

Insgesamt kann also festgehalten werden, dass die Verwendung von EDCA eine signifikante Verbesserung bringt – ein einzelner Sender erfährt deutliche Verbesserung der Dienstgüte, die maximale Latenz konnte experimentell deutlich gesenkt werden. Aufgrund der gerade betrachteten Beispielrechnung wird jedoch deutlich, dass es bei Verwendung durch mehrere Geräte nicht gut skaliert. Daher wird langfristig Unterstützung für weitere, besser Skalierende Verfahren im WiFi-Standard erforderlich.

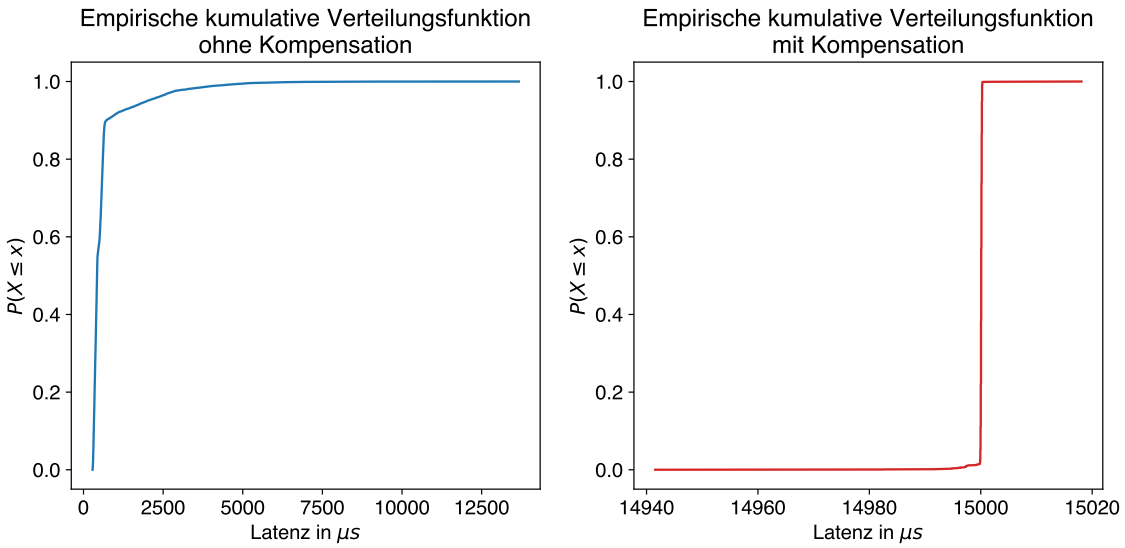


Abbildung 7.12: Empirische kumulative Verteilungsfunktion der Latenzen mit und ohne Kompensation der Latenzvarianz

7.4 Kompensation der variablen Paketverzögerung

In diesem Abschnitt wird der Einfluss des Time-Aware Shapers zur Kompensation der variablen Paketverzögerung betrachtet. Ein Hinweis vorweg: Die bisherigen Messungen wurden mit dem ESP32-C6 durchgeführt. Zu Ende des Abschnitts werden für den in Szenario 4 (Unterabschnitt 5.5.2) geschilderten Versuchsaufbau drei gleichartige Mikrocontroller benötigt. In allen folgenden Messungen kommt daher der ESP32 (statt des ESP32-C6) zum Einsatz, da nur dieser Mikrocontroller in ausreichender Stückzahl vorhanden ist. Die Option „WiFi AMPDU TX“ des WiFi-Treibers ist in allen nachfolgenden Versuchen ausgeschaltet.

Da erstmalig ein anderes Modell des Mikrocontrollers zum Einsatz kommt, erfolgt zu Beginn analog zur Basismessung eine Messung von 10.000 Rahmen ohne Kompensation der Latenzvarianz. Sodann kommt der Time-Aware Shapers zum Einsatz; dessen Auswirkungen wird diskutiert.

Nun bestehen verschiedene Möglichkeiten zur Auwahl einer Gate Control List. Sofern bereits die empirische kumulative Verteilungsfunktion eines Senders unter den vorliegenden Bedingungen des Netzwerks (Auslastung des Mediums etc.), gemessen ohne Kompensation der Latenzvarianz, vorliegt, kann man daraus bereits das Verhalten bei Hinzufügen der Kompensierung der Latenzvarianz ableiten: Für die empirische kumulative Verteilungsfunktion $P(X)$ und Latenz l kann man bei $P(X \leq l) = p$ davon ausgehen, dass sich auch nach einer Kompensationszeit l der Anteil p aller Pakete zum Zeitpunkt des Öffnen des Gates in der Warteschlange befindet.

Etwas deutlicher wird dies anhand von Abbildung 7.12: Die maximale beobachtete Latenz einer Basismessung des ESP32 liegt bei 13,67 ms. Wählt man die GCL also so, dass das die TSN-Bridge das Gate der Verkehrsklasse dieses Flusses erst 15 ms nach Aufruf der Socket-API zum Senden öffnet, liegt – statistische Unsicherheit außen vor lassend, die kumulative Verteilungsfunktion ist schließlich nur empirisch ermittelt – eine 100 %-ige Wahrscheinlichkeit vor, dass der Rahmen

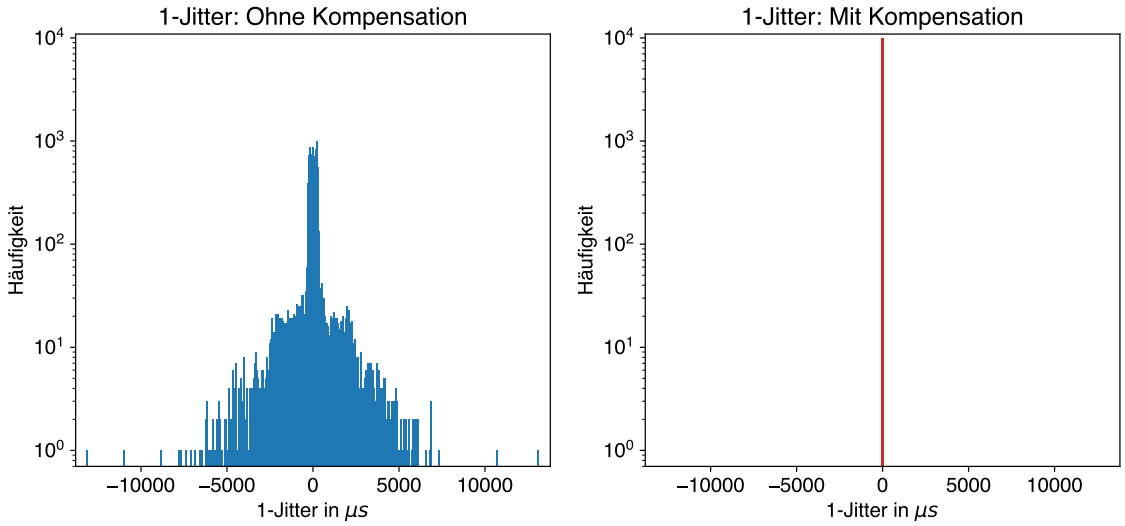


Abbildung 7.13: Histogramme der 1-Jitter mit und ohne Kompensation der Latenzvarianz

innerhalb dieser 15 ms die Warteschlange am ausgehenden Port der TSN-Bridge erreicht hat. Befinden sich dort keine anderen Rahmen vor ihm, kann er sofort mit Öffnen des Gates durch die TSN-Bridge weitergeleitet werden.

Somit wird die gesamte durch den WiFi-Link verursachte Latenzvarianz kompensiert. In Abbildung 7.13 wird dies anhand der Histogramme des 1-Jitters bereits deutlich. Die Key Performance Indikatoren bestätigen die Effektivität der Kompensation: Der Min-Max-Jitter sinkt von 13 382,92 µs ohne Kompensation auf 76,52 µs mit Kompensation. Die Standardabweichung der Latenz sinkt von 734,51 µs ohne Kompensation auf 1,12 µs mit Kompensation.

Die Latenz ist bekanntlich die Differenz zwischen Empfangszeitpunkt und Sendezeitpunkt. Der Sendezeitpunkt befindet sich nicht immer in exakt gleichem zeitlichen Abstand zur Auslösung des Triggersignals, also hat selbst schon Jitter. Auch dieser wird nun kompensiert. Betrachtet man also die Zeitdifferenz zwischen Ankunftszeitpunkt und Trigger-Signal (statt Ankunftszeitpunkt und Sendezeitpunkt), ist die Varianz dieser Zeitdifferenzen noch geringer. Die Standardabweichung sinkt von 1,12 µs auf 0,50 µs. Der Ankunftszeitpunkt lässt sich nun sehr genau eingrenzen; die Zuverlässigkeit ist also sehr hoch: 100% der Rahmen erreichen den Empfänger innerhalb eines Zeitfensters von 31,08 µs (min. 15 020,24 µs, max. 15 051,32 µs nach Auslösen des Trigger-Signals).

Die Verringerung der Latenzvarianz muss jedoch abgewogen werden gegen die Erhöhung der Latenz. Der hier gewählte Ansatz – das Gate so lange geschlossen zu halten, dass alle Rahmen die TSN-Bridge innerhalb dieses Zeitraums erreichen, ist in der Praxis oft nicht möglich. Die durchschnittliche Latenz steigt von 635 µs auf 15 000 µs. Im der Evaluation zu EDCA wurde gezeigt, dass die maximale Latenz der Zugriffskategorie AC_BE in der Gegenwart eines Senders, der das Medium zu hohem Grad auslastet, bei ca. 185 ms liegt – alle Rahmen für 185 ms zurückzuhalten ist kein praxistauglicher Ansatz. In der Praxis muss man also abwägen zwischen: Einerseits einer Gate Control List, die das Gate so lange geschlossen hält, bis sich zum Zeitpunkt des Öffnens des Gates

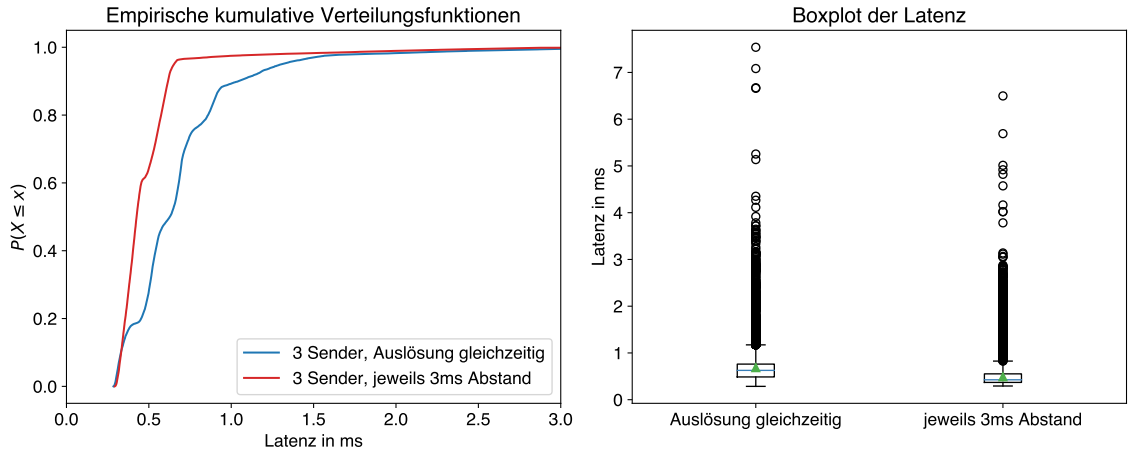


Abbildung 7.14: Auswirkung des Schedulings der Übertragungen

ein sehr hoher Prozentsatz der Rahmen in der Warteschlange befindet. Oder andererseits einer Gate Control List, die das Gate nur etwas kürzer geschlossen hält, sodass aber nicht alle Rahmen direkt zu Öffnen dieses Gates bereits in der Warteschlange befinden.

Um diese Abwägung so gestalten zu können, dass bei akzeptabler Rückhaltezeit durch das Gate eine hohe Zuverlässigkeit erzielt werden kann, muss also die Latenz der Übertragung über den WiFi-Link verringert werden. Dazu kann, wie im vorangegangene Abschnitt beschrieben wurde, EDCA verwendet werden, oder, wie im nächsten Abschnitt beschrieben wird, die Latenz durch Scheduling der Stationen verringert werden.

7.5 Reduktion der Latenz durch Scheduling der Übertragungen

In Szenario 4 (Unterabschnitt 5.5.2) werden mehrere Stationen mithilfe des Triggersignals synchronisiert. In der Implementierung wurde dies beschrieben; eine Sempahore synchronisiert den sendenden Task mit dem durch den Triggersignal ausgelösten Interrupt.

Nun werden, wie bereits im Entwurf eingeführt, die beiden Szenarien verglichen:

- Im sendenden Task wird unverzüglich gesendet. Alle drei Sender rufen dadurch *fast* gleichzeitig die Socket-API zum Versand des UDP-Datagramms auf. (Nur *fast* gleichzeitig, da auch die Dauer zwischen Taktflanke des Triggersignals über die Zwischenschritte der Interrupt Service Routine und der folgenden Kontextwechsel der FreeRTOS-Tasks bis hin zum tatsächlichen Aufruf der Socket-API eine gewisse Varianz hat.)
- Der sendende Task hält zunächst für eine gewisse Rückhaltezeit zurück.

Dieses Szenario wurde für einen Abstand von 3 ms zwischen den Sendern evaluiert. Das heißt, ein ESP verzögert gar nicht, der zweite ESP hält für einen Zeitraum von 3 ms zurück, der dritte ESP verzögert für 6 ms.

In Abbildung 7.14 sind die Empirischen kumulativen Verteilungsfunktionen sowie die Boxplots eines ESPs zu sehen. Dargestellt wird die Latenz eines der ESPs – betrachtet man die gleichzeitige Auslösung, spielt keine Rolle, welchen der drei Sender man analysiert, da alle den gleichen Code ausführen. Die Darstellung zeigt für das Szenario mit jeweils 3 ms Abstand die Latenz des zeitlich letzten Senders. Durchgeführt wurden jeweils 10.000 Latenzmessungen.

Die maximale Differenz der beiden empirischen kumulativen Verteilungsfunktionen $P(X \leq x)$ befindet sich bei $x = 637{,}44\,\mu s$. Ohne Verwendung der Wartezeit, also bei annähernd gleichzeitigem Aufruf der Socket-API, ist $P(X \leq 637{,}44\,\mu s) = 50.87\%$, mit zeitlichem Versatz der Sender schon $P(X \leq 637{,}44\,\mu s) = 93.70\%$. Der Median der Latenz sinkt von 626,96 µs ohne Scheduling auf 425,58 µs mit Scheduling, das entspricht einer Reduktion von etwa 32%.

Durch das Scheduling senkt sich auch $J_{min-max}$ von 7,25 ms auf 6,20 ms.

Bereits durch diese sehr einfach implementierte Art des Schedulings auf Anwendungsebene lässt sich also eine Verbesserung der Latenz erzielen, die beispielsweise in Kombination mit der Kompensation der variablen Paketverzögerung von großem Vorteil sein kann.

Zusammenfassung und Ausblick

Drahtlose Kommunikation unter Verwendung des WiFi-Standards ist heutzutage allgegenwärtig und hat die Gewohnheiten der Menschen drastisch verändert. Eine Vielzahl neuartiger Anwendungen etwa im Bereich von Industrie 4.0 und cyber-physischen Systemen, aber auch die alltägliche Anwendungen von Endkonsumenten, wie Onlinespiele oder AR/VR-Anwendungen, erfordern dabei zunehmend eine deterministische Güte der Übertragung.

Während die reine Datenrate des WiFi-Standards sich in den vergangenen Jahren immer weiter erhöht hat und inzwischen eine Übertragung mit bis zu $10\,\mathrm{Gbit\,s^{-1}}$ möglich ist, bleibt die Ermöglichung von Echtzeitkommunikation über WiFi eine offene Herausforderung. Zunächst sind präzise Aussagen über die bisher erzielten Latenzen notwendig – die genaue Messung der Latenzen einer Übertragungsrichtung über das Netzwerk gestaltet sich dabei bislang schwierig.

In dieser Arbeit wird ein System entworfen und implementiert, welches mithilfe von Mikrocontrollern als Sender und Empfänger die präzise Messung der Latenz einer Übertragungsrichtung ermöglicht und das Erstellen von Messungen so deutlich vereinfacht. Dazu kommt eine drahtgebundene Verbindung der GPIO-Pins zum Einsatz, so dass Sende- und Empfangszeitstempel mithilfe einer Uhr erfasst werden. Durch Verwendung von Input Capture und der Erfassung der Zeitstempel in Hardware kann eine hohe Messgenauigkeit erzielt werden.

Eines der Ziele des Einsatzes des Messsystems ist, die Veränderung der Übertragungseigenschaften durch Einsatz eines Time-Aware Shapers beispielsweise zur Kompensation der variablen Paketverzögerung zu charakterisieren. Dazu wird ein Konzept entworfen, um Mikrocontroller ohne die Notwendigkeit einer Implementierung des Precision Time Protocols an die Periode einer TSN-Bridge zu synchronisieren. Das Verfahren erlaubt darüber hinaus, auf mehreren Mikrocontrollern gleichzeitige Sendevorgänge auszulösen.

Zur Evaluation der Latenzen von WiFi-Links werden vier Szenarien eingeführt, die die entworfene Funktionalität zusammenfügen.

Dazu wird zunächst die Latenz der Übertragung über einen WiFi-Link bei möglichst freiem Übertragungsmedium charakterisiert. Im Anschluss wird die Effektivität von Enhanced Distributed Channel Access (EDCA), einem bereits im WiFi-Standard spezifiziertem Verfahren zur Differenzierung der Dienstgüte, ermittelt und bewertet. Die Resultate zeigen, dass EDCA geeignet ist, die Übertragungseigenschaften bei hoher Auslastung des Mediums deutlich zu verbessern.
Zwei der entworfenen Szenarien verwenden Konzepte des Time-Sensitive Networking. Die variable Paketverzögerung der Übertragung über einen WiFi-Link wird durch Einsatz des Time-Aware Shapers kompensiert. So kann die Standardabweichung der Latenz von $734\,\mu\mathrm{s}$ ohne Kompensation auf $1\,\mu\mathrm{s}$ mit Kompensation reduziert werden.
Mithilfe der Synchronisierung der Mikrocontroller können durch zeitliche Positionierung der Übertragungen mehrere Mikrocontroller die Latenzen aufgrund des Medienzugriffsverfahrens verringert werden. Der Median der Latenz sinkt durch Scheduling der Stationen so um 32%.

In dieser Arbeit werden Aspekte wie die Kompensation der variablen Paketverzögerung oder der Effekt des Schedulings der Übertragungen mehrerer Stationen jeweils eigenständig betrachtet. In zukünftigen Arbeiten könnten diese in Kombination betrachtet werden; so könnte beispielsweise Scheduling der Stationen, Priorisierung mithilfe von EDCA und und die Kompensation der

Latenzvarianz kombiniert werden, um eine weitere Verbesserung von Zuverlässigkeit und Latenz zu erzielen. Darüber hinaus ist die Betrachtung weiterer Parameter möglich, so könnte der Einfluss der Belegung des Mediums durch andere Protokolle als WiFi untersucht werden.

Bereits das Scheduling der Übertragungen mehrerer Geräte auf Anwendungsebene hat sich als effektiv erwiesen. In Hinblick auf die Weiterentwicklung des WiFi-Standards für 802.11be und den folgenden Standards kann darauf aufbauend die erreichbare Leistung durch Integration des Time-Aware Shapers in das Medienzugriffsverfahren analysiert werden.

Literaturverzeichnis

[14] „IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks“. In: *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)* (2014), S. 1–1832. DOI: 10.1109/IEEESTD.2014.6991462 (zitiert auf S. 36).

[18] „IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks“. In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (Juli 2018), S. 1–1993. DOI: 10.1109/IEEESTD.2018.8403927 (zitiert auf S. 25, 26).

[21a] „IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Redline“. In: *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016) - Redline* (2021), S. 1–7524 (zitiert auf S. 19).

[21b] „IEEE Standard for Jitter and Phase Noise“. In: *IEEE Std 2414-2020* (2021), S. 1–42. DOI: 10.1109/IEEESTD.2021.9364950 (zitiert auf S. 38).

[23a] *ESP32 Datasheet*. Espressif Systems. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (zitiert auf S. 27).

[23b] *ESP32-C6-WROOM-1 Datasheet*. Espressif Systems. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32-c6-wroom-1_wroom-1u_datasheet_en.pdf (zitiert auf S. 27).

[23c] *ESP32-WROOM-32 Datasheet*. Espressif Systems. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (zitiert auf S. 27).

[23d] *Homepage of wireshark*. Accessed on 18 July 2023. 2023. URL: https://www.wireshark.org/ (zitiert auf S. 69).

[23e] *netsniff-ng toolkit*. Accessed on 18 July 2023. 2023. URL: http://netsniff-ng.org/ (zitiert auf S. 68).

[3GP20] 3GPP. *Service requirements for cyber-physicla control applications in vertical domains (Release 16)*. Technical Specification TS 22.104. Version 16.5.0. European Telecommunications Standard Institute (ETSI), Sep. 2020 (zitiert auf S. 24, 29).

[3GP21] 3GPP. *Service requirements for the 5G system (Release 16)*. Technical Specification TS 22.261. Version 16.14.00. European Telecommunications Standard Institute (ETSI), Apr. 2021 (zitiert auf S. 24, 38).

[ACB21] T. Adame, M. Carrascosa-Zamacois, B. Bellalta. „Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7“. In: *Sensors* 21.15 (2021), S. 4954 (zitiert auf S. 30).

[BC13] S. Banerji, R. S. Chowdhury. „On IEEE 802.11: wireless LAN technology“. In: *International Journal of Mobile Network Communications and Telematics* (2013) (zitiert auf S. 18, 22, 29).

[Cas14] M. Castells. „The impact of the internet on society: a global perspective“. In: *Change* 19 (2014), S. 127–148 (zitiert auf S. 15).

[CCSR22] D. Cavalcanti, C. Cordeiro, M. Smith, A. Regev. „WiFi TSN: Enabling Deterministic Wireless Connectivity over 802.11“. In: *IEEE Communications Standards Magazine* 6.4 (2022), S. 22–29 (zitiert auf S. 24, 29, 30, 50, 75).

[CKY20] E. Chang, H. T. Kim, B. Yoo. „Virtual reality sickness: a review of causes and measurements“. In: *International Journal of Human–Computer Interaction* 36.17 (2020), S. 1658–1682 (zitiert auf S. 15).

[Com12] W. A. T. Committee. *Wi-Fi Multimedia Technical Specification, Version 1.2.0*. Techn. Ber. 2012. DOI: 10.1109/IEEESTD.2016.7786995 (zitiert auf S. 23).

[CRC+18] D. Cavalcanti, M. Rashid, L. Cariou, G. Venkatesan, R. Stacey. *Controling latency in 802.11*. Dokumentennr. 802.11-18/1160r0, IEEE. Juli 2018 (zitiert auf S. 30, 31).

[CRV+18] D. Cavalcanti, M. Rashid, G. Venkatesan, L. Cariou, R. Stacey. *Time-Aware shaping (802.1Qvb) support in the 802.11 MAC*. Dokumentennr. 802.11-18/1892r0, IEEE. Nov. 2018 (zitiert auf S. 30, 50).

[Esp23a] Espressif. *ESP-IDF API Reference - Networking APIs - Wi-Fi*. Accessed on 13 July 2023. 2023. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html (zitiert auf S. 58, 76).

[Esp23b] Espressif. *ESP32 Modules*. 2023. URL: https://www.espressif.com/en/products/modules (zitiert auf S. 27).

[Esp23c] Espressif. *Modules | Espressif Systems*. 2023. URL: https://www.espressif.com/en/products/modules (zitiert auf S. 27).

[Far19] J. Farkas. *IEEE 802.1 TSN - An Introduction*. Dokumentennr. 802.11-19/1298r1, IEEE. Juli 2019.

[GW07] G. Gridling, B. Weiss. „Introduction to microcontrollers“. In: *Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group* (2007) (zitiert auf S. 26).

[HJA+21] J. Haxhibeqiri, X. Jiao, M. Aslam, I. Moerman, J. Hoebeke. „Enabling TSN over IEEE 802.11: Low-overhead time synchronization for wi-fi clients“. In: *2021 22nd IEEE international conference on industrial technology (ICIT)*. Bd. 1. IEEE. 2021, S. 1068–1073 (zitiert auf S. 40).

[IEE16] IEEE. „IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications“. In: *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)* (2016). DOI: 10.1109/IEEESTD.2016.7786995 (zitiert auf S. 18–23, 78).

[IEE22] IEEE. „IEEE Standard for Ethernet“. In: *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)* (2022), S. 1–7025. DOI: 10.1109/IEEESTD.2022.9844436 (zitiert auf S. 18).

[IJ18] K. Iyer, A. Jones. *Real-time Console Game Network Profile*. Dokumentennr. 802.11-18/1499r0, IEEE. Sep. 2018 (zitiert auf S. 31).

[KIS+13] A. Kishida, M. Iwabuchi, T. Shintaku, T. Sakata, T. Hiraguri, K. Nishimori. „User-oriented QoS control method based on CSMA/CA for IEEE802. 11 wireless LAN system“. In: *IEICE transactions on communications* 96.2 (2013), S. 419–429 (zitiert auf S. 19, 22).

[KLA20] E. Khorov, I. Levitsky, I. F. Akyildiz. „Current Status and Directions of IEEE 802.11be, the Future Wi-Fi 7“. In: *IEEE Access* 8 (2020), S. 88664–88688. DOI: 10.1109/ACCESS.2020.2993448 (zitiert auf S. 17–19, 29, 30).

[KLG+21] Y. Kang, S. Lee, S. Gwak, T. Kim, D. An. „Time-sensitive networking technologies for industrial automation in wireless communication systems“. In: *Energies* 14.15 (2021), S. 4497 (zitiert auf S. 24, 25, 29, 30).

[KRD+18] S. Kim, M. M. Rashid, S. Deo, J. Perez-Ramirez, M. Galeev, G. Venkatesan, S. Dey, W. Li, D. Cavalcanti. „Demo/poster abstract: Enabling time-critical applications over next-generation 802.11 networks“. In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2018, S. 1–2 (zitiert auf S. 30, 31).

[MGS+12] B. Malinowsky, J. Grønbæk, H.-P. Schwefel, A. Ceccarelli, A. Bondavalli, E. Nett. „Timed broadcast via off-the-shelf WLAN distributed coordination function for safety-critical systems“. In: *2012 Ninth European Dependable Computing Conference*. IEEE. 2012, S. 144–155 (zitiert auf S. 18, 30).

[Mil94] D. L. Mills. „Precision synchronization of computer network clocks“. In: *ACM SIGCOMM Computer Communication Review* 24.2 (1994), S. 28–43 (zitiert auf S. 39).

[MTM22] E. Mozaffariahrar, F. Theoleyre, M. Menth. „A Survey of Wi-Fi 6: Technologies, Advances, and Challenges“. In: *Future Internet* 14.10 (2022), S. 293 (zitiert auf S. 29).

[NXP23] NXP. *i.MX RT1060 Crossover MCU with ARM Cortex-M7 Core*. 2023. URL: https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/i-mx-rt-crossover-mcus/i-mx-rt1060-crossover-mcu-with-arm-cortex-m7-core-operating-up-to-600-mhz-with-1-mb-ram:i.MX-RT1060 (zitiert auf S. 27).

[PGB13] A. Patro, S. Govindan, S. Banerjee. „Observing home wireless experience through wifi aps“. In: *Proceedings of the 19th annual international conference on Mobile computing & networking*. 2013, S. 339–350 (zitiert auf S. 18).

[PJR23] PJRC. *Teensy 4.1 Development Board*. 2023. URL: https://www.pjrc.com/store/teensy41.html (zitiert auf S. 27).

[PZC+16] C. Pei, Y. Zhao, G. Chen, R. Tang, Y. Meng, M. Ma, K. Ling, D. Pei. „WiFi can be the weakest link of round trip network latency in the wild“. In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, S. 1–9 (zitiert auf S. 31).

[RCAL12] A. L. Ruscelli, G. Cecchetti, A. Alifano, G. Lipari. „Enhancement of QoS support of HCCA schedulers using EDCA function in IEEE 802.11 e networks“. In: *Ad Hoc Networks* 10.2 (2012), S. 147–161 (zitiert auf S. 23).

[Rot05] J. Roth. *Mobile Computing: Grundlagen, Technik, Konzepte*. Dpunkt Lehrbuch. dpunkt-Verlag, 2005. ISBN: 9783898643665 (zitiert auf S. 17, 18).

[Sem21] N. Semiconductors. *i.MX RT1060 Processor Reference Manual*. Rev. 3. NXP Semiconductors. Juli 2021 (zitiert auf S. 27, 43, 54, 61–65, 67).

[SMBC21] S. Sudhakaran, V. Mageshkumar, A. Baxi, D. Cavalcanti. „Enabling QoS for collaborative robotics applications with wireless TSN“. In: *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE. 2021, S. 1–6 (zitiert auf S. 31).

[WAA+15] S. T. Watt, S. Achanta, H. Abubakari, E. Sagen, Z. Korkmaz, H. Ahmed. „Understanding and applying precision time protocol“. In: *2015 Saudi Arabia Smart Grid (SASG)*. IEEE. 2015, S. 1–7 (zitiert auf S. 40).

[WMW20] J. Wang, E. Meneses, M. Wanderley. „The scalability of WiFi for mobile embedded sensor interfaces“. In: *Proceedings of the Conference on New Interfaces for Musical Expression*. 2020, S. 73–76 (zitiert auf S. 31).

Alle URLs wurden zuletzt am 17. 08. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift