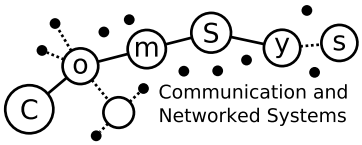
FACULTY OF
COMPUTER SCIENCE



Communication and Networked Systems

Masterarbeit

Implementierung und Analyse der Taktilen Koordinierungsfunktion im MIoT-Testbed

Johannes Behrens

Betreuer: Prof. Dr. rer. nat. Mesut Güneş
Betreuender Assistent: Frank Engelhardt, M.Sc.

Institut für Intelligente Kooperierende Systeme, Otto-von-Guericke-Universität Magdeburg

15. Dezember 2021

Aufgrund der besseren Lesbarkeit wird im Text hauptsächlich das generische Maskulinum verwendet. Gemeint sind jedoch immer alle Geschlechter.

Zusammenfassung

Kurzfassug

Ziel der Arbeit ist es, die *Taktile Koordinierungsfunktion* von Engelhardt et al. [1] in der realen Welt zu implementieren. Insbesondere werden die Probleme bei einer Integration in ein *Linux*-Betriebssystem angesprochen. Mit dieser Implementierung werden Experimente durchgeführt, um die Eignung dieser Funktion für das *Taktile Internet* zu bewerten. Zu diesem Zweck wird ein modularer Testaufbau innerhalb des *MIoT-Testbed* erstellt, der eine einfache Anpassung der Experimente ermöglicht.

In den Experimenten wird gezeigt, dass die *Taktile Koordinierungsfunktion* niedrigere Latenzen bei hoher Funkkanalauslastung ermöglichen kann. Gleichzeitig profitiert das *Taktile Internet* von einem geringeren Jitter. Zusammenfassend lässt sich sagen, dass die *Taktile Koordinierungsfunktion* einen positiven Einfluss auf die Eignung von *IEEE 802.11* für das *Taktile Internet* hat. Allerdings bleiben die Latenzen stark abhängig von der Auslastung und äußeren Einflüssen auf den Funkkanal.

Abstract

The goal of the work is to implement the *Tactile Coordination Function* of Engelhardt et al. [1] in the real world. In particular, the problems with an integration into a *Linux* operating system will be addressed. Experiments will be conducted with this implementation to evaluate the suitability of this feature for the *Tactile Internet*. For this purpose, a modular test setup is created within the *MIoT-Testbed*, which allows easy customization of the experiments.

In the experiments, it is shown that the *Tactile Coordination Function* can enable lower latencies under high wireless channel utilization. At the same time, the *Tactile Internet* benefits from lower jitter. In conclusion, the *Tactile Coordination Function* has a positive impact on the suitability of *IEEE 802.11* for the *Tactile Internet*. However, the latencies remain highly dependent on the load and external influences on the radio channel.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Akronyme

AIFS Arbitration Interframe Space. 12–14, 20, 35, 45, 46
ARP Address Resolution Protocol. 30

CAF Channel Access Function. 11–13, 19
CSMA/CA Carrier Sense Multiple Access with Collision-Avoidance. 11
CW Contention Window. 12–14
CWmax maximales Contention Window. 12–14
CWmin minimales Contention Window. 12, 14

DCF Distributed Coordination Function. 11
DIFS Distributed Interframe Space. 12
DMESG display message. 22
DPDK Data Plane Development Kit. 8

EDCA Enhanced Distributed Channel Access. 4, 9, 11, 12, 16, 19, 21, 23
EIFS Extended Interframe Space. 12

IBSS Independent Base Service Set. 18
IEEE Institute of Electrical and Electronics Engineers. 4
IFS Interframe Space. 11
IoT Internet of Things. 3, 4

MCCA Mesh Coordinated Channel Access. 19
MCF Mesh Coordination Function. 19
MIoT Magdeburg Internet of Things. 15, 16, 18

NIC Network Interface Card. 28
NTP Network Time Protocol. 28

PAT Personal Access Token. 29

PTP Precision Time Protocol. 25, 27, 29, 36

QoS Quality of Service. 8, 11, 13, 15

TBMS Testbed Management System. 15, 16, 26, 27, 29, 43

TCP Transmission Control Protocol. 25

TKF Taktile Koordinierungsfunktion. 9, 11, 13, 14, 36

TOS Type of Service. 25, 27

TX Transmit. 4, 20, 23

TXOP Transmission Opportunity. 9, 12–14, 20, 46

UDP User Datagram Protocol. 25

uRLLC Ultra-Reliable Low-Latency Communications. 7

VR Virtual Reality. 3, 4

WNIC wireless network interface controller. 16–22, 30, 40

"The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform. It can follow analysis; but it has no power of anticipating any analytical relations or truths. Its province is to assist us in making available what we are already acquainted with."

- Ada Lovelace, 1843

KAPITEL 1

Einleitung

Das *Taktile Internet* wird als die nächste große Revolution des *Internet of Things (IoT)* bezeichnet [2]. Die International Telecommunication Union (ITU) definiert den Begriff *Taktiles Internet* im August 2014. Laut dieser Definition bietet das *Taktile Internet* eine extrem niedrige Latenz in Verbindung mit einer hohen Verfügbarkeit, Zuverlässigkeit und Sicherheit [3]. Zu den Anwendungen gehören Fälle, bei denen Menschen mit Menschen oder Menschen mit Maschinen in einer entfernten Umgebung kommunizieren. Neben den üblichen Daten wie Audio und Video werden auch Daten übertragen, die sich auf den Tastsinn beziehen. Dies geschieht mit dem Ziel, dem Benutzer das Gefühl zu geben, in die entfernte Umgebung einzutauchen. Beispielsweise sind hier die *Immersive Virtual Reality (VR)* und die Steuerung von Robotern aus der Ferne (*Teleoperation*, siehe Abbildung 1.1) zu nennen.

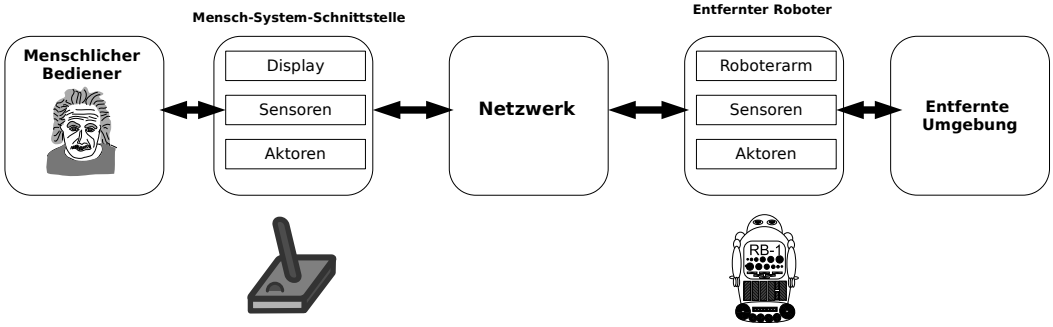


Abbildung 1.1: Teleoperation System mit bidirektionaler taktiler Kommunikation. (Eigene Abbildung in Anlehnung an [4])

Die für diese Anwendungen übertragenen haptischen Daten sind durch geringe Paketgrößen von 2-8 Byte pro Freiheitsgrad [5] gekennzeichnet. Benötigt wird aber eine sehr hohe Datenrate von bis zu 1000 Paketen pro Sekunde [4], um Signale für den Tastsinn zu übertragen. Gleichzeitig gibt es die Anforderungen an eine geringe Latenz, welche das Eintauchen in die entfernte Umgebung ermöglicht (< 1 ms) [6].

Für viele der Anwendungen ist es wünschenswert, dass Geräte drahtlos angebunden sind. Kabel sind beispielsweise bei Geräten, wie Datenhandschuhen oder einer VR-Brille störend,

da sie den Bewegungsradius einschränken. Aufgrund der weiten Verbreitung und der großen Kompatibilität bezüglich vorhanderer Hardware bietet sich hier der Standard *Institute of Electrical and Electronics Engineers (IEEE) 802.11*, allgemein bekannt als *Wi-Fi*, an. Problematisch bei einer Übertragung über *IEEE 802.11* sind allerdings die Latenzen, die nicht vorhersagbar und für die taktile Kommunikation in der Regel zu hoch sind.

An dieser Stelle wird die Koordinierungsfunktion *EDCA* als Erweiterung von *IEEE 802.11* interessant. Bei *EDCA* handelt es sich um eine Koordinierungsfunktion für den Kanalzugriff innerhalb des Funknetzwerks. Über *EDCA* werden vier verschiedene TX-Warteschlangen mit unterschiedlich priorisierten Verkehrsklassen (Voice, Video, Best Effort, Background) verwaltet. In der aktuellen Implementierung besteht jedoch noch keine dedizierte Klasse für die taktile Kommunikation. Die Anforderungen an das Netzwerk für die taktile Kommunikation liegen dabei über den Anforderungen für die Sprachkommunikation.

1.1 Motivation und Zielsetzung

Das *Taktile Internet* ist als Neuerung des *IoT* eine aufstrebende Technologie. Zu den bestehenden Kommunikationsmodalitäten wie Ton und Video wird es ermöglicht, dass Berührungen übertragen werden können. Dadurch müssen die Nutzenden sich nicht mehr direkt an den Orten befinden, wo sie ihre Tätigkeiten ausführen. Das ermöglicht beispielsweise, dass Operationen von spezialisierten Operierenden ausgeführt werden, ohne, dass die Operierten vorher an einen anderen Ort transportiert werden müssen. Das größte Hindernis für die Erreichung dieses Zieles ist die zu hohe Latenz. Diese muss unter 1 ms liegen, damit die Nutzenden den Eindruck haben, dass sie sich vor Ort befinden [7]. Viele Ansätze zielen darauf ab, neue Kommunikationstechnologien anstelle der bestehenden Infrastruktur und Technologie zu nutzen. Dadurch entstehen hohe Kosten und die neuen Technologien lassen sich gegebenenfalls nicht ohne Probleme zu den bestehenden Infrastrukturen ergänzen.

In einer Arbeit von Engelhardt et al. [1] aus dem Jahr 2019 wird dieses Problem angegangen. Hier wird Untersucht, ob eine Anpassung des *EDCA* unter Schaffung einer fünften Verkehrsklasse für das *Taktile Internet* zu einer geringeren Latenz für diese Pakete führt. Die sich daraus ergebende Zugriffsfunktion wird dort auf den Namen *Taktile Koordinierungsfunktion* getauft. Dabei konnte gezeigt werden, dass dieses Vorgehen in einer Simulation zu positiven Ergebnissen führt. Allerdings ist noch unklar, ob diese Ergebnisse in einem Testbed unter realen Bedingungen reproduziert werden können. Dies hat meine Neugierde für dieses Thema geweckt. Eine kurze Einführung in *EDCA* findet sich in Abschnitt 3.1.

Hinzu kommt, dass ich mich bereits in meiner Bachelorarbeit mit dem *Taktilen Internet* befasst habe. Das Thema dort war die Schaffung eines allgemeinen Testframeworks für die Steuerung von entfernten Applikationen (z.B. Roboter) über ein Netzwerk. Hier wurden haptische Daten in Form von vielen kleinen Datenpakete mit einer hohen Frequenz übertragen. In den damaligen Testaufbauten ist aufgefallen, dass über ein Funknetzwerk die dafür benötigten Latenzen (< 1 ms) regelmäßig nicht eingehalten werden konnten.

Das Ziel dieser Arbeit ist, zu untersuchen, wie die *Taktile Koordinierungsfunktion* in einem realen System umgesetzt werden kann. Dabei soll auf die Probleme bei der Umsetzung in einem realen System im Gegensatz zu einer Simulation eingegangen werden. Über ei-

nen geeigneten Testaufbau sollen damit die in Abschnitt 1.2 genannten Forschungsfragen beantwortet werden.

1.2 Forschungsfragen

Im Verlauf der Arbeit sollen die folgenden Forschungsfragen genauer untersucht und beantwortet werden:

1. Bestätigt sich das Ergebnis von Engelhardt et al. [1], dass die *Taktile Koordinierungsfunktion* als Ergänzung zu *IEEE 802.11* für das *Taktile Internet* geeignet ist, in einem Testbed?
2. Welche Auswirkungen hat die Schaffung einer erhöhten Priorität für das *Taktile Internet* für die weitere Kommunikation in dem Netzwerk?
3. Welche Hürden bestehen bei der Implementierung einer erhöhten Priorität für Datenpakete des *Taktilen Internets* im Rahmen von IEEE 802.11?

1.3 Aufbau der Arbeit

Die Arbeit ist entsprechend der gegebenen Vorlage der Arbeitsgruppe *ComSys*[1] an der Otto-von-Guericke-Universität Magdeburg gegliedert und aufgebaut.

Nach der Einleitung in das Thema folgt in dem nächsten Kapitel eine Einordnung des Themas in bestehende und verwandte Forschungsarbeiten. Dort werden weitere Ansätze vorgestellt und verglichen, welche das Ziel verfolgen, dass das *Taktile Internet* drahtlos genutzt werden kann. Für die Einordnung des Vorgehens innerhalb dieser Arbeit folgt in Kapitel 3 eine Einführung in den Medienzugriff innerhalb von *IEEE 802.11* und die umzusetzende *Taktile Koordinierungsfunktion*.

Kapitel 4 widmet sich der Frage, wie die *Taktile Koordinierungsfunktion* in der realen Welt implementiert werden kann. Dazu werden in dem Kapitel die Tools und die Plattform vorgestellt, welche für die Durchführung der Experimente in dem *MIoT-Testbed* benötigt werden. Das dient der Beantwortung der dritten Forschungsfrage.

Die Durchführung der Experimente mit der Taktilen Koordinierungsfunktion und der genaue Versuchsaufbau werden in Kapitel 5 ausgearbeitet. In Kapitel 6 folgt die Evaluierung der Ergebnisse. Damit können die Forschungsfragen 1 und 2 beantwortet werden. In Kapitel 7 folgt eine Zusammenfassung sowie das Fazit und ein Forschungsdesiderat.

1.4 Definitionen

In diesem Abschnitt werden wichtige Begriffe für die Masterarbeit definiert, die in der Literatur nicht immer eindeutig beschrieben sind.

[1] `https://comsys.ovgu.de/de/`, Abgerufen am 27.11.2021

1.4.1 Latenz

Wenn in dieser Arbeit von Latenzen gesprochen wird, so wird immer die Laufzeit eines Pakets vom Sender zum Empfänger gemeint (Ein-Weg-Latenz). Sollte jedoch die Paketumlaufzeit (Zwei-Wege-Latenz) gemeint sein, so wird dies extra erwähnt. Es wurde sich für die einfache Latenz entschieden, da hiermit asymmetrische Effekte verhindert werden sollen.

1.4.2 Jitter

Als Jitter wird die Varianz in den Latenzen der übertragenen Pakete bezeichnet. Der Jitter wird direkt aus dem genutzten Tool *iperf2* übernommen. Innerhalb des Tools wird sich an der RFC 1889 [8] des *Real Time Protocols (RTP)* orientiert. Dort wird der Jitter wie folgt kontinuierlich bei jedem empfangenen Datenpaket i berechnet:
$J = J + (|D(i-1,i)| - J)/16$
J steht für den Jitter und wird zu Beginn mit 0 initialisiert. $D(i-1,i)$ steht für die Differenz zwischen der aktuellen Latenz und der vorherigen Latenz. Der Verstärkungsfaktor $1/16$ sorgt für eine Rauschreduzierung und stellt dabei sicher, dass die notwendige Konvergenzrate erreicht wird [9].

KAPITEL 2

Related Work

Das *Taktile Internet* gewinnt in der aktuellen Zeit an Bedeutung. Entsprechend gibt es zahlreiche Forschungsbemühungen, um die erforderliche *1 ms Challenge* zu bewältigen. Bereits in der Vergangenheit wurden verschiedene Ansätze verfolgt, um die Latenz für drahtlose Anwendungen zu senken. Einige dieser Forschungsansätze werden in diesem Kapitel vorgestellt. Auch wird die Arbeit von Engelhardt et al. [1], welche die Grundlage für diese Masterarbeit bietet, tiefergreifend vorgestellt. Der Großteil der vorgestellten Arbeiten unterscheidet sich von der hier anvisierten Lösung darin, dass die dort vorgestellten Technologien nicht in die bestehende *IEEE 802.11* Infrastruktur integriert werden können.

2.1 5G

Während der Entwicklung von *5G* standen bereits die Anforderungen an das *Taktile Internet* fest, weshalb diese dort berücksichtigt werden konnten. Daher müssen keine Kompromisse eingegangen werden, so wie es bei Ergänzung bestehender Technologien der Fall ist. So bietet *5G* direkt eine neue *Ultra-Reliable Low-Latency Communications (uRLLC)* Service-Kategorie. Diese Kategorie ist für Anwendungen mit besonders hohen Anforderungen an die Latenz und Zuverlässigkeit gedacht.

Laut Slalmi et al. [10] ist *5G* die ideale Technologie für die Steuerung von Robotern innerhalb von Fabriken mit Latenzen deutlich unter 1 ms. Erreicht wird dies dadurch, dass die Sub-Frames (jeweils 1 ms) im Gegensatz zu *4G* feiner unterteilt werden können. Dadurch ergeben sich mehr Übertragungsmöglichkeiten für eine niedrigere Latenz und eine höhere Zuverlässigkeit. *5G* ermöglicht damit flüssige, schnelle und perfekt synchronisierte Bewegungen der Roboter, um komplexe Aufgaben zu lösen. *IEEE 802.11* sieht Slalmi als nicht geeignet an, da die Übertragungen zu anfällig bezüglich Störungen innerhalb von Fabriken sind.

5G klingt in der Theorie nach einer guten Wahl für das *Taktile Internet*. Jedoch stehen dieser Wahl hohe Investitionskosten für neue Hardware gegenüber. Bei einer Erweiterung des *IEEE 802.11* Standards hingegen kann optimalerweise die bestehende Infrastruktur weiterhin genutzt werden.

2.2 IEEE 802.11be (WiFi 7)

Das aktuell in der Entwicklung stehende *WiFi 7* soll im Gegensatz zu den bestehenden *WiFi*-Standards einen großen Fokus auf zeitsensitive Anwendungen legen. Dies beinhaltet, dass eine sehr niedrige Latenz und gleichzeitig eine hohe Zuverlässigkeit gewährleistet werden. Damit würden auch die Anforderungen an das *Taktile Internet* erfüllt werden. Von Adame et al. [11] werden die aktuellen Probleme von *EDCA* aufgezeigt und Möglichkeiten für einen effizienteren Medienzugriff für *WiFi 7* vorgestellt. Wie genau die Eignung für zeitsensitive Anwendungen erreicht werden soll, steht allerdings noch nicht fest. Die Autoren machen klar, dass auf einem lizenz-freien und geteilten Medium Echtzeit-Anforderungen nie zu 100 % erfüllt werden können. Allerdings gibt es laut Adame et al. Spielraum innerhalb des *IEEE 802.11* Standards, um geringe Latenzen zu ermöglichen. Die Schwächen von *EDCA* liegen demnach in zu wenigen Zugriffskategorien, einer fehlenden Priorisierung innerhalb einer Zugriffskategorie und je nach Hardware nur einem Buffer, um Pakete aller Kategorien zu speichern. Es besteht laut den Autoren aktuell kein Mechanismus, welcher einen *Quality of Service (QoS)* in einem stark ausgelasteten *WiFi*-Netz garantieren kann.

Besonders hervorzuheben an *WiFi 7* ist die zu erwartende Abwärtskompatibilität zu älteren *IEEE 802.11 Standards*. Damit können diese Anwendungen in die bestehende Infrastuktur integriert werden.

2.3 IEEE 802.11 User-Space Architektur

Backhaus et al. [12] haben eine User-Space Implementierung von Treibern und dem Linux Wi-Fi Stack auf Basis des *Data Plane Development Kit (DPDK)* geschaffen. In Experimenten konnten sie zeigen, dass die Verarbeitung der IEEE 802.11 Frames innerhalb des User-Spaces den Durchsatz erhöht und die Latenz verringert. Bei kleinen Datenpaketen wurde ein bis zu 100 % größerer Durchsatz und eine um bis zu 27 % geringere Latenz erreicht. Gleichzeitig ermöglicht der Ansatz im Vergleich zu der Kernelimplementierung flexible und einfache Experimente mit neuen Protokollen und Anpassungen der bestehenden Standards.

Die hier erreichte Verringerung der Latenz ist nicht die Folge einer Priorisierung des *Taktilen Internets*, sondern verringert die Latenz dadurch, dass der Overhead durch Interrupts und Kernelaufrufe wegfällt. Daher ist das gewählte Vorgehen nicht direkt mit dem Ansatz dieser Masterarbeit vergleichbar. Prinzipiell könnte dieser Ansatz aber mit einer zusätzlichen Priorisierung des *Taktilen Internets* zu einer weiteren Verbesserung der Ergebnisse führen. Vorteilhaft ist ebenfalls, dass der Ansatz kompatibel zu der bestehenden *IEEE 802.11* Infrastuktur ist.

2.4 Simulation eines Netzwerks mit der taktilen Koordinierungsfunktion

Von besonders großem Interesse für diese Masterarbeit ist die Veröffentlichung von Engelhardt et al. [1]. Die Autoren erschaffen und evaluieren dort die *Taktile Koordinierungsfunktion*, welche innerhalb dieser Masterarbeit in der realen Welt umgesetzt wird, anhand einer Simulation mit der Software *OMNeT++*.

Konkret wird eine fünfte Warteschlange für haptische Daten zu dem *EDCA* hinzugefügt. Die Warteschlange ist so parametriesiert, dass sie eine besonders hohe Priorität besitzt, um die geringen Latenzen zu gewährleisten. Dadurch konnte eine 1 ms Ende-zu-Ende Latenz über ein drei-hop Netzwerk erreicht werden. Gleichzeitig konnte gezeigt werden, dass die Latenz und der Jitter nicht linear zu der Anzahl an Hops wächst. Auch waren diese beiden Metriken in der Simulation unabhängig vom Hintergrundverkehr im Netzwerk.

In einigen grundlegenden Punkten weist diese Masterarbeit Parallelen zu der Veröffentlichung von Engelhardt et al. auf. So wird die *TKF* und damit die Parametriesierung der Zugriffskategorie für die haptischen Daten übernommen. Im Gegensatz zu Engelhardt et al. werden die Experimente im Rahmen dieser Masterarbeit jedoch unter realen Bedingungen auf einem Testbed durchgeführt. Dies bringt einige zusätzliche Herausforderungen und Einschränkungen mit sich, welche im Folgenden genannt werden:

- Erstellen eines 802.11 Multi-Hop-Netzwerks inklusive Routing
- Anpassung des Kernels, um Backoff-Parameter anzupassen
- Software zum Setzen der Backoff-Parameter
- Erstellen und Analysieren der Datenströme
- Synchronisierung der Uhren zwischen den Testbed-Knoten
- Keine Anpassung der Warteschlangen-Größe möglich

Eine weitere große Abweichung im Aufbau ergibt sich in der Wahl der für die Experimente übertragenen Daten. Hier wurde für die Masterarbeit besonders großer Wert darauf gelegt, dass die Datenströme den Anforderungen an das *Taktile Internet* entsprechen. Auch wurde die Zuweisung der Daten zu den Warteschlangen anders gewählt. In der Arbeit von Engelhardt et al. werden die zusätzlich zu den haptischen Daten übertragenen Daten in die Warteschlange für Hintergrunddaten abgelegt. In der Praxis wird diese Warteschlange kaum genutzt und der Großteil der Daten landet in der Warteschlange *Best Effort* [13]. Das genaue Vorgehen wird in dem Kapitel 5 beschrieben.

2.5 Simulationsanalyse des IEEE 802.11e EDCA Protokolls für industriell relevante Echtzeitkommunikation

Eine weitere Simulation des EDCA Mechanismus wurde durch Moraes et al. [14] durchgeführt, um zu prüfen, ob sich die *IEEE 802.11e* Erweiterung für eine Echtzeit-Kommunikation in einer industriellen Umgebung eignet. Für die Simulation wurden die Standard-Backoff-Parameter genutzt. Es wurde geprüft, wie sich die Zugriffskategorie für Sprachdaten verhält, wenn darüber kleine Pakete in periodischen Intervallen gesendet werden. Die Paketgröße entspricht dabei 45 bytes, welche mit einer Rate von 500 Paketen pro Sekunde versendet werden. Dies entspricht ungefähr den Daten, wie sie bei einer haptischen Kommunikation anfallen.

Bei der Simulation wird gezeigt, dass der *Transmission Opportunity (TXOP)*-Mechanismus den Durchsatz für Nachrichten-Streams erhöht, wenn kleine Pakete versendet werden. Dies allerdings unter einer Erhöhung des Jitters. Gleichzeitig wird gezeigt, wie stark der Einfluss von externen Störungen für die Echtzeit-Kommunikation ist. Bereits eine Erhöhung der

Netzwerkauslastung von 10 % auf 30 % erhöht die Latenz und den Paketverlust deutlich. Zusammenfassend sagen die Autoren, dass die Standardparameter des EDCA-Mechanismus nicht für die industrielle Kommunikation geeignet sind, wenn die Zugriffskategorie für Sprachdaten genutzt wird.

Allerdings handelt es sich auch hier nur um eine Simulation und nicht um Ergebnisse aus einem Experiment in einer realen Umgebung. In dieser Masterarbeit werden zusätzlich andere Backoff-Parameter genutzt, welche die *Taktile Koordinierungsfunktion* bilden.

KAPITEL 3

Grundlagen zu Medienzugriff in IEEE 802.11

Das Kapitel beginnt mit einer Einführung in den Kanalzugriff innerhalb des Standards *IEEE 802.11*. Anschließend werden die *QoS*-Erweiterung *EDCA* des Standards *IEEE 802.11e* und seine Erweiterung, die *TKF*, vorgestellt.

Der Kanalzugriff in *IEEE 802.11* folgt dem *Carrier Sense Multiple Access with Collision-Avoidance (CSMA/CA)*-Verfahren. Das bedeutet, dass ein Knoten, der ein Paket übertragen möchte, zunächst wartet, bis er den Kanal für eine bestimmte Zeit als frei erkennt. Diese Zeit wird als *Interframe Space (IFS)* bezeichnet. Sobald der Kanal für *IFS* als frei erkannt wird, wird eine zufällige Backoff-Zeit festgelegt. Der Kanal muss anschließend für diese Zeit frei bleiben, bevor der Knoten sein Paket senden darf. Die Backoff-Zeit wird nach jeder verstrichenen Slot-Zeit um einen Zähler verringert, wenn der Kanal während dieser Zeit frei ist. Wird jedoch festgestellt, dass der Kanal belegt ist, wird der Zähler eingefroren und es muss erneut für *IFS* gewartet werden. Der Empfänger sendet eine Bestätigung, wenn das Paket fehlerfrei übertragen wurde [15].

Im Laufe der Zeit gab es zahlreiche Erweiterungen zu dem *IEEE 802.11* Standard, welche den Kanalzugriff zum Beispiel um *QoS* erweitert haben. Sofern diese Erweiterungen relevant sind, werden sie im Laufe der Arbeit an den entsprechenden Stellen vorgestellt.

3.1 Enhanced Distributed Channel Access (EDCA)

Bei dem *EDCA* handelt es sich um ein QoS-Zugriffsverfahren aus der *IEEE 802.11e* Erweiterung. *EDCA* ist eine erweiterte Version der weit verbreiteten *Distributed Coordination Function (DCF)*. Das bedeutet, dass der *EDCA* ebenfalls auf dem *CSMA/CA*-Protokoll basiert [17].

EDCA organisiert den Zugriff auf einen Funkkanal auf Basis von *Channel Access Functions (CAFs)*. Eine Station kann gleichzeitig mehrere *CAFs* besitzen. Laut dem 802.11e Standard sind dies maximal vier *CAFs*. Alle erstellten Pakete einer Station werden in einer zu der *CAF* gehörenden Warteschlange einsortiert. Auf diese Warteschlange wendet jede *CAF* einen eigenen Backoff-Vorgang an. Dieser wird über vier konfigurierbare Parameter gesteu-

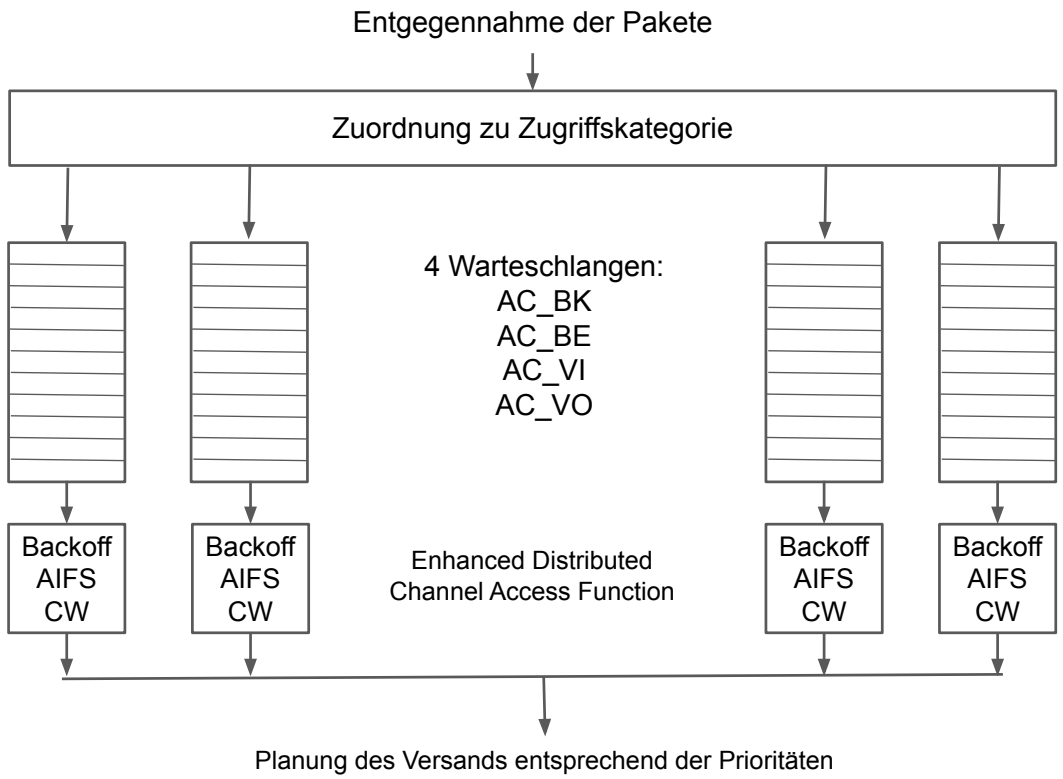


Abbildung 3.1: Verarbeitung eines Datenpakets mit dem EDCA. (Eigene Abbildung in Anlehnung an [16])

ert: *minimales Contention Window (CWmin)*, *maximales Contention Window (CWmax)*, *Arbitration Interframe Space (AIFS)* und *TXOP*.

Der Backoff-Vorgang läuft so ab, dass eine *CAF* mit einem zu übertragenden Paket den Kanal abhört. Sobald der Kanal für eine bestimmte Zeitspanne (AIFS) frei ist, wird das Paket übertragen. Sollte der Kanal während der *AIFS*-Zeitspanne als belegt erkannt werden, so wird weiter auf den Kanal gehört, bis er für eine Komplette *AIFS*-Zeitspanne als frei erkannt wird. Hier beginnt der eigentliche Backoff-Prozess. Die *CAF* initialisiert ihren Backoff-Zeit-Zähler auf einen zufälligen Wert aus dem gleichmäßig verteilten Bereich (0,Contention Window (CW)−1). *CW* ist hier das Contention Window der *CAF* und abhängig von der Anzahl an zuvor fehlgeschlagenen Übertragungen. Beim ersten Übertragungsversuch wird das *CW* auf *CWmin* gesetzt.

So lange, wie der Kanal als frei erkannt wird, wird nach jeder abgelaufenen Slot-Zeit der Backoff-Zeit-Zähler einmalig verringert. Die Slot-Zeit wird durch den genutzten 802.11 Standard vorgegeben. In dem Moment, in dem eine Übertragung auf dem Kanal erkannt wird, wird der Zähler gestoppt. Er wird wieder aktiviert, wenn der Kanal für eine bestimmte Zeit als frei detektiert wird (die Zeit entspricht der *AIFS*-Zeit, wenn die Übertragung mit einer korrekten *CRC*-Prüfsumme empfangen wird, ansonsten *Extended Interframe Space (EIFS)−Distributed Interframe Space (DIFS)+AIFS*). Wenn der Backoff-Zeit-Zähler bei Null ankommt, wird das Paket durch die *CAF* zu der nächsten Slot-Zeit übertragen.

Bei dem Fall, dass zwei Stationen zeitgleich eine Übertragung starten, kommt es zu einer Kollision. Tritt der Fall auf, dass eine Station mehrere *CAF* betreibt und diese zum gleichen Zeitpunkt eine Übertragung starten möchten, wird dies intern durch einen Scheduler geregelt. Die Warteschlange mit der höchsten Priorität darf ihre Übertragung als erstes starten. Eine Übertragung ist dann erfolgreich, wenn ein *ACK*-Paket innerhalb einer festgelegten Zeitperiode (ACK timeout) bei der übertragenden *CAF* ankommt. Kommt dieses Paket nicht an, so geht die *CAF* davon aus, dass das Paket nicht erfolgreich empfangen wurde und initiiert eine erneute Übertragung. Dafür wird der Backoff-Vorgang dort fortgesetzt, wo er unterbrochen wurde.

Nach jeder gescheiterten Übertragung wird das *CW* verdoppelt, bis es seinen maximalen Wert *CWmax* erreicht hat. Solche fehlerhaften Übertragungen können beispielsweise durch Kollisionen oder durch Rauschen und Störungen des Funkkanals entstehen. Ist eine Übertragung aber erfolgreich, so besteht die Möglichkeit, dass die *CAF* direkt weitere Pakete überträgt. Die Zeit dafür wird durch das *TXOP* begrenzt [18].

3.2 Taktile Koordinierungsfunktion

Die von Engelhardt et al. [1] vorgestellte *TKF* fügt der QoS-Implementierung aus *IEEE 802.11e* eine fünfte Zugriffskategorie für das *Taktile Internet* hinzu. Für diese Zugriffskategorie wird eine neue Warteschlange geschaffen, welche durch eine eigene *CAF* verwaltet wird. Die Backoff-Parameter für diese Warteschlange sind Tabelle 3.1 zu entnehmen.

Zugriffskategorie	TXOP (* 32 µs)	CWmin (* st)	CWmax (*st)	AIFS (*st)
Tactile Internet (AC_TC)	**31**	**3**	**7**	**1**
Voice (AC_VO)	47	3	7	2
Video (AC_VI)	94	7	15	2
Best Effort (AC_BE)	0	15	1023	3
Background (AC_BK)	0	15	1023	7

Tabelle 3.1: Backoff-Parameter der TKF nach Engelhardt et al. [1]. Die Einheit für *CWmin*, *CWmax* und *AIFS* ist das Vielfache der Slotzeit (st) für den genutzten 802.11 Standard.

Betrachtet werden die Parameter der Zugriffskategorie *AC_TC* im Vergleich zu der bisher am höchsten priorisierten Kategorie *AC_VO*. Verändert hat sich unter anderem die *TXOP*. Die *TXOP* ist von 1504 µs auf 992 µs gesunken. Durch die *TXOP* wird der Zeitrahmen vorgegeben, welcher für die Übertragung von Paketen nach dem Ablauf der Backoff-Zeit zur Verfügung steht. Theoretisch bedeutet dies, dass eine hohe *TXOP* eine höhere Priorität widerspiegelt. Allerdings ist zu beachten, dass es viele Anwendungsfälle innerhalb des *Taktilen Internets* gibt, bei denen haptische Daten in beide Richtungen zwischen zwei Knoten übertragen werden. Gleichzeitig zeichnen sich die Daten dadurch aus, dass sie sehr klein sind und daher nicht fragmentiert werden müssen. Damit nicht einer der beiden Knoten bei einer bidirektionalen Übertragung den anderen Knoten zu lange blockiert, wird die *TXOP* also auf ungefähr 1 ms reduziert. So kann für beide Knoten die Wartezeit innerhalb der Warteschlange reduziert werden.

Weiterhin ist bei der *TKF* für die Zugriffskategorie *AC_TC* im Gegensatz zur Kategorie für Sprachdaten die *AIFS* von zwei Slotzeiten auf eine Slotzeit verringert worden. Dadurch wird der Backoff-Prozess verkürzt, da die Wartezeit vor dem Beginn und der Wiederaufnahme des Backoff-Vorgangs kürzer ist. Die Parameter für die *Contention Windows* bleiben unverändert. Dies wird damit begründet, dass Kollisionen zwischen den gleichen Prioritäten vermieden werden sollen. Bei einem kleinen Intervall zwischen *CWmin* und *CWmax* gibt es nur wenige mögliche *CW*, welche zufällig gewählt werden können. Dadurch steigt die Wahrscheinlichkeit für Kollisionen.

Zusammenfassend kann festgestellt werden, dass der größte Effekt für niedrigere Latenzen durch die Verringerung der *AIFS* zu erwarten ist. Die Verringerung der *TXOP* sorgt hingegen für eine gerechtere Verteilung des Mediums. Insgesamt steigt die Wahrscheinlichkeit, dass die Latenzen geringer sind.

KAPITEL 4

Implementierung der Taktilen Koordinierungsfunktion und Integration der Experimente in das MIoT-Testbed

Das Kapitel gibt Details zur Implementierung und Konfiguration der für die Masterarbeit benötigten Software, Tools und Lösungen. Dabei erfolgt eine Orientierung an den Problemen, welche sich im Vergleich zu der bestehenden Simulation ergeben. Kurze Scripte werden teilweise direkt in den Abschnitten gezeigt. Als Testbed-Action gekennzeichnete Befehle werden durch das Testbed Management System (TBMS) des *MIoT-Testbed* ausgeführt, welches zu Beginn dieses Kapitels vorgestellt wird. Weiterer Quellcode findet sich im Anhang der Arbeit und im zum Masterarbeit gehörenden Git-Repository[1]. Zur Vereinfachung und Übersichtlichkeit werden triviale Befehle, wie beispielsweise das Erstellen von Ordnern nicht dargestellt.

4.1 Auswahl der genutzten Plattform (Hardware, Betriebssystem, Testumgebung)

Für die Durchführung eines Netzwerkexperiments in der realen Welt muss in einem ersten Schritt ein Betriebssystem und die Hardware ausgewählt werden. Diese Auswahl ist für die folgenden Schritte entscheidend, da die Implementierung der Software davon abhängig ist.

Da das *MIoT-Testbed* zur Durchführung der Experimente genutzt werden soll, fällt diese Entscheidung allerdings leicht. Auf den Testbed-Knoten ist ein *Debian*-Linux installiert und auch die Hardware ist durch das Testbed vorgegeben und kann der Tabelle 4.1 entnommen werden. Durch die Nutzung des Linux-Kernels ergeben sich automatisch einige Vorteile. So enthält der Kernel bereits eine vollständige Implementierung für *IEEE 802.11* inklusive der QoS-Unterstützung und auch *IEEE 802.11s (Mesh)* ist innerhalb des Kernels implementiert. Gleichzeitig ist der Linux-Kernel weit verbreitet und wird aktiv gepflegt.

[1] `https://code.ovgu.de/comsys-group/students/msc/2021-behrens-johannes`, Abgerufen am 07.09.2021

PC Engines APU.3C4 Board	
CPU	1,0 GHz AMD Embedded G-Series GX-412TC (Quad-Core, 64 Bit
RAM	4 GByte DDR3-1333 DRAM
wireless network interface controller (WNIC)	Qualcomm Atheros QCA986x/988x 802.11ac
OS	Debian GNU/Linux 10 (buster)
Kernel	#1 SMP Debian 4.19.194-1 (2021-06-10)

Tabelle 4.1: Informationen über Testbed-Knoten des *MIoT-Testbeds*.

Das MIoT-Testbed der Otto-von-Guericke Universität Magdeburg zielt hauptsächlich auf den Vergleich zwischen Simulationen und Experimenten in der Umgebung der realen Welt. Aktuelle Veröffentlichungen haben gezeigt, dass nach wie vor Unterschiede in den Ergebnissen zwischen Simulationen und Experimenten in der echten Welt existieren [19]. Die meisten Netzwerksimulatoren abstrahieren die Eigenschaften des drahtlosen Mediums. Eigenschaften des genutzten Betriebssystem auf einer realen Hardware werden dahingegen vernachlässigt. Dazu gehören beispielsweise Scheduling-Strategien und Multi-Threading, welche einen Einfluss auf das Ergebnis haben. Nachteilig an einem solchen Testbed im Gegensatz zu einer Simulation ist die Implementierung des Netzwerk-Stacks, welcher tief im Betriebssystem verankert sind, was eine Anpassung der Algorithmen (in diesem Fall EDCA) erschwert [19]. Die in dieser Masterarbeit durchgeführten Experimente sollen das Testbed nutzen, um zu prüfen, ob die *Taktile Koordinierungsfuktion* in der realen Welt die gleichen Ergebnisse liefert, wie in den zuvor durchgeführten Simulationen. Die Einschränkungen durch und Anpassungen am Netzwerkstack werden im Verlauf dieses Kapitels genauer erklärt.

Das Testbed besteht aktuell aus zehn Knoten, welche im Fakultätsgebäude der Fakultät für Informatik an der Otto-von-Guericke Universität aufgestellt sind. Zukünftig sollen über 100 Knoten auf dem gesamten Universitätscampus verteilt sein [19]. Bei diesen Knoten handelt es sich um Single Board Computer des Herstellers PC Engines[2]. Diese Computer werden aufgrund der zahlreichen Netzwerkschnittstellen (bis zu drei Gigabit Ethernet Kanäle) und der vergleichsweise hohen Leistungsfähigkeit häufig als Router, Firewalls oder Server genutzt. Eine genauere Auflistung der Hardwarekonfiguration findet sich in Tabelle 4.1.

Die Knoten sind über Ethernet mit einem Testbed-Netzwerk untereinander und zu dem Testbed-Server verbunden. Auf diesem Server läuft unter anderem das TBMS und eine Weboberfläche zur Experimenterstellung. Das TBMS lädt die Software und Einstellungen, steuert die Experimente und sammelt die Ergebnisse ein [20]. Die Software des MIoT-Lab basiert auf der Software des DES-Testbeds [21] der Freien Universität Berlin und wird an der Universität Magdeburg weiterentwickelt [20].

Für die Erstellung der Experimente kann eine Weboberfläche genutzt werden. Dort können allgemeine Eigenschaften zu dem Experiment festgelegt werden. Dazu gehören zum Beispiel ein Name, eine Startzeit und eine Anzahl an Wiederholungen. Auch werden die für das Experiment auszuführenden Aktionen zu den verschiedenen Knoten(gruppen) zugewiesen. Jede Aktion entspricht einem Kommando für die Kommandozeile. Die so erstellten Experimente können in ein DES-Cript [22] exportiert werden. Dabei handelt es sich um ein

[2] `https://www.pcengines.ch/about.htm`, Abgerufen am 26.08.2021

Dateiformat, welches die Einstellungen und Aktionen für das Experiment beinhaltet und für ein späteres Experiment wieder geladen werden kann [20].

Durch die Nutzung des Testbeds mit fest positionierten Knoten, einheitlicher Hardware und einem relativ stabilen Umfeld [20] in Kombination mit der eindeutigen DES-Experiment-Beschreibung ergibt sich so ein hohes Maß an Reproduzierbarkeit.

4.2 802.11 Subsystem im Linux Kernel

Für die Umsetzung der Experimente wurde sich für die Nutzung des *802.11* Subsystems des Linux Kernels entschieden. In Abbildung 4.1 ist ein Überblick über die Implementierung innerhalb des Betriebssystems zu sehen. Dieser Abschnitt stellt Begrifflichkeiten und die Implementierung des Subsystems vor und geht auf die für die Arbeit wichtigen Bestandteile ein.

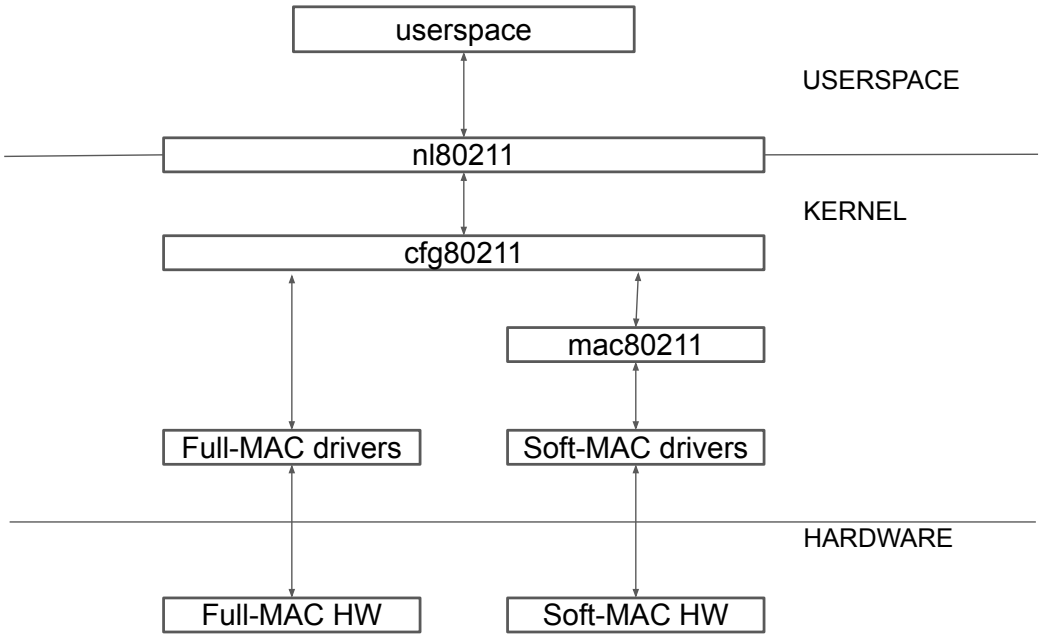


Abbildung 4.1: Überblick über die Linux WiFi-Architektur. (Eigene Abbildung in Anlehnung an [23] und [24])

4.2.1 Soft-Mac und Full-MAC

Bei *WNIC* kann grundsätzlich zwischen *Full-MAC*- und *Soft-MAC*-Hardware unterschieden werden. Bei *Soft-MAC*-Hardware erfolgt das Frame-Management auf der Seite der Software. Die Frames werden im Beispiel von Linux durch das *mac80211* Subsystem zerlegt und erzeugt. Auch erfolgt die Steuerung und Konfiguration der Hardware softwareseitig [25]. Heutzutage sind nur noch selten *WNIC* von dem Typ *Full-MAC* anzutreffen. Die meisten aktuellen *WNIC* sind aufgrund der einfacheren Treiber und Firmwareentwicklung

Soft-MAC [25]. Für die Experimente dieser Arbeit wird eine *Soft-MAC* Netzwerkkarte genutzt. Dies ermöglicht die Nutzung des *mac80211* Subsystems, moderner Netzwerkkarten und gleichzeitig auch die Nutzung des Testbeds der Universität Magdeburg.

4.2.2 mac80211

Einen großen Bestandteil der *802.11* Implementierung in Linux bildet das *mac80211* Subsystem. *mac80211* ist seit Juli 2007 (Kernelversion 2.6.22) ein Bestandteil des Kernel mainstream und befindet sich unter *net/mac80211* in dem Kernelverzeichnis. Das dazugehörige Modul nennt sich *mac80211.ko* [26]. *mac80211* bildet dabei für *Soft-MAC* Geräte eine Abstraktionsschicht zwischen der WLAN-Hardware und dem Betriebssystem [27]. Dies erleichtert die Treiberentwicklung und es müssen nur noch Teile der *802.11* Funktionalität in der Hardware beziehungsweise Firmware der Netzwerkkarte implementiert werden [25].

4.2.3 cfg80211 & nl80211

Zur Konfiguration der *WNIC* stellt Linux die API *cfg80211* bereit. Im Falle der *Soft-MAC WNIC* werden die Treiber auf *mac80211* ausgerichtet. *mac80211* implementiert wiederum die benötigten *cfg80211* callback-Funktionen. Daher ist *mac80211* für die Konfiguration und Registrierung der *WNIC* abhängig von *cfg80211* [25]. Treiber für *Full-MAC* Hardware kommunizieren hingegen direkt mit *cfg80211*. Zur Registrierung eines neuen Gerätes in Richtung der *cfg80211* API müssen die Geräte ihre callback-Funktionen und unterstützen Funktionalitäten mitteilen [28].

Zur Kommunikation zwischen dem Kernel und dem Userspace wird *nl80211* genutzt. Dabei handelt es sich um einen *netlink interface public header*. Mit der Hilfe von *nl80211* können so zum Beispiel Informationen über den verwendeten *WNIC* abgerufen werden, es können verfügbare Netzwerke angezeigt und auch die Konfiguration des *WNIC* vorgenommen werden [29].

4.3 802.11s Mesh Verbindung

Die nächste große Frage, die sich bei der Umsetzung in einem realen Netzwerk stellt ist, in welchem *IEEE 802.11* Modus die Knoten untereinander verbunden werden. Häufig genutzt ist der Infrastruktur-Modus. Das bedeutet, dass alle Knoten sich mit einem zentralen Punkt (Access-Point) innerhalb des Netzwerks verbinden. Dieser Access-Point vermittelt die Daten zwischen den einzelnen Geräten. Problematisch ist daran allerdings, dass sich dieser Modus nicht besonders gut skalieren lässt. Alle Knoten innerhalb eines Netzwerks müssten einen gemeinsamen Knoten besitzen, welcher in ihrer Funkreichweite ist. Daher ist dieser Modus für die Experimente innerhalb des *MIoT-Testbeds* ungeeignet.

Eine andere Möglichkeit wäre das Aufbauen eines Netzwerks im *IEEE 802.11 Independent Base Service Set (IBSS)* Modus. Dort können einzelne Testbed-Knoten direkt miteinander kommunizieren. Allerdings enthält der Modus keine Funktion zum Routing von Paketen für den Fall, dass zwei Knoten sich nicht direkt erreichen können. Für die Experimente

wurde sich daher für *IEEE 802.11s Mesh* Modus entschieden. Vorteil hier ist die einfache Skalierbarkeit und Umsetzbarkeit des Netzwerks innerhalb des Testbeds.

In dem *802.11s* Protokoll ist für den Zugriff auf das Medium die *Mesh Coordination Function (MCF)* vorgesehen. Diese basiert auf *EDCA* und *Mesh Coordinated Channel Access (MCCA)*. *MCCA* ist dabei optional und verhindert durch die Reservierung von Slotzeiten in der Zukunft zusätzlich Kollisionen. Für den eigentlichen Medienzugriff wird weiterhin *EDCA* genutzt [30]. In der open80211s [3] Implementierung, welche im Linux-Kernel genutzt wird ist *MCCA* noch nicht implementiert. Hier wird ausschließlich *EDCA* genutzt.

Das Netzwerk wird über das Script 4.1 aufgebaut, welches auf jedem der Testbed-Knoten ausgeführt wird.

```
1  #!/bin/bash
2  # brings wifi interface in mesh mode and sets ip
3  DEVICE=$1
4  IP=$2
5
6  sudo iw reg set DE
7  sudo iw dev $DEVICE interface add mesh0 type mp
8  sudo iw mesh0 set channel 44
9  sudo ifconfig mesh0 $IP
10 sudo iw dev mesh0 mesh join mesh
```

Quellcode 4.1: mesh.sh - Script zum Beitritt eines Knotens in das Mesh-Netzwerk.

Der Kanal 44 (5210–5230 MHz) wird für jedes Experiment fest gewählt. Es wurde sich für einen 20 MHz breiten Frequenzbereich entschieden, da hier mit weniger Kollisionen gerechnet werden kann. Da die Latenz innerhalb der Experimente eine größere Rolle als die Bandbreite spielt, war dieser Kompromiss einzugehen. Die zu nutzende WNIC und die IP-Adresse können über Parameter für jeden Knoten frei gewählt werden.

4.4 Schaffung einer dedizierten Zugriffskategorie für das *Taktile Internet*

Die *Taktile Koordinierungsfunktion* benötigt eine fünfte Warteschlange für taktile Daten innerhalb des *EDCA*, welche eine höhere Priorität als die weiteren Warteschlangen besitzt. Standardmäßig sieht *IEEE 802.11e* allerdings nur vier Warteschlangen vor. In einer Simulation ist es kein Problem diese Warteschlange umzusetzen. In einem realen System sieht dies anders aus.

Problematisch ist hier die Umsetzung in Richtung der *WNIC*. Die Warteschlangen sind nicht komplett innerhalb des *mac80211* Subsystems implementiert, sondern werden durch die *WNIC* umgesetzt. Je nach Umsetzung innerhalb des Treibers holen (pull) sich die *WNIC* die zu versendenden Pakete aus dem *mac80211* Subsystem, oder die Pakete werden in Richtung der Hardware gepusht. Dort werden sie innerhalb der Warteschlangen gehalten und durch die entsprechenden *CAFs* verwaltet.

[3] `https://github.com/o11s/open80211s/wiki/Status`, Abgerufen am 27.10.2021

ID	Zugriffskategorie	TXOP (* 32 µs)	CWmin (* st)	CWmax (*st)	AIFS (*st)
original					
0	AC_VO	47	3	7	2
1	AC_VI	94	7	15	2
2	AC_BE	0	15	1023	3
3	AC_BK	0	15	1023	7
adapted					
0	**AC_TC**	**31**	**3**	**7**	**1**
1	AC_VI	94	7	15	2
2	AC_BE	0	15	1023	3
3	**AC_VO**	**47**	**3**	**7**	**2**

Tabelle 4.2: Konfiguration der unterschiedlichen TX-Queues. Die ID entspricht der ID innerhalb des mac80211 Kernelmoduls von Linux. Die Einheit für cwmin, cwmax und aifs ist das Vielfache der Slotzeit (st) für den genutzten 802.11 Standard.

Eine ausführliche Recherche und Analyse der Treiber innerhalb des Linux-Kernels hat gezeigt, dass es keine aktuelle *WNIC* gibt, die ein Hinzufügen einer fünften Warteschlange unterstützt. Hier scheitert es daran, dass die Treiber zwar Open Source sind, aber Anpassungen an der Fimware oder sogar der Hardware notwendig wären. Die Firmware ist allerdings für den Großteil der *WNIC* nicht öffentlich verfügbar. Auch eine Nachfrage an die Linux-Wireless-Mailinglist hat diese Rechereche bestätigt. Der Maintainer (Johannes Martin Berg) des mac80211 Subsystems zweifelt an der Umsetzbarkeit einer fünften Warteschlange. Dafür empfiehlt er dort die Nutzung einer der wenig genutzten Warteschlangen (Voice, Background) [13].

Im Zuge der Masterarbeit werden die vorhandenen Warteschlangen daher umprogrammiert. Wichtig sind hier nicht nur die Parameter, sondern auch die Reihenfolge in der die Warteschlangen angeordnet sind. Sollte es zu der Konstellation kommen, dass auf mehr als einer Warteschlange der Backoff-Zeit-Zähler zur gleichen Zeit abgelaufen ist, so wird die Warteschlange mit der niedrigsten ID bevorzugt. Für die Experimente in denen die Zugriffskategorie *Taktiles Internet* benötigt wird, werden die Parameter auf die Warteschlange programmiert, welche ursprünglich für die Zugriffskategorie *Sprache* genutzt worden ist. Die Warteschlange für die Zugriffskategorie *Hintergrunddaten* wird zur Warteschlange für die Daten der Zugriffskategorie *Sprache*. Dies ist möglich, da die Zugriffskategorie *Sprache* nicht mit anderen Warteschlangen kombiniert wird. Deutlich wird dies in Kapitel 5. Ein kompletter Shift der Warteschlangen wurde vermieden, da standardmäßig Pakete über die Zugriffskategorie *Best Effort* versendet werden. Dies würde die Experimente verfälschen, wenn diese mit veränderten Parametern behandelt werden.

4.5 Festlegen der Backoff-Parameter für die Zugriffskategorien

Um die Backoff-Parameter unter der Nutzung einer *802.11s* Mesh-Verbindung anpassen zu können sind zwei Schritte erforderlich. Es muss der Linux-Kernel angepasst werden und es muss ein Tool programmiert werden, welches die Parameter aus dem Userspace über die *nl80211* Schnittstelle an den Kernel sendet. Dieser ruft die Konfigurationsfunktion des *WNIC*-Treibers auf.

4.5.1 Anpassung des Linux-Kernels

Der Linux-Kernel unterstützt die manuelle Anpassung der *EDCA*-Parameter nicht, wenn der *Mesh* Modus genutzt wird. Der Grund hierfür ist, dass alle Teilnehmer des *Mesh*-Netzwerks mit den gleichen *EDCA*-Parametern kommunizieren sollen. Diese werden daher während der Initialisierung auf die Standardwerte gesetzt. Für die Experimente ist es allerdings notwendig die Parameter anzupassen, um eine Warteschlange für die haptischen Daten zu schaffen. Damit dies trotzdem möglich ist wird das Kernelmodul *cfg80211* angepasst. Dies betrifft die Funktion *nl80211_set_wiphy(...)* innerhalb der Quellcode-Datei *net/wireless/nl80211.c*. Diese Funktion wird aufgerufen, wenn über das *nl80211* Interface die *WNIC* konfiguriert wird. Innerhalb dieser Funktion werden die Informationen aus einem *'wiphy'* auf die *WNIC* übertragen. Bei einem *'wiphy'* handelt es sich um die grundlegende C++-Datenstruktur eines jeden angeschlossenen *WNIC*, welche die Hardware und ihre Konfiguration beschreibt. Eine Anpassung ist erforderlich, da innerhalb der Funktion *nl80211_set_wiphy(...)* geprüft wird in welchem Modus sich die *WNIC* befindet. Eine Konfiguration der *EDCA*-Parameter wird dort nicht ermöglicht, wenn sich das Gerät im Mesh-Modus befindet (Quellcode 4.2, Zeile 11,12,13). Es ist dort ausreichend die drei entsprechenden Zeilen auszukommentieren, um auch eine Anpassung in einem anderen Modus, als dem Access Point (AP)- und Peer to Peer (P2P)-Modus zu ermöglichen. Trotzdem wird weiterhin ein Tool benötigt, um die Parameter über das *nl80211* Interface zu setzen. Eine genaue Erklärung dafür findet sich in Kapitel 4.5.2.

```
1   if (info->attrs[NL80211_ATTR_WIPHY_TXQ_PARAMS]) {
2     struct ieee80211_txq_params txq_params;
3     struct nlattr *tb[NL80211_TXQ_ATTR_MAX + 1];
4
5     if (!rdev->ops->set_txq_params)
6       return -EOPNOTSUPP;
7
8     if (!netdev)
9       return -EINVAL;
10    // Die drei folgenden Zeilen wurden auskommentiert.
11    //if (netdev->ieee80211_ptr->iftype != NL80211_IFTYPE_AP &&
12    //    netdev->ieee80211_ptr->iftype != NL80211_IFTYPE_P2P_GO)
13    //  return -EINVAL;
14 // code ist an dieser Stelle gekuerzt
15  }
```

Quellcode 4.2: net/wireless/nl80211.c - nl80211_set_wiphy(...): Funktion aus dem Linux-Kernel. Diese Funktion muss angepasst werden, damit die EDCA-Parameter auch im Mesh-Modus gesetzt werden können.

Hauptsächlich zu Test- und Debugzwecken wird auch das Modul *ath10k* angepasst. Dabei handelt es sich um den Treiber für die *WNIC*. Mit den vorhandenen Tools und Schnittstellen ist es nicht möglich die aktuell gesetzten EDCA-Parameter auszulesen. Die Anpassung fügt eine Nachricht in den Kernel-Ringbuffer hinzu, welche mithilfe des Tools *display message (DMESG)* ausgelesen wird. So wird sichergegangen, dass die Parameter tatsächlich auf der *WNIC* gesetzt worden und nicht später wieder verändert worden sind.

Die Kernel-Module werden wie folgt gebaut:

1. Herunterladen der Quellcode-Pakete [4][5][6]
2. Entpacken des Quellcodes
   $ dpkg-source -x linux_4.19.194-3.dsc
3. Anpassungen an Quellcode vornehmen
4. Exportierte Symbole des aktuellen Builds kopieren
   $ cp -v /usr/src/linux-headers-$(uname -r)/Module.symvers .
5. Build konfigurieren
   a) *$ make oldconfig*
   b) *$ make menuconfig*
6. Kernel-Module bauen
   a) *$ make scripts prepare modules_prepare*
   b) *$ make -C . M=net/wireless*
   c) *$ make -C . M=drivers/net/wireless/ath/ath10k*

Mit dem folgenden Script 4.3 werden die nun gebauten Kernel-Module ausgetauscht:

```
1  #!/bin/bash
2  # unload kernel modules and load changed modules
3  MODULEDIR=~/2021-behrens-johannes/kernel_modules
4
5  sudo rmmod ath10k_pci
6  sudo rmmod ath10k_core
7  sudo rmmod ath
8  sudo rmmod mac80211
9  sudo rmmod cfg80211
10 sudo insmod $MODULEDIR/cfg80211.ko
```

[4] `http://security.debian.org/debian-security/pool/updates/main/l/linux/linux_4.19.194-3.dsc`, Abgerufen am 10.11.2021
[5] `http://security.debian.org/debian-security/pool/updates/main/l/linux/linux_4.19.194.orig.tar.xz`, Abgerufen am 10.11.2021
[6] `http://security.debian.org/debian-security/pool/updates/main/l/linux/linux_4.19.194-3.debian.tar.xz`, Abgerufen am 10.11.2021

```
11 sudo modprobe mac80211
12 sudo modprobe ath
13 sudo insmod $MODULEDIR/ath10k_core.ko
14 sudo insmod $MODULEDIR/ath10k_pci.ko
```

Quellcode 4.3: changemodules.sh - Script zum Tausch der veränderten Module.

4.5.2 Tool zum Setzen der Enhanced Distributed Channel Access (EDCA) Parameter

Innerhalb eines *Mesh*-Netzwerks werden für die TX-Warteschlangen normalerweise die Standardparameter für den *EDCA* genutzt. Für die Experimente müssen jedoch die Backoff-Parameter der Warteschlangen angepasst werden. Um dies zu ermöglichen wurde ein Tool programmiert, welches die Werte über die *nl80211* Schnittstelle an den Kernel sendet. Dieses Tool funktioniert nur mit der zuvor vorgestellten Anpassung des Kernels. Der Quellcode findet sich in Anhang A.1. Die Abbildung **??** zeigt einen Programmablaufplan für das Tool, welches in diesem Abschnitt weiter erklärt wird.

Zu Beginn werden die an das Programm übergebenen Argumente auf ihre Gültigkeit geprüft. Orientiert wurde sich dort an den Werten aus dem Kernel-Header *include/net/cfg80211.h*. Eine Übersicht über die Parameter und deren möglichen Werte findet sich in Tabelle 4.3.

Nr.	Name	Beschreibung
1	AC	AC identifier (NL80211_AC_*). 0: VO, 1: VI, 2: BE, 3: BK
2	txop	Maximum burst time in units of 32 usecs, 0 meaning disabled
3	cwmin	a value of the form $2^n - 1$ in the range 1..32767
4	cwmax	a value of the form $2^n - 1$ in the range 1..32767
5	aifs	Arbitration interframe space [0..255]
6	interface	Index of the interface

Tabelle 4.3: Die Argumente für das Tool mit dem die EDCA-Parameter gesetzt werden.

Im Anschluss daran wird die *nl80211* Verbindung initialisiert. Hierfür wird ein neuer Socket allokiert, die Buffergröße gesetzt und die Verbindung zu dem Socket hergestellt. Dies geschieht alles in der Funktion *static intinitNl80211(Netlink *nl, Wifi *w)*. *nl* ist eine Struktur dar, die Informationen zu dem Socket speichert und *w* speichert Informationen über das *Wifi*-Interface. Innerhalb dieser Funktion werden auch callback-Funktionen registriert. Diese sind allerdings nicht unbedingt notwendig und dienten ursprünglich Debug-Zwecken, um empfangene Daten ausgeben zu können.

Jetzt besteht der netlink Socket und die Parameter sind auf ihre Gültigkeit geprüft. Mit diesen Daten wird die Funktion *setTxQNl80211(&nl, &w, ac, txop , cwmin , cwmax , aifs ,ifindex);* aufgerufen. Innerhalb dieser Funktion wird die Nachricht an den Kernel zusammengebaut und versendet. Der Vorgang beginnt damit, dass eine neue Struktur für eine netlink Nachricht allokiert wird (*nlmsg_alloc()*). Dieser Struktur werden die benötigten Header hinzugefügt (*genlmsg_put(...)*). Dabei handelt es sich um eine Nachricht, die die Informationen in dem *wiphy* verändern soll. Daher ist sie vom Typ *NL80211_CMD_SET_WIPHY*.

Als weitere Informationen fehlen die zu setzenden Daten, welche als eine eingenistete Struktur hinzugefügt werden. Zum Schluss wird die Nachricht versendet und der Speicher wieder freigegeben. Jetzt muss nur noch die Socket-Verbindung geschlossen werden und das Programm endet.

4.6 Erzeugung von Datenverkehr und Messen der Performance (iperf2)

Um das Ziel zu erreichen, dass die Simulation in einem realen System wiederholt wird, müssen Daten über ein Netzwerk versendet werden. Hierfür wird eine Software benötigt, welche Datenströme erzeugt und Statistiken dazu erstellt. In diesem Kapitel wird zuerst festgelegt, was die Anforderungen an eine solche Software sind. Daraufhin wird die genutzte Software vorgestellt und gezeigt, welche Anpassungen getroffen werden müssen.

4.6.1 Auswahl des Tools

In den Experimenten müssen Datenströme erzeugt und versendet werden. Diese Datenströme sollen verbindungslos über das *User Datagram Protocol (UDP)* versendet werden. Für jeden Datenstrom soll pro Paket die Latenzzeit ermittelt werden. Von weiterem Interesse ist der Jitter. Damit die Berechnung des Jitters erfolgen kann, werden Zeitstempel für den Empfang der Nachricht benötigt. Vorgegeben werden soll je Datenstrom die Paketrate (Pakete pro Sekunde), die Größe eines Pakets und der *Type of Service (TOS)* der Pakete. Anhand des *TOS* werden die Pakete vom Linux-Kernel in die passende Warteschlange geleitet. Gleichzeitig soll der durch das verwendete Tool entstehende Overhead minimal sein, da dieser das Netzwerk zusätzlich belastet.

Diesen Anforderungen entspricht das open source Tool *iperf2* [7]. Das Software-Tool kann die Netzwerkperformance in Bezug auf Latenz und Datendurchsatz beurteilen. Es kann festgelegt werden, welche Pakete in einem bestimmten Zeitraum versendet werden sollen. Dabei werden die Protokolle *Transmission Control Protocol (TCP)* und *UDP* unterstützt. Außerdem ist *iperf2* zu vielen Plattformen und Betriebssystemen kompatibel [31]. Für die Nutzung muss *iperf2* auf einem der Knoten als Server (um Pakete zu empfangen) und auf einem weiteren Knoten als Client (um Pakete zu generieren und versenden) gestartet werden [32].

Es wurde sich gezielt für *iperf2* und nicht für *iperf3*[8] entschieden. Der Grund dafür ist, dass *iperf3* eine neue Implementierung ist und einen anderen Funktionsumfang als *iperf2* bietet. So ist es zum Beispiel nicht möglich mit *iperf3* die Latenzen für *UDP* zu ermitteln [33].

Für die Ermittlung der Latenz mit dem Tool *iperf2* ist die genaue Synchronisierung der Systemzeiten sehr wichtig, da es sich hierbei um eine Ende-zu-Ende-Latenz handelt. Es wird also ein Zeitstempel mitgesendet, welcher mit einem Zeitstempel beim Empfang verglichen wird. In den Manuals wird hierfür das Protokoll *Precision Time Protocol (PTP)* empfohlen. Die genaue Implementierung, Konfiguration und Anpassung von *iperf2* für die Experimente wird im Folgenden beschrieben.

4.6.2 Anpassung und Konfiguration

Genutzt wird für die Experimente *iperf2*[9] in der Version 2.1.4-rc. Standardweise gibt *iperf2* die Ergebnisse erst zum Ende der Ausführung aus. Dort ist dann nur ein Durchschnittswert,

[7] `https://sourceforge.net/projects/iperf2/`, Abgerufen am 12.08.2021
[8] `https://github.com/esnet/iperf`, Abgerufen am 12.08.2021
[9] `https://sourceforge.net/projects/iperf2/`, Abgerufen am 27.09.2021

der Minimal- und Maximalwert und eine Standardabweichung für die Latenz ersichtlich. Für die wissenschaftliche Auswertung der Experimente werden allerdings die Latenzen für jedes einzelne Paket benötigt. Daher wird *iperf2* so angepasst, dass nach jedem Paket der Sendezeitpunkt, Empfangszeitpunkt und die Latenz dieses Pakets ausgegeben wird. Dafür wird in dem Quellcode *src/Reporter.c* in der Funktion *reporter_handle_packet_oneway_transit(...)* direkt vor dem return-Statement folgendes ergänzt:

```
1 printf("%f ", transit);
2 printf("%f ", TimeDouble(packet->sentTime));
3 printf("%f\n", TimeDouble(packet->packetTime));
```

Quellcode 4.4: iperf-2.1.4-rc/src/Reporter.c - Der Code wird direkt vor dem return-Statement der Funktion *static inline double reporter_handle_packet_oneway_transit (struct ReporterData *data, struct ReportStruct *packet)* in den Zeilen 736 - 738 hinzugefügt. Dadurch wird für jedes empfangene Paket der Empfangszeitpunkt, Sendezeitpunkt und die Latenz ausgegeben.

Diese Funktion wird bei jedem empfangen Paket aufgerufen. Damit ergibt sich je Paket eine Zeile Informationen in der Standardausgabe. Die erste Spalte beinhaltet die Latenz, die zweite den Sendezeitpunkt und die dritte den Empfangszeitpunkt des Pakets. Anschließend kann *iperf2* kompiliert werden. Wichtig ist die vorherige Konfiguration, so dass *iperf2* eine Genauigkeit von 100 Mikrosekunden bietet. Hierfür muss das Konfigurationsscript wie folgt aufgerufen werden: *./configure –enable-fastsampling*. Anschließend reicht es, wenn das make-Script aufgerufen wird. Nach der Kompilierung befindet sich die *iperf2*-Binärdatei in dem Ordner *src/*.

Der *iperf2* Server kann dann als Testbed-Action über das *TBMS* mit dem folgenden Kommando (Quellcode 4.5) aufgerufen werden:

```
1 /home/testbed-user/johannes/iperf-2.1.4-rc/src/iperf -s -p 5004 -u -l {
    audio_packetsize} -e -t {explengthS} -o /home/testbed-user/johannes/
    results/{expID}/iperf/audio/server/iperf.txt
```

Quellcode 4.5: Testbed-Action: Start einer iperf2-Server-Instanz.

Als Beispiel wird hier nur der Aufruf für die Instanz des Servers für die Audiodaten dargestellt. Innerhalb der geschweiften Klammern stehen die Variablen, welche über das *TBMS* festgelegt werden. Die Flags haben dabei die folgenden Bedeutungen:

-b Clientseitig wird die Bandbreite der zu sendenden Daten festgelegt (bit/s).

-c iperf2 wird im Client-Modus gestartet. Als Parameter wird die IP-Adresse des Servers angegeben.

-e Aktiviert einen erweiterten Report zum Ende des Experiments.

-l Die Größe eines UDP-Pakets wird hiermit festgelegt (Byte).

-o Umleiten der Ausgaben in die hier angegebene Datei.

-p Port für die Experimente (haptic: 5002, haptic feedback: 5003, audio: 5004, video: 5005)

Zugriffskategorie	TOS (iperf2)
Voice (AC_VO)	0xC0, 0xB8, 0xE0
Video (AC_VI)	0x80, 0xA0, 0x88
Best Effort (AC_BE)	0x00, 0x60, or other
Background (AC_BK)	0x40, 0x20
Tactile Internet (AC_TI)	0x40, 0x20

Tabelle 4.4: Die Tabelle zeigt die Zuordnung der genutzten TOS-Werte zu den Warteschlangen [34].

-s iperf2 wird im Server-Modus gestartet.

-S Setzt das IP_TOS Feld innerhalb des IP-Headers, wodurch die Daten in die richtige Warteschlange einsortiert werden.

-t Client: Zeit der Datenübertragung. Server: Warte- und Empfangszeit. [Sekunden]

-u iperf2 im UDP-Modus ausführen.

Die entsprechende Testbed-Action für die *iperf2*-Client-Instanz sieht damit so aus (Quellcode 4.6):

```
1 /home/testbed-user/johannes/iperf-2.1.4-rc/src/iperf -p 5004 -c {bobIP}
     -u -t {explength} -b {audio_bandwidth} -S {av_tos} -l {
     audio_packetsize} -e -o /home/testbed-user/johannes/results/{expID}/
     iperf/audio/client/iperf.txt
```

Quellcode 4.6: Testbed-Action: Start einer iperf2-Client-Instanz.

Die Aufrufe der Testbed-Actions für *iperf2* werden durch das *TBMS* durchgeführt und parallel ausgeführt. So laufen die einzelnen Server- und Clientinstanzen nebenläufig. Bei dem *TOS*-Parameter muss aufgepasst werden, dass die Pakete tatsächlich in die korrekte Warteschlange einsortiert werden. Der Linux-Kernel sortiert die Pakete ausschließlich anhand des *TOS*-Felds innerhalb des IPv4-Headers. *iperf2* benutzt hierbei die ursprüngliche Spezifikation von IPv4 (RFC 791). So ergibt sich die Zuordnung (Tabelle 4.4) der *TOS*-Werte zu den einzelnen Warteschlangen. Da die haptischen Daten unter Nutzung der taktlilen Koordinierungsfunktion über die Warteschlange für Hintergrunddaten gesendet werden, muss hier der dafür entsprechende *TOS* genutzt werden.

4.7 Zeitsynchronisierung mit Precission Time Protocol (PTP)

Entscheidend für die Genauigkeit der Ergebnisse von *iperf2* ist, dass die Uhren der Testbed-Knoten sehr genau übereinstimmen. Hierfür müssen sie durchgehend synchronisiert werden. Daher wird für die Synchronisierung der Uhren wie in der *iperf2* Dokumentation empfohlen das *PTP* genutzt.

In der exakten Umsetzung auf dem Testbed wird zu Beginn der Experimente auf allen Knoten der *Network Time Protocol (NTP)*-Service gestoppt (Quellcode 4.7). Dies verhin-

dert, dass die Zeit während der Synchronisierung verfälscht wird. Auf einem der Knoten, welcher *PTP*-Master genannt wird, wird die Zeit mittels des *NTP* mit dem *NTP*-Server der Otto-von-Guericke-Univerität Magdeburg synchroniert (time.ovgu.de) (Quellcode 4.8). Dieser Schritt ist nicht notwendig, allerdings ist es bei der Auswertung der Ergebnisse übersichtlicher, wenn die Zeit mit der tatsächlichen Zeit übereinstimmt. Alle weiteren Knoten außer des *PTP*-Masters werden ab jetzt innerhalb dieses Kapitels *PTP*-Slave genannt und werden die Zeit von dem *PTP*-Master beziehen.

```
1 sudo service ntp stop
```

Quellcode 4.7: Stoppen der Synchronisierung über das NTP.

```
1 sudo ntpdate -s time.ovgu.de
```

Quellcode 4.8: Einmaliges Synchronisieren der Zeit über das NTP.

Nach den Vorbereitungen beginnt die eigentliche Synchronisierung der Zeit zwischen den Knoten. Genutzt wird dafür das Testbed-Netzwerk (ethernet) und die *PTP* Implementierung *linuxptp*[10]. Von dieser Implementierung werden die beiden Tools *ptp4l* und *phc2sys* genutzt.

Das Tool *ptp4l* führt die eigentliche Synchronisierung der Uhrzeit zwischen den Knoten durch. Es hat die Aufgabe die Hardwareuhren der *Network Interface Card (NIC)*s untereinander zu synchronisieren. Dafür wird eine Masteruhr festgelegt. Diese Masteruhr wird auf der Seite des *PTP*-Masters bereitgestellt (Quellcode 4.9). Bei den *PTP*-Slaves wird die Zeit von der Masteruhr auf die Hardwareuhren übernommen (Quellcode 4.10). Damit der *PTP*-Master tatsächlich als Masteruhr genutzt wird, wird *ptp4l* dort mit der *priority1*-Option konfiguriert.

```
1 sudo ./linuxptp/ptp4l -i eth0 -f ./ptp4lMaster.conf
```

Quellcode 4.9: Startet ptp4l auf dem PTP-Master. Dabei wird das Netzwerkinterface mit dem Flag -i festgelegt. Die Konfigurationsdatei enthält die Priorität der Uhr.

```
1 sudo ./linuxptp/ptp4l -i eth0 -s
```

Quellcode 4.10: Startet ptp4l auf den PTP-Slaves. Dabei wird das Netzwerkinterface mit dem Flag -i festgelegt. -s aktiviert den slaveOnly Modus.

Jetzt muss die Uhrzeit der Hardwareuhr mit der Systemuhr synchronisiert werden. Dafür wird das Tool *phc2sys* benutzt. Auf der Seite des *PTP*-Masters wird damit die zuvor über *NTP* synchronisierte Systemzeit auf der *NIC* bereitgestellt (Quellcode 4.11). Die *PTP*-Slaves synchronisieren die Uhrzeit ihrer NIC in Richtung der Systemuhr (Quellcode 4.12). Wichtig ist die hohe Update-Rate von 100 Hz, welche mit dem Argument *-R* festgelegt wird. In ersten Experimenten hat sich gezeigt, dass eine geringere Rate nicht ausreichend ist. Dort kam es zu größeren Abweichungen der Zeit, welche nur langsam angepasst werden konnten.

[10] `https://sourceforge.net/projects/linuxptp/`, Abgerufen am 07.09.2021

```
1 sudo ./linuxptp/phc2sys -a -r -r -R 100 -S 0.1
```

Quellcode 4.11: Startet phc2sys auf dem PTP-Master. Nimmt die Systemzeit als Zeitquelle und synchronisiert sie automatisch mit der durch ptp4l festgelegten Uhr.

```
1 sudo ./linuxptp/phc2sys -a -r -R 100 -S 0.1
```

Quellcode 4.12: Startet phc2sys auf den PTP-Slaves. Es wird automatisch die durch ptp4l synchronisierte Hardwareuhr als Quelle genommen und mit der Systemuhr synchronisiert.

Das hier beschriebene Vorgehen wird mit der Hilfe mehrerer Scripte und Services umgesetzt. Auf der Seite des *PTP*-Masters wird das Script *ptpMaster.sh* ausgeführt, welches die Dienste *ptp4lmaster.service* und *phc2sysslave.service* einrichtet und startet. Die dazu analogen Scripte und Service existieren ebenfalls für die *PTP*-Slaves.

Da das Ethernet-Netzwerk des Testbeds keine Pakete an die standardmäßig genutzte *PTP* Multicast-Adresse (224.0.1.129) weiterleitet, wurde diese im Quellcode von *linuxptp* auf die Adresse 224.0.0.42 angepasst. Diese Änderung erfolgt in der Datei *udp.c (#define PTP_PRIMARY_MCAST_IPADDR "224.0.0.42")*. Mithilfe von *PTP* können Genauigkeiten von bis zu einer 1 µs erreicht werden [35]. Die tatsächliche Genauigkeit ist jedoch stark von der zugrundeliegenden Netzwerkarchitektur abhängig.

4.8 Nutzung des Git-Repositories und Integration in MIoT-Testbed

Die für die Experimente benötigten Dateien und Tools werden zu Beginn eines jeden Experiments aus einem *Git-Repository* in den Ordner */home/testbed-user/johannes* geklont (Quellcode 4.13). Damit die Nutzerdaten für das *Git* nicht in dem *DES-Cript* hinterlegt werden müssen, wird ein *Personal Access Token (PAT)* genutzt.

```
1 git clone https://oauth2:XXXXXXXXX@code.ovgu.de/comsys-group/students/
    msc/2021-behrens-johannes.git /home/testbed-user/johannes
```

Quellcode 4.13: Testbed-Action: Auschecken des Git-Repositories. Der PAT ist unkenntlich gemacht.

Der Ordner in den das *Git*-Verzeichnis geklont wird befindet sich auf einem für alle Knoten erreichbaren Netzwerklaufwerk. So muss er nur einmalig durch einen Knoten heruntergeladen werden. In diesen Ordner werden auch die Ergebnisse der Experimente gespeichert, so dass sie nach dem Experiment automatisch durch das Testbed in das *Git*-Repository commited und gepusht werden können. Schlussendlich wird der zu Beginn erstellte Ordner wieder gelöscht (Quellcode 4.14).

Vor der Durchführung eines Experiments werden die Knoten durch das Testbed neugestartet. Dies verhindert, dass Konfigurationen aus vorherigen Experimenten verbleiben. Zur Vereinfachung der Durchführung der einzelnen Experimente werden Variablen des *TBMS* genutzt. Erkennbar sind sie in den Testbed-Actions dadurch, dass sie in geschweiften Klammern *{Variable}* stehen.

```
cd /home/testbed-user/johannes/
git config user.email "testbed-user@testbed"
git config user.name "testbed-user"
git commit -m "commit from testbed"
git push
sudo rm -r /home/testbed-user/johannes/
```

Quellcode 4.14: Testbed-Action: Beendigung eines Experiments. Die Ergebnisse werden commited und in das Git-System gepusht.

4.9 Auslastung des Funkkanals

Um die Auslastung des Funkkanals einschätzen zu können wird vor und nach jedem Experiment mit dem Tool *iw* ein so genannter *survey dump* erstellt. Dieser enthält Informationen darüber wie lange die *WNIC* auf dem gewählten Kanal aktiv war, wie lange der Kanal als belegt erkannt wurde und für wie lange Daten sowohl gesendet, als auch empfangen worden sind. Für die Auswertung werden die Daten in ein gemeinsames Dokument geschrieben, so dass sie mit Python evaluiert werden können. Der dazugehörige Testbed-Aufruf ist in Quellcode 4.15 zu finden.

```
iw dev mesh0 survey dump > /home/testbed-user/johannes/results/{expID}/
    surveydumpafter/{_current#node.name}.txt
```

Quellcode 4.15: Testbed-Action: Erstellen des Survey-Dumps. Wird vor und nach einem Experiment auf jedem Knoten ausgeführt.

Da es zu einem nicht weiter nachvollziehbaren Problem mit dem *Address Resolution Protocol (ARP)* unter der Nutzung des 5 GHz Frequenzbereichs gekommen ist, werden zu Beginn die *ARP*-Tabellen mit einem Script (Quellcode 4.16) fest konfiguriert. Bei dem Problem war keine Kommunikation zwischen den Knoten möglich.

```
#!/bin/bash
# manually create arp table
arp -s 192.168.4.1 04:f0:21:46:36:34
arp -s 192.168.4.2 04:f0:21:54:42:a5
arp -s 192.168.4.3 04:f0:21:54:42:de
arp -s 192.168.4.4 04:f0:21:54:3f:27
arp -s 192.168.4.5 04:f0:21:54:3f:39
arp -s 192.168.4.6 04:f0:21:54:2f:c6
arp -s 192.168.4.7 04:f0:21:54:2f:bd
arp -s 192.168.4.8 04:f0:21:54:2f:b7
```

Quellcode 4.16: arp.sh - Script erstellt die ARP-Tabellen für die Testbed-Knoten.

4.10 Setzen der Hybrid Wireless Mesh Protocol (HWMP) Routen (Mesh)

Für die Experimente soll sichergestellt werden, dass die gewünschten Routen zwischen den Knoten eingehalten werden. Nur so sind die Experimente miteinander vergleichbar und auch innerhalb eines Experiments treten keine Änderungen in den Routen auf. Dafür werden die Routen auf den Testbed-Knoten mit der Testbed-Action 4.17 und dem Tool *iw* festgelegt.

```
1 iw dev <devname> mpath new <destination MAC address> next_hop <next hop
     MAC address>
```

Quellcode 4.17: Testbed-Action: Festlegen einer Route zu einer MAC-Adresse.

KAPITEL 5

Experimente

In diesem Kapitel werden die mit der Implementierung durchgeführten Experimente vorgestellt. Mit diesen sollen die Fragen beantwortet werden, ob sich die positiven Ergebnisse aus der Simulation in einer echten Welt wiederholen lassen und wie die weitere Kommunikation beeinflusst wird.

Die Experimente orientieren sich an dem Anwendungsfall *Teleoperation* der *IEEE 1918.1 "Tactile Internet" Standards Working Group*. Bei diesem Szenario taucht ein menschlicher Nutzer in eine entfernte oder nicht erreichbare Umgebung ein und erfüllt dort eine komplexe Aufgabe. Ein solches *Teleoperation System* besteht aus einem *Master* (menschlicher Nutzer) und einem *Slave* (zu steuernde entfernte Anwendung). Zwischen dem *Master* und dem *Slave* werden kontinuierlich über ein Kommunikationsnetz Daten ausgetauscht. In die Richtung des *Masters* sind dies Audio- und Videodaten sowie ein haptisches Feedback. Der *Master* übertragt an den *Slave* wiederum haptische Daten zur Steuerung der Anwendung [5].

Im weiteren Verlauf der Arbeit wird das konkrete Beispiel mit einem Roboter (*Slave*), welcher von einem menschlichen Nutzer (*Master*) aus der Ferne gesteuert wird, genutzt. Der Roboter verfügt über eine Kamera, ein Mikrofon und eine Vielzahl an Sensoren. Diese übermitteln unter anderem Informationen über die Geschwindigkeit, die Position und den Zustand des Roboters. Die Daten werden an den menschlichen Nutzer gesendet und dort auf geeignete Weise dargestellt beziehungsweise wiedergegeben. Der Nutzer wiederum trifft anhand dieser Daten Entscheidungen und steuert den Roboter aus der Ferne. So entsteht eine geschlossene Steuerungsschleife. Zwischen dem Roboter und dem menschlichen Nutzer befindet sich ein Kommunikationsnetzwerk, wobei der Roboter drahtlos über *IEEE 802.11s (mesh)* angebunden ist. Dieser Anwendungsfall ist in Abbildung 5.1 dargestellt.

In den folgenden Abschnitten wird anhand des beschriebenen Anwendungsbeispiels die Durchführung der Experimente festgelegt. Dafür wird auf die Anforderungen, die an ein Netzwerk für das *Taktile Internet* gestellt werden, eingegangen. Das Kapitel endet mit genauen Definitionen der einzelnen Experimente.

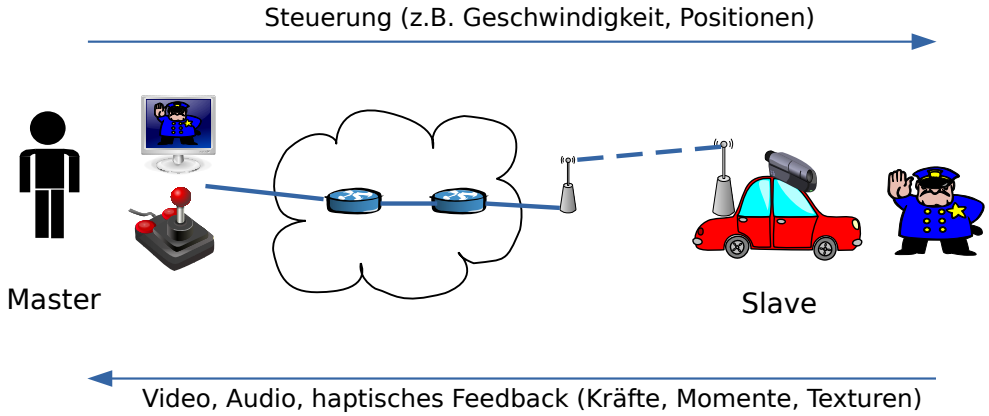


Abbildung 5.1: Fallbeispiel für das Konzept der Experimente. Ein Roboter (Slave) wird aus der Ferne von einem menschlichen Nutzer (*Master*) gesteuert. (Eigene Abbildung in Anlehnung an [5])

5.1 Anforderungen an ein Netzwerk für das *Taktile Internet*

In dem Implementierungsteil der Arbeit wurde ermöglicht, dass Daten mit einer festgelegten Größe, Frequenz und Zugriffskategorie innerhalb der Taktilen Koordinierungsfunktion versendet werden können. Gleichzeitig werden die Latenz, der Jitter und der Paketverlust dieser Datenströme erfasst. Aufbauend darauf werden die Experimente ausgestaltet.

Um zu beurteilen, ob die *Taktile Koordinierungsfunktion* für das *Taktile Internet* geeignet ist, müssen für die Experimente geeignete Datenströme erzeugt werden. Das bedeutet, dass die Paketgröße und die Sendefrequenz entsprechend eines realistischen Anwendungsfalls gewählt werden müssen. Hierbei wird sich an den bereits vorgestellten Anwendungsfall aus Abbildung 5.1 orientiert. Es soll ein entfernter Roboter von einem menschlichen Nutzer ferngesteuert werden.

Bei den Daten, die vom Menschen an den Roboter gesendet werden, handelt es sich um haptische Daten zur Steuerung des Roboters, welche beispielsweise über einen Joystick aufgenommen werden. Insgesamt werden sechs Freiheitsgrade (je Freiheitsgrad ein Gleitkommawert, double → 8 B) übertragen. Daraus ergibt sich eine Paketgröße von jeweils 48 Byte. Da sich der Roboter auf einer Fläche mit anderen Teilnehmern bewegt, handelt es sich um ein hochdynamisches Umfeld, wodurch eine Latenz von 1 ms - 10 ms gefordert wird. Außerdem wird eine Komprimierung des Datenstroms angenommen. Dies führt zu einer Frequenz der zu sendenden Daten von 500 Paketen pro Sekunde (192 kbps) [5]. In die entgegengesetzte Richtung werden ebenfalls haptische Daten übertragen. Dabei handelt es sich um taktile Signale, die ein Feedback an den menschlichen Nutzer geben. Dies sind unter anderem Kräfte oder Informationen über Texturen der Oberfläche, welche über den Joystick in Form von Vibrationen eingespielt werden können. Die Anforderungen sind hierbei die gleichen wie bei den Steuerungsdaten. Allerdings sind die Datenpakete mit jeweils 80 B (Annahme: 10 Freiheitsgrade) größer als bei der Steuerung des Roboters. So ergibt sich mit einer Sendefrequenz von 500 Pakete pro Sekunde eine Datenrate von 320 kbps [5]. Laut Holland et al. [5] kann für den Anwendungsfall *Teleoperation* eine durchschnittliche Datenrate von 1 Mbps - 100 Mbps für die Videoübertragung bei einer Paketgröße von je-

Datentyp	Paketgröße	Zuverlässigkeit	Latenz	Datenrate	PPS
Haptisches Feedback	80 B	99,999 %	1-10 ms	320 kb/s	500 pkts/s
Video	1,5 kB	99,999 %	10-20 ms	1 Mb/s	83,3 pkts/s
Audio	50 B	99,9 %	10-20 ms	256 kb/s	640 pkts/s
Haptisch	48 B	99.999 %	1-10 ms	192 kb/s	500 pkts/s

Tabelle 5.1: Für die Experimente genutzte Paketgrößen und Datenraten mit den dazugehörigen geforderten Zuverlässigkeiten und Latenzen [5].

weils 1,5 kB vom *Slave* zum *Master* angenommen werden. Da die Experimente auch mit mehreren Hops durchgeführt werden sollen und dabei die Bandbreite nicht zum begrenzenden Element werden soll, wird eine Datenrate von 1 Mbps ausgewählt. Dies entspricht einer Sendefrequenz von 83,3 Paketen pro Sekunde. Dabei muss eine Latenz von 10 ms - 20 ms eingehalten werden. Die Latenzanforderungen für die Übertragung der Audiodaten sind die gleichen. Allerdings werden hierbei 50 B große Pakete mit einer Datenrate von 256 kbps übertragen (640 Pakete pro Sekunde). Eine Übersicht über die Daten findet sich in der Tabelle 5.1.

5.2 Auswahl der Experimente

In diesem Kapitel soll gezeigt werden, dass sich die positiven Ergebnisse aus der Simulation von Engelhardt et al. auch in der realen Welt reproduzieren lassen. Dafür wird eine Reihe an Experimenten durchgeführt, welche den simulierten Experimenten ähneln. In den Simulationen sind die zusätzlich zu den haptischen Daten übertragenen Daten über die Zugriffskategorie *Background* versendet worden. Diese wird in der Praxis allerdings nahezu nicht genutzt und findet daher auch keine Anwendung in dieser Masterarbeit. Die meisten Daten werden standardmäßig mit der Zugriffskategorie *Best Effort* versehen.

Für die hier durchgeführten Experimente zu der taktilen Koordinierungsfunktion wird angenommen, dass die zusätzlichen Daten (Audio und Video) auch Teil des *Taktilen Internets* sind. Damit gelten auch dafür spezielle Anforderungen an das Netzwerk. Entsprechend werden sie über die Kategorie *Voice* versendet, um eine möglichst hohe Priorität zu erreichen und trotzdem eine Abstufung zu den haptischen Daten zu besitzen, welche über die Kategorie *Taktiles Internet* versendet werden. Zusätzlich werden die Experimente auch so durchgeführt, dass alle Daten in einem Experiment mit der Zugriffskategorie *Best Effort* und in einem weiteren Experiment mit der Kategorie *Voice* versehen werden. Die Daten der Kategorie *Best Effort* entsprechen dem Fall, dass keine Priorisierung vorgenommen wird (Normalfall). Das Versehen aller Daten mit der Kategorie *Voice*, spiegelt den *Best Case* unter der Nutzung der Standard-Backoff-Parameter wieder.

Unabhängig davon werden in gesonderten Experimenten ausschließlich die haptischen Daten übertragen. Es wird angenommen, dass die Latenzen allein aufgrund der geringeren *AIFS* bessere Ergebnisse liefern. Als Referenzwerte werden Latenzen gewählt, welche ausschließlich durch das Betriebssystem und *iperf2* erzeugt werden. Dafür werden die Experimente

ohne einen Hop (localhost) durchgeführt. Eine Übersicht aller Experimente findet sich in Tabelle 5.4.

In den Experimenten werden die Daten ausschließlich in eine Richtung versendet. Alle *iperf2* Server-Anwendungen werden auf dem gleichen Testbed-Knoten ausgeführt und die Client-Anwendungen auf dem entsprechenden anderen Testbed-Knoten. Diese Vereinfachung wird vorgenommen, da Störeffekte, die sich auf die Ergebnisse auswirken, wie zum Beispiel das *Hidden Node Problem*, vermieden werden sollen.

Die Uhrzeit wurde für die Durchführung der Experimente nicht berücksichtigt, da die Auslastung des Netzwerks nicht direkt mit der Uhrzeit korreliert. Eine Erfassung der Anzahl an Personen innerhalb der Fakultät für Informatik ist leider nicht möglich. Auch kann in der Nacht nicht davon ausgegangen werden, dass das Gebäude ungenutzt ist, da es durchgängig geöffnet ist. Allerdings wird darauf geachtet, dass alle Experimente nacheinander ausgeführt werden, damit keine großen Änderungen innerhalb der Umgebung stattfinden.

5.3 Ablauf der Experimente

Das Sequenzdiagramm in Abbildung 5.2 zeigt zusammenfassend den kompletten Ablauf eines Experiments. Die einzelnen Schritte werden in Kapitel 4 genauer erklärt. Eine schriftliche Zusammenfassung des Ablaufs folgt in diesem Kapitel.

Ein Experiment beginnt immer damit, dass alle dafür benötigten Daten aus einem *Git-Repository* geklont werden. Dazu gehören die Tools, Kernelmodule und Konfigurationsdateien, welche für die Experimente genutzt werden. Anschließend wird die Zeit zwischen den einzelnen Testbed-Knoten synchronisiert, um die Latenzen mit *iperf2* korrekt zu ermitteln. Dafür wird das *PTP* genutzt. Als Masterzeit wird davor einmalig bei dem *PTP-Master* die Zeit über NTP synchronisiert.

Um die Experimente durchzuführen wird die *mesh* Netzwerkverbindung zwischen den Knoten aufgebaut. Falls die *TKF* für die Experimente genutzt wird, so werden zuvor die veränderten Kernelmodule geladen. Diese ermöglichen das Setzen der Backoff-Parameter im *mesh*-Modus. Für die Reproduzierbarkeit werden die Routen zwischen den Knoten fest gesetzt. Dann kann mit den Experimenten mit *iperf2* gestartet werden. Zum Abschluss werden die Ergebnisse in das *Git* übertragen und die Knoten von den Experimentdaten bereinigt.

In Abbildung 5.2 wird ausschließlich der Fall betrachtet, dass sich zwischen dem Server und dem Client nur ein Hop befindet. Sollten dort noch weitere Hops benötigt werden, so werden alle dazwischenliegenden Knoten ebenfalls mit der Zeit synchronisert, die Kernelmodule getauscht und die Routen werden gesetzt.

5.4 Dauer eines Experiments

Entscheidend für die Ergebnisse der Experimente ist die Dauer eines einzelnen Experiments. Direkt daraus ergibt sich die Anzahl an gemessenen Latenzen. Gerade bei Experimenten mit vielen Störeinflussen und dadurch vielen und großen Ausreißern muss die Dauer mit Bedacht gewählt werden. Je länger die Dauer eines Experiments ist, desto genauer können die Ergebnisse mithilfe eines Konfidenzintervalls angegeben werden. Problematisch ist aller-

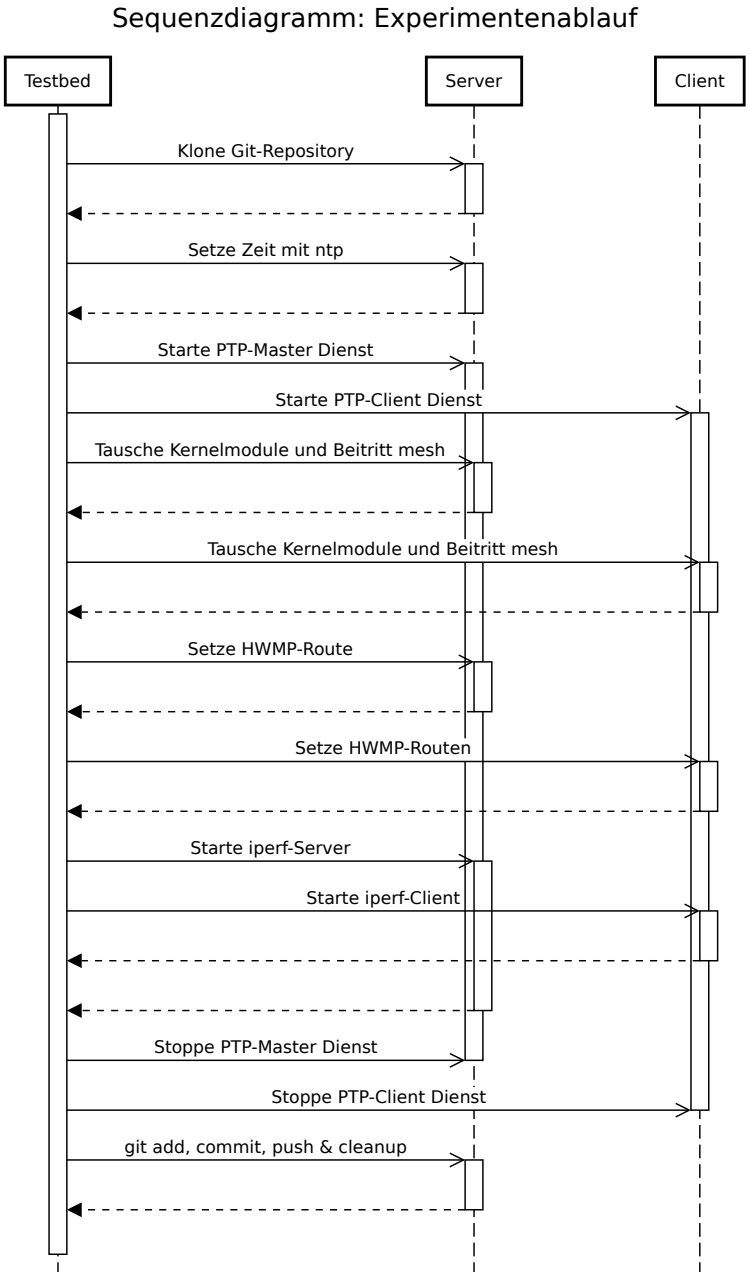


Abbildung 5.2: Vereinfachter Ablauf über die Experimente. Weitere Informationen dazu finden sich in Abschnitt 5.3. (Eigene Abbildung)

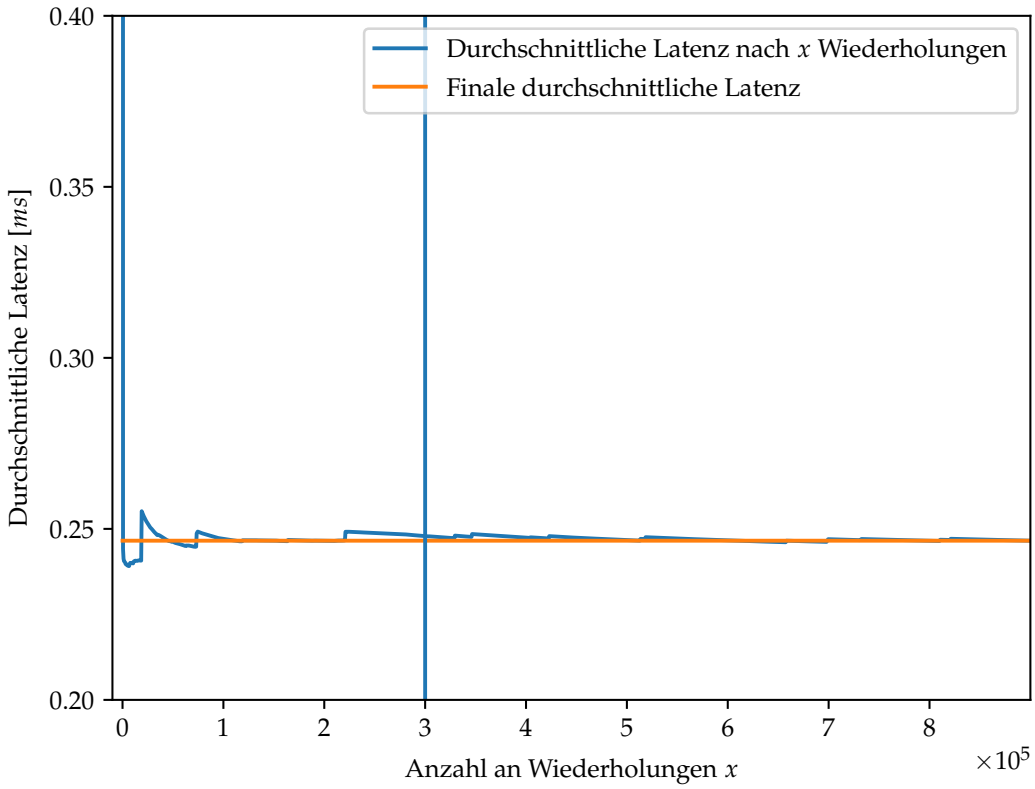


Abbildung 5.3: Annäherung des Durchschnittwerts der Latenz an den finalen Durchschnittswert. (Eigene Abbildung)

dings die schnell wachsende Menge an Daten und der Umgang damit. Um eine angemessene Dauer für die Experimente zu finden wird als erstes ein Vorexperiment ähnlich zum Experiment Nr. 3 durchgeführt. Für dieses Vorgehen wird sich an einem Guide[1] für Maschinelles Lernen orientiert. Zu Beginn wird die Dauer eines Experiments auf 30 Minuten gesetzt. Dies erfolgt unter der Annahme, dass sich spätestens nach 30 Minuten der gemessene Mittelwert auf einen Wert in der Nähe des wahren Mittelwerts eingependelt hat.

In einem ersten Schritt der Auswertung wird die Anzahl an Wiederholungen des Experiments auf der x-Achse dargestellt und dazu auf der y-Achse der Mittelwert zu dem Zeitpunkt der entsprechenden Wiederholung des Experiments. Mit der orangenen Linie wird der Mittelwert zum Ende des kompletten Experiments angezeigt. So kann schnell erkannt werden, ab wie vielen gemessenen Latenzen sich der Mittelwert an das endgültige Ergebnis annähert. Der Graph ist in Abbildung 5.3 zu sehen. Dort wird deutlich, dass bereits sehr früh (nach ca. 100000 Wiederholungen) der Mittelwert nahe dem Mittelwert zum Ende des Experiments ist. Nach 300000 Wiederholungen, was einer Experimentdauer von 10 Minuten entspricht, besteht kaum noch eine Abweichung. Die so entstehende Datenmenge ist noch gut handhabbar. Daher wird die Experimentdauer vorerst auf 10 Minuten gesetzt.

[1] `https://machinelearningmastery.com/estimate-number-experiment-repeats-stochastic-machine-learning-algorithms/`, Abgerufen am 09.11.2021

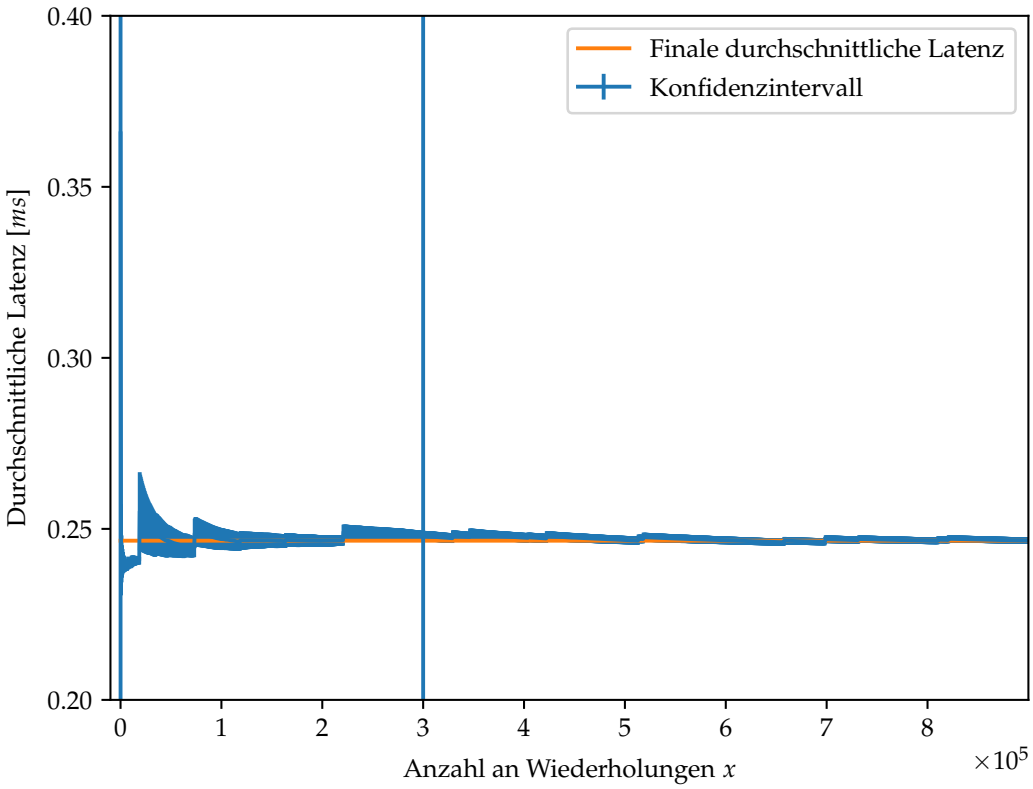


Abbildung 5.4: Konfidenzintervalle der Latenz je Anzahl an Wiederholungen des Experiments. (Eigene Abbildung)

In Abbildung 5.4 werden die sich ergebenden 99%-Konfidenzintervalle zu den einzelnen Anzahlen an Wiederholungen dargestellt. Die orangene Linie zeigt dabei erneut den finalen Mittelwert zum Ende des Experiments. Dabei dient dieser als Schätzung für den wahren Mittelwert der Population und kann genutzt werden, um zu sehen, ob die Konfidenzintervalle den Mittelwert beinhalten. Abbildung 5.4 bestätigt, dass 300000 Wiederholungen des Experiments ausreichend sind. Bereits dort ist das Konfidenzintervall sehr schmal und deckt auch den entgültigen Durchschnittswert ab.

Trotz der sehr sorgfältigen Durchführung der Experimente und einer bewussten Wahl der Experimentendauer kann es bei Wiederholungen der hier durchgeführten Experimente an anderen Orten und zu unterschiedlichen Zeiten zu Abweichungen in den Ergebnissen kommen. Dies ist auf die große Anfälligkeit von Funknetzwerken auf das Umfeld zurückzuführen.

5.5 Zusammenfassung über Versuchsaufbau

In diesem Abschnitt wird das zuvor gewonnene Wissen in einer einfachen Darstellung für die Durchführung der Experimente zusammengefasst. Dies erleichtert die Replikation der Experimente für die weitere Forschung. Die ausführliche Erläuterung und Herleitung dieser Informationen erfolgte im Laufe der letzten beiden Kapitel.

5.5.1 Testbed-Knoten

Für die Experimente wird das *MIoT-Testbed* genutzt. Die Testbed-Knoten besitzen die in Tabelle 5.2 aufgelistete Hardware.

PC Engines APU.3C4 Board	
CPU	1,0 GHz AMD Embedded G-Series GX-412TC (Quad-Core, 64 Bit)
RAM	4 GByte DDR3-1333 DRAM
WNIC	Qualcomm Atheros QCA986x/988x 802.11ac
OS	Debian GNU/Linux 10 (buster)
Kernel	#1 SMP Debian 4.19.194-1 (2021-06-10)

Tabelle 5.2: Hardware der Testbed-Knoten.

5.5.2 Datenströme

Grundsätzlich ergeben sich für die Experimente zwei unterschiedliche Datenströme. Der Datenstrom des Typ 1 enthält ausschließlich die haptischen Daten und das dazugehörige Feedback. Der Datenstrom des Typ 2 enthält zusätzlich Audio- und Videodaten. Die mit *iperf2* generierten Datenströme werden in der Tabelle 5.3 dargestellt. Je Datentyp wird eine eigene *iperf2*-Instanz genutzt. Alle Daten werden von einem Startknoten zu einem Zielknoten gesendet (alle Daten in eine Richtung!).

Datentyp	Paketgröße	Datenrate	Datenstrom 1 (D1)	Datenstrom 2 (D1)
Haptisches Feedback	80 B	320 kb/s	✓	✓
Haptisch	48 B	192 kb/s	✓	✓
Video	1,5 kB	1 Mb/s	✗	✓
Audio	50 B	256 kb/s	✗	✓

Tabelle 5.3: Zuordnung der Datentypen zu den zwei unterschiedlichen Arten von Datenströmen.

5.5.3 Aufbau des Netzwerks

Alle für die Experimente genutzten Testbed-Knoten sind über ein *IEEE 802.11s* vermaschtes Netzwerk (Kanal 44, 20Mhz breit) untereinander verbunden. Um die Experimente wiederholbar zu gestalten, werden die Routen zwischen den Knoten fest konfiguriert (siehe Kapitel 4). Es wurden die Testbed-Knoten bewusst so ausgewählt, dass miteinander kommunizierende Knoten eine gute Signalstärke besitzen.

Bei den Experimenten mit nur einem Hop betrifft dies nur zwei Knoten. Bei den Experimenten mit zwei Hops existieren drei Knoten. Angenommen, dass es die Knoten A, B und

ID	Datenstrom	Haptisch (Feedback)	Audio & Video	Hops	EDCA Parameter
1	D1	localhost	✗	0	✗
2	D2	localhost	localhost	0	✗
3	D1	AC_BE	✗	1	Standard
4	D2	AC_BE	AC_BE	1	Standard
5	D1	AC_VO	✗	1	Standard
6	D2	AC_VO	AC_VO	1	Standard
7	D1	AC_TC	✗	1	TKF
8	D2	AC_TC	AC_VO	1	TKF
9	D1	AC_BE	✗	2	Standard
10	D2	AC_BE	AC_BE	2	Standard
11	D1	AC_VO	✗	2	Standard
12	D2	AC_VO	AC_VO	2	Standard
13	D1	AC_TC	✗	2	TKF
14	D2	AC_TC	AC_VO	2	TKF

Tabelle 5.4: Experimentübersicht. Die Einträge bei den Datentypen stehen für die verwendeten Zugriffskategorien.

C gibt und der Knoten C die Rolle des Routers einnimmt, bedeutet dies folgendes: Knoten C empfängt die Pakete von A und B. Knoten C leitet die Pakete von A weiter an B und die Pakete von B weiter an A.

5.5.4 Auflistung der Experimente

Alle Experimente werden für 10 Minuten durchgeführt. Eine finale Auflistung der Experimente findet sich in Tabelle 5.4. Die EDCA Parameter finden sich in Tabelle 4.2.

KAPITEL 6

Evaluierung

In diesem Kapitel werden die Ergebnisse der Experimente dargestellt und ausgewertet. Aus einer tabellarischen Darstellung einer Reihe an Ergebnissen, können erste Erkenntnisse gezogen werden. Die genauere Betrachtung erfolgt im Verlauf des Kapitels anhand von Histogrammen, sowie Box- und Whiskerplots, welche mit verschiedenen Python-Tools generiert werden. Hauptsächlich werden die mit dem Python-Tool Seaborn [1] erstellten Box-Whisker-Plots genutzt. Die Standardkonfiguration von Seaborn, bei der die Whiskerlänge auf das 1,5-fache des Interquartilabstands beschränkt ist, wird genutzt. Alle außerhalb des Bereichs liegenden Werte sind sogenannte *Ausreißer*.

Die Tabelle 6.1 listet die zuvor vorgestellten Experimente erneut auf und ergänzt die Zeitpunkte der Durchführung und die genutzten Knoten innerhalb des Testbeds. Die Nummer entspricht der ID innerhalb des *TBMS*. Es ist ersichtlich, dass die Experimente so durchgeführt wurden, dass zwischen den Experimenten keine großen Zeitabstände liegen, um zu verhindern, dass sich die Umgebung stark verändert. Ausgenommen davon sind die Experimente, welche als Referenz dienen (1 und 2). Bei diesen Experimenten besteht keine Verbindung zu anderen Knoten, weshalb eine Veränderung der Umgebung keinen Einfluss auf die Ergebnisse hat.

Die darauf folgende Tabelle 6.2 gibt bereits einen ersten Überblick über die Latenzen, welche während der Experimente gemessen wurden. Die Durchschnittswerte, die Standardabweichung, der Jitter und die Konfidenzintervalle der Durchschnittswerte sind dargestellt. Zusätzlich wird angegeben, wie viel Prozent der ermittelten Latenzen über einer Millisekunde liegen und dementsprechend nicht mehr der Anforderung an das *Taktile Internet* genügen.

6.1 Referenzmessung

Um Referenzwerte zur Betrachtung der Ergebnisse zu haben, wurden die Experimente zu Beginn auf einem Testbed-Knoten lokal und ohne eine Netzwerkverbindung durchgeführt. Die gemessenen durchschnittlichen Latenzen bewegen sich für alle Datentypen in einem

[1] `https://seaborn.pydata.org/generated/seaborn.boxplot.html`, Abgerufen am 15.10.2021

	Testbed-Knoten			
Experiment ID	Tx	Rx	Router	Start (CEST)
1				2021-11-11 11:32:36
2				2021-11-11 13:25:59
3	5	7		2021-10-21 08:36:30
4	5	7		2021-10-21 08:09:44
5	5	7		2021-10-20 20:25:23
6	5	7		2021-10-20 20:51:05
7	5	7		2021-10-20 22:09:34
8	5	7		2021-10-20 21:17:01
9	8	7	5	2021-10-20 22:36:07
10	8	7	5	2021-10-20 21:43:22
11	8	7	5	2021-10-20 23:02:30
12	8	7	5	2021-10-20 23:29:23
13	8	7	5	2021-10-20 19:59:19
14	8	7	5	2021-10-22 09:20:19

Tabelle 6.1: Startzeit und Testbed-Knoten der durchgeführten Experimente. Bei den Experimenten mit zwei Hops ist der dazwischenliegende Mesh-Knoten als Router angegeben.

sehr niedrigen Bereich (0,0213 ms - 0,0234 ms). Diese Latenzen entstehen allein durch die Verarbeitung der Datenpakete durch den Kernel und *iperf2*. Bei der Betrachtung der dazugehörigen Box- und Whiskerplots in Abbildung 6.1 wird ersichtlich, dass nicht nur die Durchschnittswerte sehr nah zusammenliegen, sondern dass auch die gemessenen Latenzen innerhalb eines sehr kleinen Bereichs verteilt liegen. Nichtsdestotrotz kommt es zu gelegentlichen Ausreißern, die die Marke von 1 ms überschreiten. Die Schwierigkeit in der echten Welt tatsächlich die *1-ms-Challenge* zu schaffen, wird deutlich.

Die geringere Anzahl und niedrigere Abweichung der Ausreißer bei den Videodaten erklärt sich damit, dass die Sendefrequenz geringer ist. Die Experimente wurden alle in dem gleichen Zeitraum ausgeführt, daher ergeben sich mit der geringeren Sendefrequenz auch insgesamt weniger Datenpunkte für die Videodaten.

6.2 Betrachtung der Mittelwerte

In einem ersten Schritt werden die alleinigen Durchschnittswerte der Experimente betrachtet, in welchen ausschließlich haptische Daten übertragen werden (Experimente mit einem Hop: 3,5,7. Experimente mit zwei Hops: 9,11,13). Die erste Erwartung bei diesen Experimenten ist, dass die durchschnittliche Latenz geringer wird, je höher die Priorität einer Zugriffskategorie ist. Bei der Nutzung der Kategorie *Taktiles Internet* der Taktilen Koor-

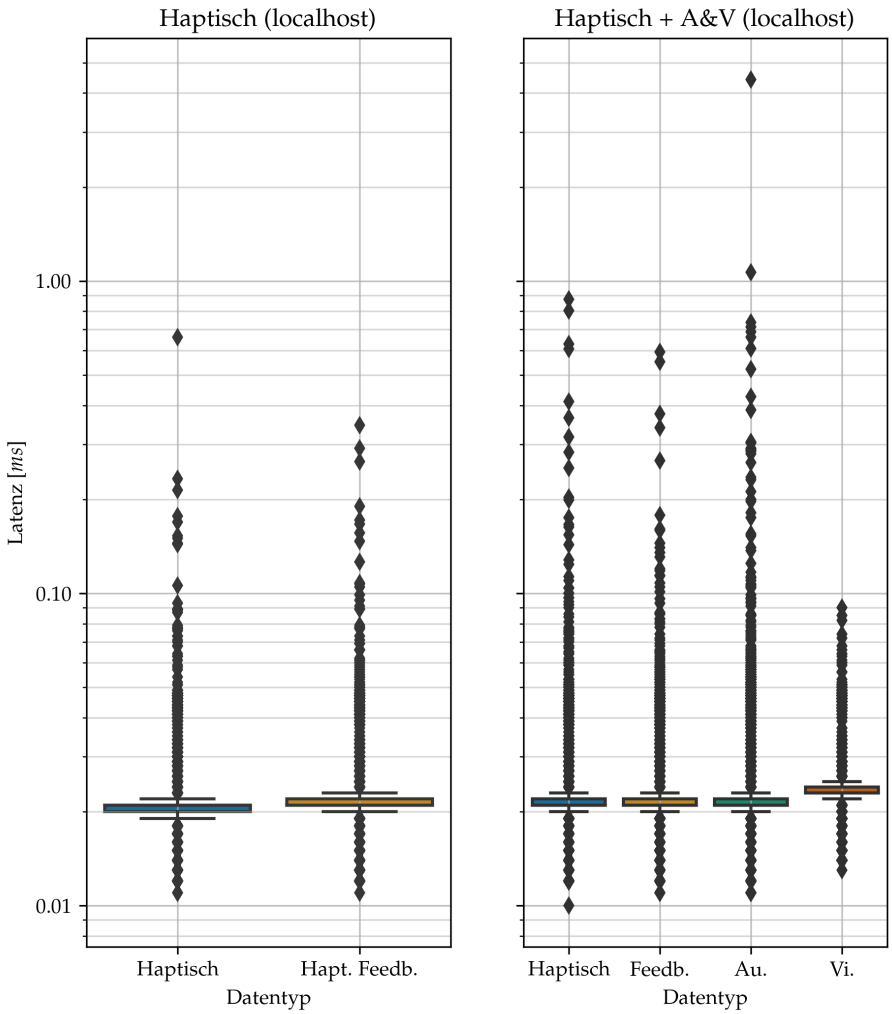


Abbildung 6.1: Box- und Whiskerplot der Referenzexperimente. (Eigene Abbildung, Vektorgrafik in Git)

dinierungsfunktion wird eine besonders niedrige Latenz erwartet, da die *AIFS*-Zeit nur die Länge einer Slotzeit besitzt.

In den Ergebnissen zeigt sich jedoch die folgende Reihenfolge für die Zugriffskategorien anhand der Durchschnittswerte der Latenzen:

- *Haptisch*
  1 Hop: 0,243 37 ms (VO) < 0,250 562 ms (TC) < 0,263 078 ms (BE)
  2 Hops: 0,462 849 ms (VO) < 0,466 122 ms (TC) < 0,813 799 ms (BE)
- Haptisches Feedback:
  1 Hop: 0,247 35 ms (VO) < 0,247 708 ms (BE) < 0,248 944 ms (TC)
  2 Hops: 0,472 106 ms (TC) < 0,474 427 ms (VO) < 0,569 378 ms (BE)

Auffallend ist, dass bei der reinen Betrachtung der Durchschnittswerte bis auf eine Ausnahme immer die Kategorie für Sprachdaten die geringsten Latenzen ergeben hat. Bei den Experimenten mit nur einem Hop liegen die Latenzen alle sehr nah beieinander. Bei zwei

Hops liegen nur die Latenzen der Zugriffskategorien für Sprache und das *Taktile Internet* nah zusammen. Die Kategorie *Best Effort* liefert bei zwei Hops deutlich höhere Latenzen.

Daraus ergibt sich die Frage, weshalb die Latenzen nicht bei der Nutzung der Zugriffskategorie für das *Taktile Internet* am geringsten sind. Erklären lässt sich das damit, dass die *TXOP* bei der Zugriffskategorie für Sprache größer ist. Dies erlaubt die Übertragung von Datenpaketen für einen längeren Zeitraum am Stück. Bei diesen ersten Experimenten werden alle Daten jeweils mit der gleichen Zugriffskategorie versendet. Dadurch ergeben sich durch die größere *TXOP* weniger Wartezeiten durch den Backoff-Vorgang, was zu geringeren Latenzen führt. Gleichzeitig führt dieses Vorgehen allerdings zu einem größeren Jitter, da der Versand der Pakete unregelmäßiger stattfindet. Eine weitere Begründung dafür, dass in dieser Betrachtung über die Durchschnittswerte kein Vorteil für die *Taktile Koordinierungsfunktion* zu erkennen ist, liegt in der geringen Auslastung des Funkkanals. Dadurch kommt es generell zu wenigen Wartezeiten. Die nur reduzierte *AIFS* hat daher kaum einen Einfluss, da eine Slotzeit innerhalb von *IEEE 802.11e* lediglich 9 Mikrosekunden beträgt. Genauer betrachtet werden die hier beobachteten Effekte in den Histogrammen und den Box- und Whiskerplots.

In einem nächsten Schritt folgt diese Betrachtung der Durchschnittswerte der Latenzen für die Experimente bei denen zusätzlich Audio- und Videodatenstreams übertragen werden (Experimente mit einem Hop: 4,6,8. Experimente mit zwei Hops: 10,12,14). Wichtig ist, dass das Label *TC* an den Daten für Audio und Video in diesem Fall bedeutet, dass die haptischen Daten über die entsprechende Zugriffskategorie übertragen worden sind. Für die Audio- und Videodaten wurden nach wie vor die *Voice*-Kategorie genutzt. Bei diesen Experimenten zeigt sich das folgende Bild:

- *Haptisch*
  1 Hop: 0,250 137 ms (TC) < 0,282 786 ms (VO) < 0,349 431 ms (BE)
  2 Hops: 1,279 068 ms (TC) < 1,394 91 ms (VO) < 1,8648 ms (BE)
- Haptisches Feedback:
  1 Hop: 0,270 228 ms (BE) < 0,282 282 ms (TC) < 0,330 809 ms (VO)
  2 Hops: 1,351 822 ms (TC) < 1,477 418 ms (VO) < 1,558 588 ms (BE)
- Audio:
  1 Hop: 0,278 139 ms (TC) < 0,282 813 ms (VO) < 0,295 47 ms (BE)
  2 Hops: 1,460 646 ms (TC) < 1,489 936 ms (VO) < 1,744 808 ms (BE)
- Video:
  1 Hop: 0,654 44 ms (VO) < 0,702 86 ms (BE) < 0,829 97 ms (TC)
  2 Hops: 2,506 967 ms (VO) < 2,616 615 ms (TC) < 2,973 781 ms (BE)

Bei den Experimenten mit zusätzlichem Datenverkehr zeigt sich bei einer Betrachtung der durchschnittlichen Latenz, dass die haptischen Daten von der neu geschaffenen Zugriffskategorie profitieren. Besonders deutlich wird dies bei einer hohen Auslastung des Funkkanals (2 Hops). Zusätzlich profitieren teilweise sogar die weiteren Daten, da durch eine kürzere *AIFS* eine höhere Auslastung des Funkkanals erreicht werden kann. Eine genauere Betrachtung der Kanalauslastung erfolgt später in diesem Kapitel.

Wie bereits zuvor erwähnt, wird bei einer Untersuchung des Jitters der Vorteil der *Taktilen Koordinierungsfunktion* gegenüber der Nutzung der Sprachkategorie unter der Standardparametrisierung besonders deutlich. Dieser Effekt ergibt sich allein durch die Reduzierung

der *TXOP*, da ein einzelner Knoten den Kanal nicht für eine so lange Zeit blockieren kann. Die Betrachtung der Kategorie *Best Effort* erfolgt in diesem Fall nicht, da dort die Möglichkeit einer *TXOP* nicht genutzt wird. So ergibt sich unter dieser Betrachtung die folgende Reihenfolge:

- *Haptisch*
  1 Hop: 0,005 409 ms (TC) < 0,006 99 ms (VO)
  2 Hops: 0,010 022 ms (TC) < 0,014 305 ms (VO)
- Haptisches Feedback:
  1 Hop: 0,005 409 ms (TC) < 0,028 747 ms (VO)
  2 Hops: 0,012 988 ms (TC) < 0,040 863 ms (VO)

Mit Audio- und Videodaten:

- *Haptisch*
  1 Hop: 0,016 623 ms (TC) < 0,067 45 ms (VO)
  2 Hops: 0,363 428 ms (TC) < 1,043 341 ms (VO)
- Haptisches Feedback:
  1 Hop: 0,0513 ms (TC) < 0,107 294 ms (VO)
  2 Hops: 0,395 593 ms (VO) < 1,151 078 ms (TC)
- Audio:
  1 Hop: 0,058 086 ms (VO) < 0,078 248 ms (TC)
  2 Hops: 0,120 218 ms (TC) < 0,136 522 ms (VO)
- Video:
  1 Hop: 0,038 628 ms (TC) < 0,057 015 ms (VO)
  2 Hops: 0,157 834 ms (VO) < 0,358 332 ms (TC)

Es zeigt sich insbesondere bei den Experimenten ohne Audio- und Videodaten, dass der Jitter tatsächlich geringer ist. Bei den Experimenten mit zusätzlichen Audio- und Videodaten ist dies nicht ganz eindeutig, da für die Audio- und Videodaten nach wie vor die Zugriffskategorie für Sprachdaten und damit die größere *TXOP* genutzt wird. Trotzdem zeigt sich gerade bei einer geringeren Kanalauslastung auch hier ein geringerer Jitter bei der Taktilen Zugriffskategorie.

Eine weitere Metrik, die in der Tabelle betrachtet wird, ist die Anzahl an gemessenen Latenzen je Datentyp und Experiment, die über der 1 ms-Marke liegen. Dies lässt erste Schlüsse auf die Verteilung der Latenzen zu. Bei der Betrachtung fällt auf, dass bei den Experimenten, bei denen zusätzliche Daten (Audio & Video) übertragen werden, die Anzahl an gemessenen Latenzen über 1 ms deutlich niedriger ist, wenn die *Taktile Koordinierungsfunktion* genutzt wird. Dies trifft zumindest für die haptischen Daten zu, welche mit der neu geschaffenen Zugriffskategorie versendet werden.

6.3 Genauere Betrachtung der Latenzen mithilfe von Histogrammen sowie Box- und Whiskerplots

Da die reine Betrachtung der Durchschnittswerte der Latenzen zu keinen klaren Ergebnissen führt, folgt in diesem Abschnitt die Betrachtung der Verteilung der gemessenen Latenzen.

Auch hier wird mit den Experimenten ohne zusätzliche Datenströme für Audio- und Video begonnen. Die Box- und Whiskerplots sind nicht als Vektorgrafik eingebunden, da sie aufgrund der hohen Anzahl an Ausreißern zu langsam dargestellt werden und sie die Größe der entstehenden PDF ungünstig beeinflussen. Die Vektorgrafiken finden sich in dem zur Masterarbeit gehörenden Git-Repository.

Abbildung 6.2 zeigt die Box- und Whiskerplots für die Experimente ohne zusätzliche Audio- und Videodatenstreams. Es werden nur die haptischen Daten übertragen. Auf den ersten Blick ist erneut ersichtlich, dass die Latenzen unter der Nutzung der Zugriffskategorien für Sprache und das *Taktile Internet* im Gegensatz zu der Zugriffskategorie *Best Effort* häufig geringer sind. Gleichzeitig wird aber auch deutlich, dass die Interquartilabstände geringer sind, also die Latenzen nicht so weit gestreut sind, wie bei der Zugriffskategorie *Best Effort*. Die Betrachtung des dazugehören Häufigkeitshistogramms in Abbildung 6.3 bestätigt dies, welches besonders bei den Experimenten mit 2 Hops deutlich wird. Ein signifikanter Unterschied zwischen der Nutzung der Warteschlange für Sprachdaten und der Warteschlange für das *Taktile Internet* wird nicht ersichtlich, sondern ergibt sich durch die geringe Auslastung des Kanals.

Bei den Experimenten mit den zusätzlichen Audio- und Videodatenströmen unterschieden sich die Ergebnisse ein wenig. Im Vergleich zu der Zugriffskategorie für Sprache kann mit der Nutzung der Zugriffskategorie für das *Taktile Internet* für die haptischen Daten ein positiver Effekt erkannt werden. Der Median liegt bei den haptischen Daten dort deutlich erkennbar niedriger. Auch die Grenzen für das untere und das obere Quartil sind niedriger. Schon bei zwei Hops liegt der Median der Latenz dennoch bei über 1 ms. Das untere Quartil und damit 25% der Daten befinden sich jedoch deutlich unter einer Latenz von 1 ms. In den Häufigkeitsistogrammen in Abbildung 6.5 lässt sich ebenfalls erkennen, dass der Anteil an geringen Latenzen unter der Nutzung der Taktilen Koordinierungsfunktion für alle Datentypen größer ist, als bei der Zugriffskategorie *Voice*. Auch das lässt sich damit erklären, dass mit einer geringeren *AIFS* eine insgesamg größere Auslastung des Funkkanals erreicht werden kann.

6.4 Auslastung des Funkkanals

In Abbildung 6.6 sind zwei Balkendiagramme zu erkennen. Das linke Diagramm gibt die je Testbed-Knoten erfasste Auslastung des Funkkanals in Prozent an. Dies ist der Anteil der Zeit, in welcher der Kanal als belegt erkannt wird, an der Gesamtzeit des Experiments. Das rechte Diagramm zeigt den prozentualen Anteil in der ein Testbed-Knoten sendet und empfängt an der Zeit in der der Kanal während des Experiments belegt ist.

Es wird deutlich, dass die Auslastung des Kanals bei den Experimenten mit zwei Hops (9-14) und dabei gleichen Daten etwa doppelt so groß ist im Vergleich zu den Experimenten mit nur einem Hop. Eine Kleine Erinnerung: Experimente, welche eine gerade ID besitzen übertragen zusätzlich zu den haptischen Daten auch Audio- und Videodaten. Die erhöhte Auslastung lässt sich damit erklären, dass der Router die Daten in beide Richtungen weiterleitet und damit das Datenaufkommen verdoppelt. Dass auch die Latenzen der Server-

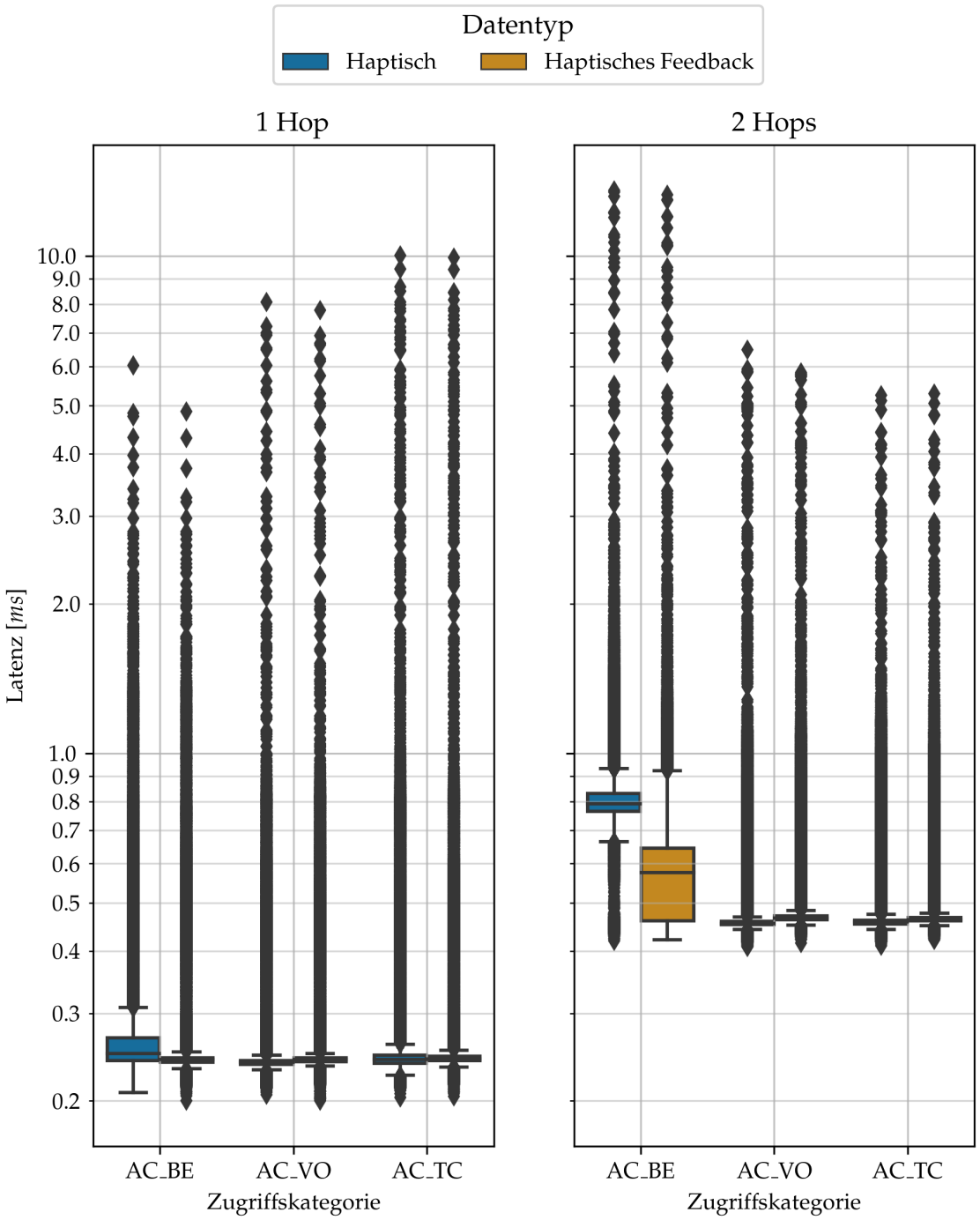


Abbildung 6.2: Box- und Whiskerplot der Experimente ohne zusätzliche Audio- und Videodatenstreams. (Eigene Abbildung, Vektorgrafik in Git)

und Client-Knoten erhöht sind lässt sich damit erklären, dass sich alle Testbed-Knoten innerhalb des Experiments gegenseitig *"hören"* können.

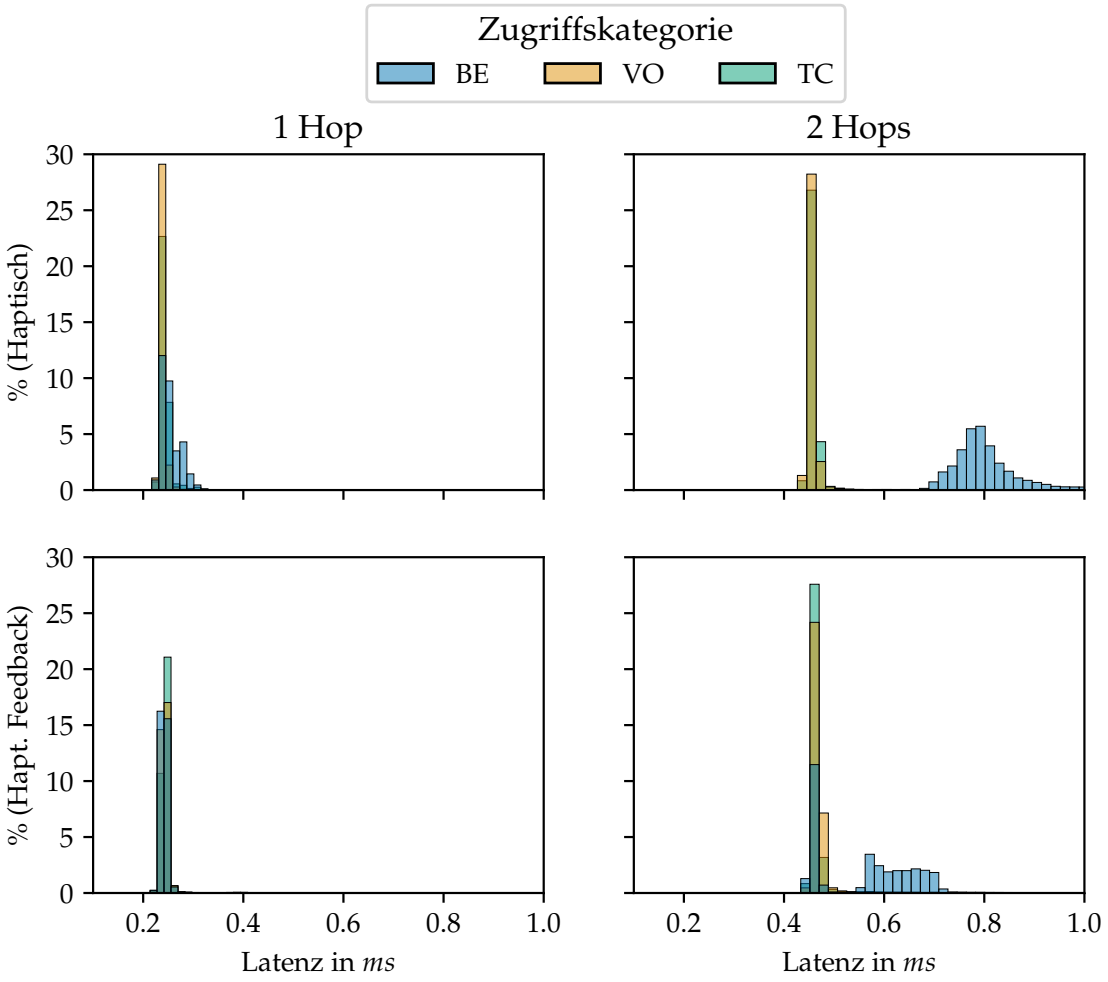


Abbildung 6.3: Häufigkeitshistogramme für die Latenzen bei den Experimenten ohne zusätzliche Datenströme. (Eigene Abbildung)

Eine weitere Beobachtung der Experimente ist, dass die Kanalauslastung bei dem Experiment 14 am höchsten ist. Das lässt sich damit erklären, dass die *AIFS*-Zeit für die haptischen Daten halbiert ist. Dadurch kann der Kanal insgesamt stärker ausgelastet werden, da weniger Zeit für den Backoff-Vorgang genutzt wird. Bereits in den vorherigen Experimenten hat sich gezeigt, dass bei den Experimenten mit zusätzlichen Audio- und Videodaten und 2 Hops die Latenzen von unter 1 ms regelmäßig nicht eingehalten werden können. Unter der Betrachtung der Kanalauslastung von fast 30 % ist dies jedoch zu erwarten [14].

In dem rechten Diagramm lässt sich erkennen, dass der Router-Knoten während der Experimente mit nur einem Hop nahezu keine Daten sendet und empfängt. Dieses Verhalten war zu erwarten, da keine von *iperf2* generierten Daten darüber geleitet werden. Bei den Experimenten mit zwei Hops wird in etwa die Summe der Daten des Server- und Clientaufkommens verarbeitet.

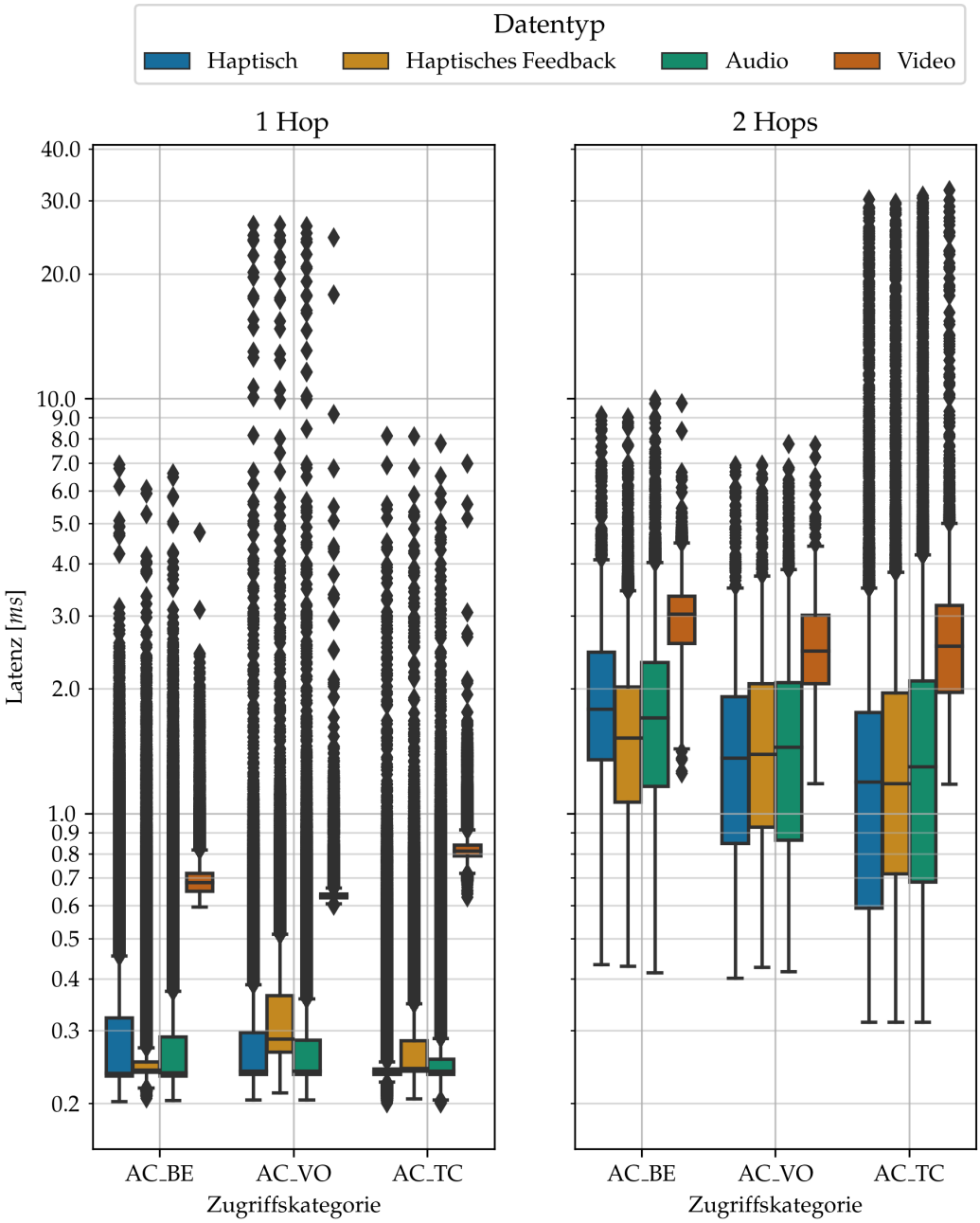


Abbildung 6.4: Box- und Whiskerplot der Experimente mit zusätzlichen Audio- und Videodatenstreams. (Eigene Abbildung, Vektorgrafik in Git)

6.5 Paketverlust

Bei der Betrachtung der Paketverluste wird klar, dass bei keinem der Experimente mit kompletten Daten (inklusive Audio- und Videodaten) eine ausreichende Zuverlässigkeit gemäß der in Tabelle 5.1 festgelegten Anforderungen erreicht werden konnte. Bei mindestens einem der Datentypen wird immer die maximale Paketverlustrate überstiegen.

Weiterhin ist zu betonen, dass der Experimentaufbau nicht auf die Bestimmung des Paketverlustes abzielt. Um dies zu erreichen hätten die Experimente länger durchgeführt, häufiger und zu unterschiedlichen Zeitpunkten wiederholt werden müssen, um sicherzustellen, dass externe Einflüsse auf die Übertragung vollständig abgedeckt werden.

6.6 Forschungsfragen

Zusammenfassend für dieses Kapitel werden die hier zu beantwortenden Forschungsfragen betrachtet. Bei der ersten Forschungsfrage gilt es zu beantworten, ob die positiven Ergebnisse aus der Simulation auch in der realen Welt wiederholt werden können. Diese Frage kann mit einem *Ja* mit einer leichten Einschränkung beantwortet werden. Es lässt sich erkennen, dass mit der Taktilen Koordinierungsfunktion unter einer hohen Auslastung des Funkkanals die Wahrscheinlichkeit steigt, dass die Latenz eines einzelnen Pakets gering ausfällt. Allerdings wird die Grenze von 1 ms bereits bei zwei Hops erreicht, sobald zusätzlich die Audio- und Videodaten für einen Anwendungsfall übertragen werden sollen. Auch wird klar, dass Umwelteinflüsse eine große Rolle spielen. Es kann bei der Übertragung über nicht lizenzierte und frei verfügbare Frequenzen nie garantiert werden, dass eine bestimmte Latenz oder Paketverlustrate eingehalten wird. Nichtsdestotrotz kann gesagt werden, dass die *Taktile Koordinierungsfunktion* für die Priorisierung von haptischen Daten geeignet ist. Ein weiterer positiver Aspekt der bestätigt werden konnte ist, dass der Jitter unter der Nutzung der Taktilen Koordinierungsfunktion bei geringer Kanalauslastung niedriger wird. Das Medium wird dort gerechter zwischen den Teilnehmern aufgeteilt.

Die zweite hier zu beantwortende Forschungsfrage beschäftigt sich mit dem Einfluss der Taktilen Koordinierungsfunktion auf weitere übertragene Daten. Für diese Experimente wurden die Audio- und Videostreams als weitere Daten für das *Taktile Internet* über die Zugriffskategorie für Sprachdaten übertragen. Daher sind die *Contention Windows* identisch zu denen der Zugriffskategorie für haptische Daten. Dementsprechend konnte nicht gezeigt werden, dass die *Taktile Koordinierungsfunktion* hier zu einer Benachteiligung der weiteren Daten führt. Im Gegenteil konnte gezeigt werden, dass eine geringere *AIFS* zu einer höheren Kanalauslastung führt und damit auch zu geringeren Latenzen bei den weiteren Daten. Wenn die weiteren Daten über eine Zugriffskategorie mit einem größeren maximalen Contention Window gesendet werden, besteht die Möglichkeit andere Ergebnisse zu erhalten.

						Konfidenzintervall	
		Avg	Stdev	Jitter	>1 ms (%)	Anfang	Ende
1							
	Haptisch	0,021091	0,002425	0,001231	0,0	0,02108	0,021102
	Hapt. Feedb.	0,02152	0,002314	0,001068	0,0	0,021509	0,021531
2							
	Haptisch	0,021645	0,003744	0,001132	0,0	0,021627	0,021662
	Hapt. Feedb.	0,021313	0,002905	0,000764	0,0	0,021299	0,021326
	audio	0,021459	0,008297	0,001038	0,000521	0,021425	0,021494
	video	0,023402	0,002336	0,003115	0,0	0,023375	0,023429
3							
	Haptisch	0,263078	0,073612	0,008772	0,197338	0,262732	0,263424
	Hapt. Feedb.	0,247708	0,056231	0,002789	0,112334	0,247443	0,247972
4							
	Haptisch	0,349431	0,224226	0,270442	1,032823	0,348376	0,350486
	Hapt. Feedb.	0,270228	0,092492	0,022913	0,227332	0,269793	0,270663
	audio	0,29547	0,144873	0,122002	0,409633	0,294867	0,296072
	video	0,702866	0,100617	0,077898	1,311948	0,701707	0,704025
5							
	Haptisch	0,24337	0,057457	0,00699	0,025	0,2431	0,243641
	Hapt. Feedb.	0,24735	0,057896	0,028747	0,030667	0,247078	0,247622
6							
	Haptisch	0,282786	0,165068	0,06745	0,063666	0,28201	0,283562
	Hapt. Feedb.	0,330809	0,169515	0,107294	0,094417	0,330011	0,331606
	audio	0,282813	0,168599	0,058086	0,066927	0,282112	0,283514
	video	0,654441	0,162255	0,057015	0,50798	0,652572	0,65631
7							
	Haptisch	0,250562	0,09793	0,005409	0,064001	0,250102	0,251023
	Hapt. Feedb.	0,248944	0,090726	0,005476	0,052334	0,248517	0,24937
8							
	Haptisch	0,250137	0,063162	0,016623	0,035666	0,24984	0,250434
	Hapt. Feedb.	0,282282	0,0901	0,0513	0,049667	0,281858	0,282705
	audio	0,278139	0,120679	0,078248	0,105041	0,277637	0,27864
	video	0,82997	0,090123	0,038628	4,634093	0,828931	0,831008
9							
	Haptisch	0,813799	0,13222	0,017473	4,537303	0,813178	0,814421
	Hapt. Feedb.	0,569378	0,142061	0,103022	0,849202	0,56871	0,570046
10							
	Haptisch	1,8648	0,685028	0,306033	89,390071	1,861579	1,868022
	Hapt. Feedb.	1,558588	0,666658	0,703635	77,312485	1,555452	1,561723
	audio	1,744808	0,752795	0,981412	81,19515	1,741678	1,747937
	video	2,973781	0,548359	0,515244	100,0	2,967462	2,980101
11							
	Haptisch	0,462849	0,072631	0,014305	0,162999	0,462508	0,463191
	Hapt. Feedb.	0,474427	0,073064	0,040863	0,177665	0,474084	0,474771
12							
	Haptisch	1,39491	0,656057	1,043341	68,816541	1,391825	1,397996
	Hapt. Feedb.	1,477418	0,661155	0,395593	70,885515	1,474309	1,480528
	audio	1,489936	0,702707	0,136522	69,613366	1,487013	1,492859
	video	2,506967	0,563383	0,157834	100,0	2,500477	2,513457
13							
	Haptisch	0,466122	0,062837	0,010022	0,204332	0,465826	0,466417
	Hapt. Feedb.	0,472106	0,059528	0,012988	0,154001	0,471826	0,472386
14							
	Haptisch	1,279068	0,809715	0,363428	56,990287	1,275261	1,282876
	Hapt. Feedb.	1,351822	0,831932	1,151078	59,508906	1,347907	1,355736
	audio	1,460646	0,990351	0,120218	61,941865	1,45653	1,464763
	video	2,616615	0,94014	0,358332	100,0	2,605784	2,627446

Tabelle 6.2: Ergebnisse der Experimente. 99%-Konfidenzintervall (Student-t-Verteilung)

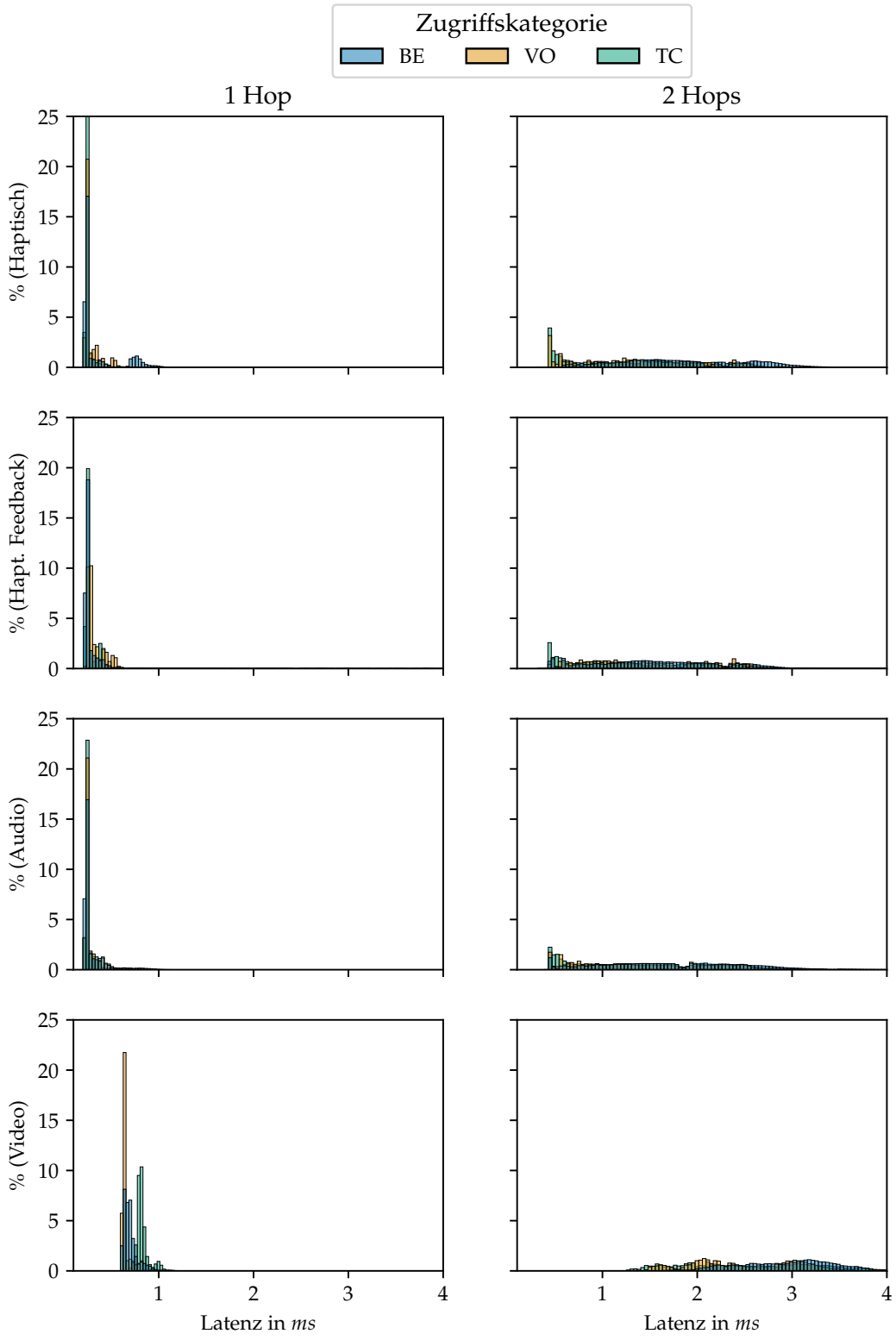


Abbildung 6.5: Häufigkeitshistogramme für die Latenzen bei den Experimenten mit zusätzlichen Datenströmen für Audio- und Videodaten. (Eigene Abbildung)

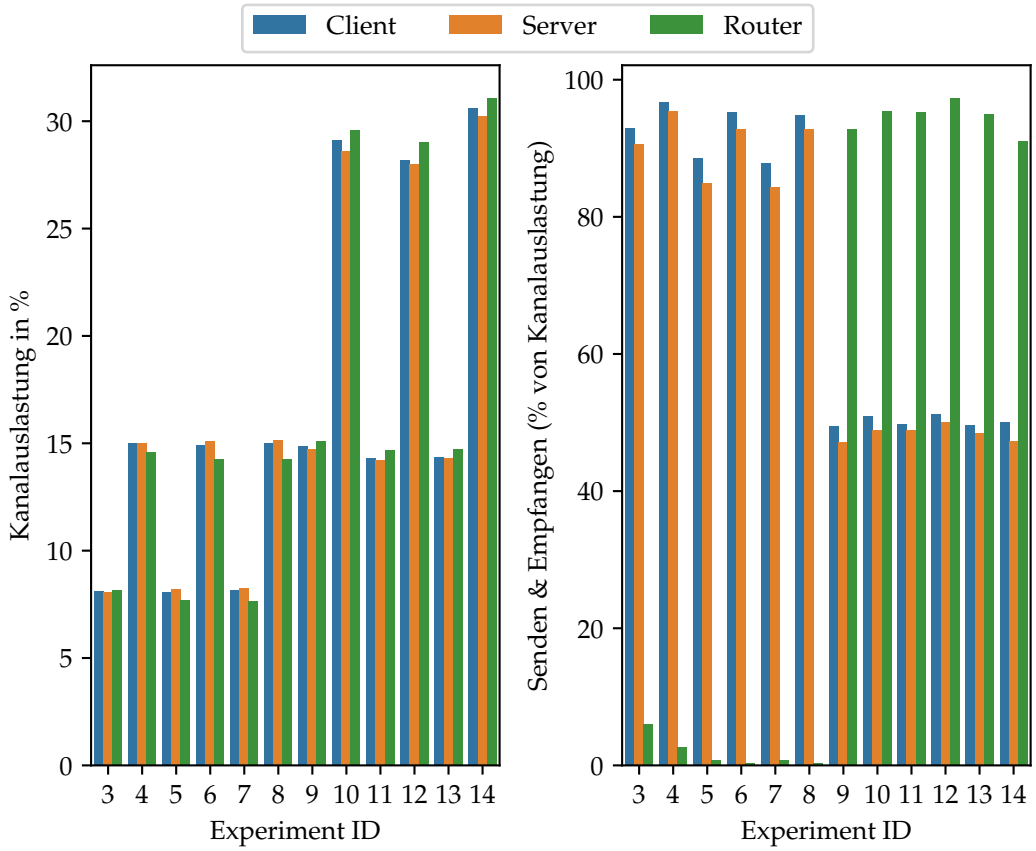


Abbildung 6.6: Auslastung des genutzten Funkkanals während der Experimente. (Eigene Abbildung)

	Paketverlust in %			
	haptic	haptic feedback	audio	video
1	0	0	–	–
2	0	0	0,013	0
3	0,003	0,001	–	–
4	0,08	0	0	0
5	0,0017	0,0023	–	–
6	0	0,089	0	0
7	0,0017	0,001	–	–
8	0	0,001	0,089	0,006
9	0	0,064	–	–
10	0	0	0	0,078
11	0	0	–	–
12	0	0,0023	0	0
13	0	0,0013	–	–
14	0	0,1	0	0,0014

Tabelle 6.3: Paketverlust

KAPITEL 7

Abschluss

Kapitel 7 gibt eine Zusammenfassung der Masterarbeit und schätzt ein, ob die Ziele erreicht wurden. Abschließend folgt ein Ausblick auf zukünftige Arbeiten, wofür während der Forschungsarbeit Ideen entstanden sind, die an diese Arbeit anknüpfen können.

7.1 Zusammenfassung

Das Ziel der Masterarbeit war die Implementierung der *Taktilen Koordinierungsfunktion* in eine reale Umgebung. Die Implementierung ermöglicht das priorisierte Versenden von haptischen Daten über den *IEEE 802.11e* Standard mit einem Linux-Betriebssystem. Dabei wurde auf die Probleme bei der Umsetzung in einem realen System im Gegensatz zu einer Simulation eingegangen. Mit der Implementierung und einem geeigneten Testaufbau wurden Experimente durchgeführt, um die in Kapitel 1 eingeführten Forschungsfragen zu beantworten:

1. Bestätigt sich das Ergebnis von Engelhardt et al. [1], dass die *Taktile Koordinierungsfunktion* als Ergänzung zu *IEEE 802.11* für das *Taktile Internet* geeignet ist, in einem Testbed?
2. Welche Auswirkungen hat die Schaffung einer erhöhten Priorität für das *Taktile Internet* für die weitere Kommunikation in dem Netzwerk?
3. Welche Hürden bestehen bei der Implementierung einer erhöhten Priorität für Datenpakete des *Taktilen Internets* im Rahmen von IEEE 802.11?

Dafür wurde ein Konzept zur Testdurchführung erarbeitet und anschließend umgesetzt. Hierfür wurde das *MIoT-Testbed* der Universität Magdeburg genutzt. Dabei konnten alle drei Forschungsfragen beantwortet werden. Die ersten beiden Forschungsfragen wurden mit Kapitel 5 und 6 beantwortet. Die dritte Forschungsfrage in Kapitel 4. Die Antworten auf die drei Forschungsfragen werden nachfolgend zusammenfassend dargestellt.

Die erste Frage kann mit einem *"Ja"* beantwortet werden, wobei die Ergebnisse nicht so deutlich wie in der Simulation sind. Dies lässt sich jedoch durch einen leicht veränderten Experimentaufbau erklären. In der Simulation wurden die zusätzlichen Daten über die Zugriffskategorie für Hintergrunddaten gesendet. Dies entspricht der niedrigsten Priorität. In

den hier durchgeführten Experimenten wurden die zusätzlichen Daten mit den Kategorien *Best Effort* und *Voice* versendet, was die Realität eher widerspiegelt. Durch die so stärkere Konkurrenz bezüglich des Funkkanals sind die Ergebnisse nicht so eindeutig, wie bei der Simulation.

Bei der Untersuchung der Forschungsfrage zwei hat sich gezeigt, dass die neu geschaffene erhöhte Priorität keinen negativen Effekt auf die weiteren Daten hat. Da die Backoff-Zeit verkürzt wird, konnte die weitere Kommunikation teilweise sogar davon profitieren, da die Kanalauslastung insgesamt steigt. Jedoch existiert hier die Einschränkung, dass in den Experimenten die zusätzlich übertragenen Daten auch einer relativ hohen Zugriffskategorie zugeordnet waren, welche ebenfalls ein geringes maximales *Contention Window* besitzt.

Forschungsfrage Drei zielt auf Probleme ab, die sich ergeben, wenn die Taktile Koordinierungsfunktion auf einer echten Hardware und in dem Kernel eines Linux-Betriebssystems integriert wird. Dabei hat sich gezeigt, dass die grundlegende Funktionalität bereits innerhalb des Kernels vorhanden ist. Allerdings mussten Anpassungen getroffen werden, damit die Parameter unter der Nutzung des *Mesh*-Modus angepasst werden können. Auch fehlte ein Tool, um die Parameter über die *nl80211*-Schnittstelle aus dem Userspace zu setzen. Das größte Problem zeigte sich bei der Hardware, da es dort aufgrund von Firmware-Einschränkungen häufig nicht möglich ist, eine fünfte Zugriffskategorie zu ergänzen. Aus diesem Grund wurde für die Experimente eine der vier bestehenden Kategorien umprogrammiert.

7.2 Zukünftige Arbeiten

Während der Bearbeitung des Themas sind immer wieder neue Ideen gewachsen, wie die Latenzen für das *Taktile Internet* weiter verbessert und welche Experimente zusätzlich hätten durchgeführt werden können.

Aufgrund der begrenzten Zeit zur Bearbeitung der Masterarbeit, musste eine Auswahl an Experimenten getroffen werden. Im Falle von weiteren Forschungsaktivitäten, sollten die Experimente zusätzlich mit unterschiedlichen Kombinationen aus Testbed-Knoten und Hops wiederholt werden, um Störungen durch die Umwelt auszugleichen. Zudem unterliegt die Implementierung von *EDCA* zu großen Teilen den Herstellern der Hardware und auch die Schnittstelle zum Linux-Kernel ist unterschiedlich gestaltet. Daher ist es für weitere Experimente ebenso interessant zu prüfen, inwieweit sich die Ergebnisse mit anderer Hardware wiederholen lassen.

Ein weiterer Ansatz für zukünftige Forschungsvorhaben liegt darin, dass die *Taktile Koordinierungsfunktion* weiter verbessert wird. Dies könnte zum Beispiel bedeuten, dass nicht nur eine neue Zugriffskategorie erstellt wird, sondern, dass die Parameter der bestehenden Kategorien ebenfalls angepasst werden. Hierfür könnte eine Reihe von Experimenten genutzt werden, um die Parameter so zu gestalten, dass einerseits eine gute maximale Auslastung des Kanals erreicht wird und gleichzeitig die Latenzen für taktile Daten gering gehalten werden.

Sehr interessant ist auch die bereits zu Beginn vorgestellte Arbeit von Backhaus et al. [12]. Dort wird die *IEEE 802.11* Implementierung in den Userspace verschoben, wodurch geringere Latenzen und ein höherer Durchsatz erreicht werden können. Gleichzeitig lässt sich

eine solche Implementierung für Experimente leichter anpassen. Daher wäre eine Kombination der hier durchgeführten Experimente in weiteren Arbeiten mit der dort vorgestellten Userspace-Implementierung sinnvoll.

Literatur

[1] Frank Engelhardt, Chenke Rong und Mesut Güneş. „Towards Tactile Wireless Multi-Hop Networks : The Tactile Coordination Function as EDCA Supplement“. In: *2019 Wireless Telecommunications Symposium (WTS)*. 2019, S. 1–7.

[2] Tanweer Alam. „5G-Enabled Tactile Internet for smart cities: vision, recent developments, and challenges“. In: *Jurnal Informatika* 13 (Juli 2019), S. 1–10.

[3] Gerhard Fettweis u. a. *The Tactile Internet ITU-T Technology Watch Report*. 2014.

[4] Martin Maier, Amin Ebrahimzadeh und Mahfuzulhoq Chowdhury. „The Tactile Internet: Automation or Augmentation of the Human?“ In: *IEEE Access* 6 (Juli 2018), S. 1–1.

[5] Oliver Holland u. a. „The IEEE 1918.1 “Tactile Internet” Standards Working Group and its Standards“. In: *Proceedings of the IEEE* 107.2 (2019), S. 256–279.

[6] Duc Le, Tri Gia Nguyen und Thi Tran. „The 1-Millisecond Challenge – Tactile Internet: From Concept to Standardization“. In: *Journal of Telecommunications and the Digital Economy* 8 (Mai 2020), S. 56–93.

[7] Daniël Van Den Berg u. a. „Challenges in Haptic Communications Over the Tactile Internet“. In: *IEEE Access* 5 (2017), S. 23502–23518.

[8] H. Schulzrinne u. a. *”RTP: A Transport Protocol for Real-Time Applications”, RFC 1889, DOI 10.17487/RFC1889*. 1996.

[9] J. A. Cadzow. *Foundations of digital signal processing and data analysis New York, New York: Macmillan*. 1987.

[10] Ahmed Slalmi u. a. „How Will 5G Transform Industrial IoT: Latency and Reliability Analysis“. In: *Human Centred Intelligent Systems*. Hrsg. von Alfred Zimmermann, Robert J. Howlett und Lakhmi C. Jain. Singapore: Springer Singapore, 2021, S. 335–345.

[11] Toni Adame, Marc Carrascosa-Zamacois und Boris Bellalta. „Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-Latency WiFi 7“. In: *Sensors* 21.15 (2021). URL: https://www.mdpi.com/1424-8220/21/15/4954.

[12] Martin Backhaus u. a. „Towards a Flexible User-Space Architecture for High-Performance IEEE 802.11 Processing“. In: *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2018, S. 1–9.

[13] Johannes Martin Berg. *Re: Adding a fifth edca tx queue*. https://www.spinics.net/lists/linux-wireless/msg209950.html. [Online; Stand 08. August 2021]. 2021.

[14] Ricardo Moraes, Paulo Portugal und Francisco Vasques. „Simulation Analysis of the IEEE 802.11e EDCA Protocol for an Industrially-Relevant Real-Time Communication Scenario“. In: *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 2006, S. 202–209.

[15] Tien-Shin Ho und Kwang-Cheng Chen. „Performance analysis of IEEE 802.11 CSMA/CA medium access control protocol“. In: *Proceedings of PIMRC '96 - 7th International Symposium on Personal, Indoor, and Mobile Communications*. Bd. 2. 1996, 407–411 vol.2.

[16] Emansa Hasri Putra u. a. „Cross Layer Design of IEEE 802.11e Enhanced Distributed Channel Access Wireless Network for Telemedicine Application“. In: *Advances in Telemedicine: Technologies, Enabling Factors and Scenarios*. Aug. 2009, S. 129–133.

[17] Pablo Serrano u. a. „Optimal Configuration of 802.11e EDCA for Real-Time and Data Traffic“. In: *IEEE Transactions on Vehicular Technology* 59.5 (2010), S. 2511–2528.

[18] A. Banchs u. a. „Applications and challenges of the 802.11e EDCA mechanism: an experimental study“. In: *IEEE Network* 19.4 (2005), S. 52–58.

[19] Communication und OvGU Magdeburg Networked Systems. *MIOT-Lab*. `https://comsys.ovgu.de/MIOT_Lab.html`. [Online; Stand 26. August 2021]. 2018.

[20] Kai Kientopf, Marian Buschsieweke und Mesut Güneş. „Technical report - Designing a testbed for wireless communication research on embedded devices“. In: *18. GI/ITG KuVS FachGespräch SensorNetze, FGSN 2019 : Programm ; 19. September - 20. September* (2019), S. 41–44. URL: `http://dx.doi.org/10.25673/28428`.

[21] Bastian Blywis u. a. „Trends, Advances, and Challenges in Testbed-based Wireless Mesh Network Research“. In: *Mobile Networks and Applications* 15.3 (Juni 2010), S. 315–329. URL: `https://doi.org/10.1007/s11036-010-0227-9`.

[22] Mesut Günes, Felix Juraschek und Bastian Blywis. „An Experiment Description Language for Wireless Network Research“. In: *Journal of Internet Technology (JIT), Special Issue for Mobile Internet* 11 (Juli 2010).

[23] Johannes Martin Berg. *mac80211 overview*. `https://wireless.wiki.kernel.org/_media/en/developers/documentation/mac80211.pdf`. [Online; Stand 03. August 2021]. 2009.

[24] Username: artm. *How nl80211 library & cfg80211 work?* `https://stackoverflow.com/questions/21456235/how-nl80211-library-cfg80211-work/33031805#3303180`. [Online; Stand 03. August 2021]. 2015.

[25] Johannes Martin Berg. *About mac80211*. `http://linuxwireless.sipsolutions.net/en/developers/Documentation/mac80211/`. [Online; Stand 08. August 2021]. 2015.

[26] Rami Rosen. *Linux Wireless - Linux Kernel Networking (4) - advanced topics*. `http://www.haifux.org/lectures/206/wirelessLec.pdf`. [Online; Stand 08. August 2021]. 2009.

[27] Jan Rähm. *Der WLAN-Stack Mac80211*. `https://www.linux-community.de/ausgaben/linuxuser/2008/07/der-wlan-stack-mac80211/`. [Online; Stand 08. August 2021]. 2008.

[28] Johannes Martin Berg. *About cfg80211*. http://linuxwireless.sipsolutions.net/en/developers/Documentation/cfg80211/. [Online; Stand 08. August 2021]. 2015.

[29] Johannes Martin Berg. *About nl80211*. http://linuxwireless.sipsolutions.net/en/developers/Documentation/nl80211/. [Online; Stand 08. August 2021]. 2015.

[30] Guido Hiertz u. a. „IEEE 802.11s: the WLAN mesh standard“. In: *Wireless Communications, IEEE* 17 (März 2010), S. 104–111.

[31] Robert McMahon. *IPerf2 - A tool that measures network performance of TCP/UDP including latency.* https://sourceforge.net/projects/iperf2/. [Online; Stand 12. August 2021]. 2021.

[32] Ajay Tirumala u. a. *IPERF - Section: User Manuals (1)*. https://iperf2.sourceforge.io/iperf-manpage.html. [Online; Stand 12. August 2021]. 2021.

[33] Ajay Tirumala u. a. *Iperf 2 & Iperf 3, Comparison Table*. https://iperf2.sourceforge.io/IperfCompare.html. [Online; Stand 12. August 2021]. 2021.

[34] Juniper Networks. *Wi-Fi Multimedia Configuration*. http://sup.xenya.si/sup/info/Juniper/ScreenOS_5.4.0/DocCD_files/Help/5.4.0/wlan_wmm_h_cnt.htm. [Online; Stand 27. September 2021]. 2006.

[35] Jeff Laird. *PTP Background and Overview*. https://www.iol.unh.edu/sites/default/files/knowledgebase/1588/ptp_overview.pdf. [Online; Stand 16. September 2021]. 2012.

Anhang

A.1 nl80211_txq_set/main.c: Tool zum Senden der EDCA-Parameter mit nl80211 an den Kernel.

```
#include <linux/nl80211.h>
#include <netlink/genl/ctrl.h>
#include <netlink/genl/family.h>
#include <netlink/genl/genl.h>
#include <netlink/netlink.h>
#include <stdbool.h>
#include <errno.h>

// Parts and inspired from: https://alamot.github.io/nl80211/
// and hostapd

typedef struct {
    int id;
    struct nl_sock *socket;
    struct nl_cb *cb1;
    int result1;
} Netlink;

typedef struct {
    char ifname[30];
    int ifindex;
} Wifi;

static int setTxQNl80211(Netlink *nl, Wifi *w, int ac, int txop, int
    cwmin, int cwmax, int aifs, int index)
{
    // allocate a netlink message structure
    struct nl_msg *msg = nlmsg_alloc();
    if (!msg) {
        fprintf(stderr, "Failed to allocate netlink message.\n");
        return -2;
```

```
31      }
32
33      // add generic netlink headers to the netlink message
34      genlmsg_put(msg, NL_AUTO_PORT, NL_AUTO_SEQ, nl->id, 0, 0,
        NL80211_CMD_SET_WIPHY, 0);
35      nla_put_u32(msg, NL80211_ATTR_IFINDEX, index);
36
37      struct nlattr *txq = nla_nest_start(msg,
        NL80211_ATTR_WIPHY_TXQ_PARAMS);
38      if (!txq) {
39          fprintf(stderr, "Failed to nest txq.\n");
40          return -2;
41      }
42      struct nlattr *params = nla_nest_start(msg, 1);
43      if (!params) {
44          fprintf(stderr, "Failed to nest txq params.\n");
45          return -2;
46      }
47
48      if (ac == 0) {
49          if (nla_put_u8(msg, NL80211_TXQ_ATTR_QUEUE, NL80211_TXQ_Q_VO)) {
50              fprintf(stderr, "Error putting ac.\n");
51              return -2;
52          }
53      } else if (ac == 1) {
54          if (nla_put_u8(msg, NL80211_TXQ_ATTR_QUEUE, NL80211_TXQ_Q_VI)) {
55              fprintf(stderr, "Error putting ac.\n");
56              return -2;
57          }
58      } else if (ac == 2) {
59          if (nla_put_u8(msg, NL80211_TXQ_ATTR_QUEUE, NL80211_TXQ_Q_BE)) {
60              fprintf(stderr, "Error putting ac.\n");
61              return -2;
62          }
63      } else if (ac == 3) {
64          if (nla_put_u8(msg, NL80211_TXQ_ATTR_QUEUE, NL80211_TXQ_Q_BK)) {
65              fprintf(stderr, "Error putting ac.\n");
66              return -2;
67          }
68      } else {
69          fprintf(stderr, "Error putting ac.\n");
70          return -2;
71      }
72      if (nla_put_u16(msg, NL80211_TXQ_ATTR_TXOP, txop)) {
73          fprintf(stderr, "Error putting txop.\n");
74          return -2;
75      }
```

```
76      if (nla_put_u16(msg, NL80211_TXQ_ATTR_CWMIN, cwmin)) {
77          fprintf(stderr, "Error putting cwmin.\n");
78          return -2;
79      }
80      if (nla_put_u16(msg, NL80211_TXQ_ATTR_CWMAX, cwmax)) {
81          fprintf(stderr, "Error putting cwmax.\n");
82          return -2;
83      }
84      if (nla_put_u8(msg, NL80211_TXQ_ATTR_AIFS, aifs)) {
85          fprintf(stderr, "Error putting aifs.\n");
86          return -2;
87      }
88
89      nla_nest_end(msg, params);
90      nla_nest_end(msg, txq);
91      nl_send_auto(nl->socket, msg);
92      nlmsg_free(msg);
93      return 0;
94  }
95
96  static int getWifiInfo_callback(struct nl_msg *msg, void *arg)
97  {
98
99      struct genlmsghdr *gnlh = (struct genlmsghdr *)nlmsg_data(nlmsg_hdr(
        msg));
100
101     struct nlattr *tb_msg[NL80211_ATTR_MAX + 1];
102
103     // nl_msg_dump(msg, stdout);
104
105     nla_parse(tb_msg, NL80211_ATTR_MAX, genlmsg_attrdata(gnlh, 0),
        genlmsg_attrlen(gnlh, 0), NULL);
106
107     if (tb_msg[NL80211_ATTR_IFNAME]) {
108         strcpy(((Wifi *)arg)->ifname, nla_get_string(tb_msg[
        NL80211_ATTR_IFNAME]));
109     }
110
111     if (tb_msg[NL80211_ATTR_IFINDEX]) {
112         ((Wifi *)arg)->ifindex = nla_get_u32(tb_msg[NL80211_ATTR_IFINDEX
        ]);
113     }
114
115     return NL_SKIP;
116 }
117
118 static int getWifiInfo(Netlink *nl, Wifi *w, int ifindex)
```

```
119 {
120     nl->result1 = 1;
121     //######################### Get wifi name and index
122     // allocate a netlink message structure
123     struct nl_msg *msg = nlmsg_alloc();
124     if (!msg) {
125         fprintf(stderr, "Failed to allocate netlink message.\n");
126         return -2;
127     }
128 
129     // add generic netlink headers to the netlink message
130     genlmsg_put(msg, NL_AUTO_PORT, NL_AUTO_SEQ, nl->id, 0, NLM_F_DUMP,
    NL80211_CMD_GET_INTERFACE,0);
131     nla_put_u32(msg, NL80211_ATTR_IFINDEX, ifindex);
132 
133     // finalize and transmit the netlink message
134     nl_send_auto(nl->socket, msg);
135 
136     // receive a set of messages from the netlink socket
137     while (nl->result1 > 0) {
138         nl_recvmsgs(nl->socket, nl->cb1);
139     }
140     // release the netlink message reference
141     nlmsg_free(msg);
142 
143     if (w->ifindex < 0) {
144         return -1;
145     }
146     return 0;
147 }
148 
149 static int finish_handler(struct nl_msg *msg, void *arg)
150 {
151     int *ret = (int *)arg;
152     *ret = 0;
153     return NL_SKIP;
154 }
155 
156 static int initNl80211(Netlink *nl, Wifi *w)
157 {
158     // allocate a netlink socket
159     nl->socket = nl_socket_alloc();
160     if (!nl->socket) {
161         fprintf(stderr, "Failed to allocate netlink socket.\n");
162         return -ENOMEM;
163     }
164 
```

```
165     // set the socket buffer size
166     nl_socket_set_buffer_size(nl->socket, 8192, 8192);
167 
168     // connect to the generic netlink socket
169     if (genl_connect(nl->socket)) {
170         fprintf(stderr, "Failed to connect to netlink socket.\n");
171         nl_close(nl->socket);
172         nl_socket_free(nl->socket);
173         return -ENOLINK;
174     }
175 
176     // ask the Kernel to resolve family name "nl80211" to family id
177     nl->id = genl_ctrl_resolve(nl->socket, "nl80211");
178     if (nl->id < 0) {
179         fprintf(stderr, "Nl80211 interface not found.\n");
180         nl_close(nl->socket);
181         nl_socket_free(nl->socket);
182         return -ENOENT;
183     }
184 
185     // allocate new callback handles
186     nl->cb1 = nl_cb_alloc(NL_CB_DEFAULT);
187     if (!nl->cb1) {
188         fprintf(stderr, "Failed to allocate netlink callback.\n");
189         nl_close(nl->socket);
190         nl_socket_free(nl->socket);
191         return ENOMEM;
192     }
193 
194     // set callbacks
195     nl_cb_set(nl->cb1, NL_CB_VALID, NL_CB_CUSTOM, getWifiInfo_callback,
    w);
196     nl_cb_set(nl->cb1, NL_CB_FINISH, NL_CB_CUSTOM, finish_handler, &(nl
    ->result1));
197     return nl->id;
198 }
199 
200 static bool isValidCw(int cw)
201 {
202     return cw == 1 || cw == 3 || cw == 7 || cw == 15 || cw == 31 || cw
    == 63 || cw == 127 ||
203            cw == 255 || cw == 511 || cw == 1023 || cw == 2047 || cw ==
    4095 || cw == 8191 ||
204            cw == 16383 || cw == 32767;
205 }
206 
207 // return 1: error in arguments
```

```
208 int main(int argc, char *argv[])
209 {
210     printf("Tool allows to set the EDCA parameters for the tx queue.\n")
    ;
211     printf("Use with following arguments:\n");
212     printf("1. ac: AC identifier (NL80211_AC_*). 0: VO, 1: VI, 2: BE, 3:
     BK\n");
213     printf("2. txop: Maximum burst time in units of 32 usecs, 0 meaning
    disabled\n");
214     printf("3. cwmin: Minimum contention window [a value of the form 2^n
    -1 in the range 1..32767].\n");
215     printf("\t Possible: 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047,
     4095, 8191, 16383, 32767\n");
216     printf("4. cwmax: Maximum contention window [a value of the form 2^n
    -1 in the range 1..32767]\n");
217     printf("\t Possible: 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 2047,
     4095, 8191, 16383, 32767\n");
218     printf("5. aifs: Arbitration interframe space [0..255]\n");
219     printf("6. ifindex\n");
220
221
222     // ################## check arguments ######################
223     if (argc < 6) {
224         printf("Missing Arguments. All arguments are required. \n");
225         return 1;
226     }
227     // set and check AC identifier
228     int ac = atoi(argv[1]);
229     if (!(ac == 0 || ac == 1 || ac == 2 || ac == 3)) {
230         printf("ac: invalid value\n");
231         return 1;
232     }
233     // set txop
234     int txop = atoi(argv[2]);
235     // set and check cwmin
236     int cwmin = atoi(argv[3]);
237     if (!isValidCw(cwmin)) {
238         printf("cwmin: invalid value\n");
239         return 1;
240     }
241     // set and check cwmax
242     int cwmax = atoi(argv[4]);
243     if (!(isValidCw(cwmax) && cwmax > cwmin)) {
244         printf("cwmax: invalid value\n");
245         return 1;
246     }
247     // set and check aifs
```

```
248     int aifs = atoi(argv[5]);
249     if (!(aifs >= 0 && aifs <= 255)) {
250         printf("aifs: invalid value\n");
251         return 1;
252     }
253     // set and check ifindex
254     int ifindex = atoi(argv[6]);
255     if (ifindex < 0) {
256         printf("ifindex: invalid value\n");
257         return 1;
258     }
259 
260     // ################ open netlink socket #############
261     Netlink nl;
262     Wifi w;
263     nl.id = initNl80211(&nl, &w);
264     if (nl.id < 0) {
265         fprintf(stderr, "Error initializing netlink 802.11\n");
266         return -1;
267     }
268 
269      // ################ configuration of parameters #############
270     getWifiInfo(&nl, &w, ifindex);
271     printf("Interface: %s \n", w.ifname);
272     printf("Interfaceindex: %d \n", w.ifindex);
273     setTxQNl80211(&nl, &w, ac, txop, cwmin, cwmax, aifs, ifindex);
274 
275     // ################ cleanup #############
276     nl_cb_put(nl.cb1);
277     nl_close(nl.socket);
278     nl_socket_free(nl.socket);
279     return 0;
280 }
```

Quellcode A.1: nl80211_txq_set/main.c: Tool zum Senden der EDCA-Parameter mit nl80211 an den Kernel.