Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Anushavan Melkonyan

Prototyp-basierte Analyse eines automobilen Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Anushavan Melkonyan

Prototyp-basierte Analyse eines automobilen Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 6. Juni 2019

Anushavan Melkonyan

Thema der Arbeit

Prototyp-basierte Analyse eines automobilen Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung

Stichworte

Virtual Local Area Network, Audio/Video-Bridging, Time-Triggered Ethernet, Fahrzeugnetz, Time-Sensitive Networking, Ethernet, CAN, Mikrocontroller, Steer-by-Wire

Kurzzusammenfassung

Das Ziel dieser Bachelorarbeit ist die prototyp-basierte Analyse eines TSN-Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung. Anhand der Virtual Local Area Networks (VLAN) soll untersucht werden, wie sich die Kommunikation in einem gemeinsamen Netzwerk sowohl zwischen priorisierten echtzeitbasierten Anwendungen als auch nicht-priorisierten echtzeitbasierten Anwendungen verhält. Es soll dabei analysiert werden, ob der nicht-priorisierte zeitkritische Cross-Traffic Auswirkung auf die Echtzeitfähigkeit des Systems hat. Des Weiteren wird geprüft, ob die priorisierten Nachrichten durch den Cross-Traffic beeinflusst werden. Dazu werden die Latenz sowie der daraus berechnete Jitter untersucht.

Anushavan Melkonyan

Title of the paper

Prototype-based analysis of a TSN communication backbone using Steer By Wire applications as an example

Keywords

Virtual Local Area Network, Audio/Video Bridging, Time-Triggered Ethernet, In-Vehicle Networks, Time-Sensitive Networking, Ethernet, CAN, Mikrocontroller, Steer-by-Wire

Abstract

The focus of this bachelor thesis is the prototype-based analysis of a TSN communication backbone using the example of Steer by Wire application.The purpose of the Virtual Local Area Network (VLAN) is to investigate how the communication in a shared network behaves between prioritized, real-time-based applications as well as non-prioritized, real-time-based applications. It should be analyzed whether the non-prioritized, time-critical cross-traffic has an impact on the real-time capability of the system. In addition, care is taken to ensure that the prioritized messages are influenced by the cross-traffic, in addition to the latency and calculated jitter is examined.

Inhaltsverzeichnis

1 Einleitung

Mit der zunehmenden Vernetzung von Komponenten im Automobil werden neue Funktionen realisiert, wie zum Beispiel die Einparkhilfe. Dabei spielen auch Multimediaanwendungen eine immer größere Rolle. Durch stetig wachsende Technologien, wie Sicherheit, Leistung und Komfort, ist die erforderliche Kommunikation zwischen den Komponenten gestiegen. Abbildung 1.1 zeigt beispielhaft ein komplexes Bordnetz eines Audi A8. Hier ist zu erkennen, dass Fahrzeuge ohne Bussysteme in der heutigen Entwicklung nicht mehr denkbar sind, da sich eine direkte Verkabelung der einzelnen Komponenten umfangreich gestaltet und dadurch die Wartbarkeit des gesamten Systems gemindert wird. Durch die Bussysteme kann von einer zentralen Stelle aus auf jede Komponente zugegriffen werden, wodurch die Elektronik übersichtlicher wird. Weiterhin ist der Vorteil einer zentralen Stelle, dass eine schnelle Diagnose des Systems erfolgen kann.
Aktuell sind in modernen Fahrzeugen bis zu einhundert Steuergeräte verbaut und die Zahl steigt kontinuierlich an, um den Anforderungen des Kunden standhalten zu können [4]. Der starke Anstieg der Vermaschung der Kommunikation zwischen Sensoren und ECUs, spielt eine zunehmend bedeutsamere Rolle in Fahrzeugen [5]. Durch höhere Bandbreitenanforderungen, wie zum Beispiel durch die von Laserscannern erzeugten Daten für das autonome Fahren, werden die Grenzen aktueller Busse überschritten (siehe Tabelle 1.1).

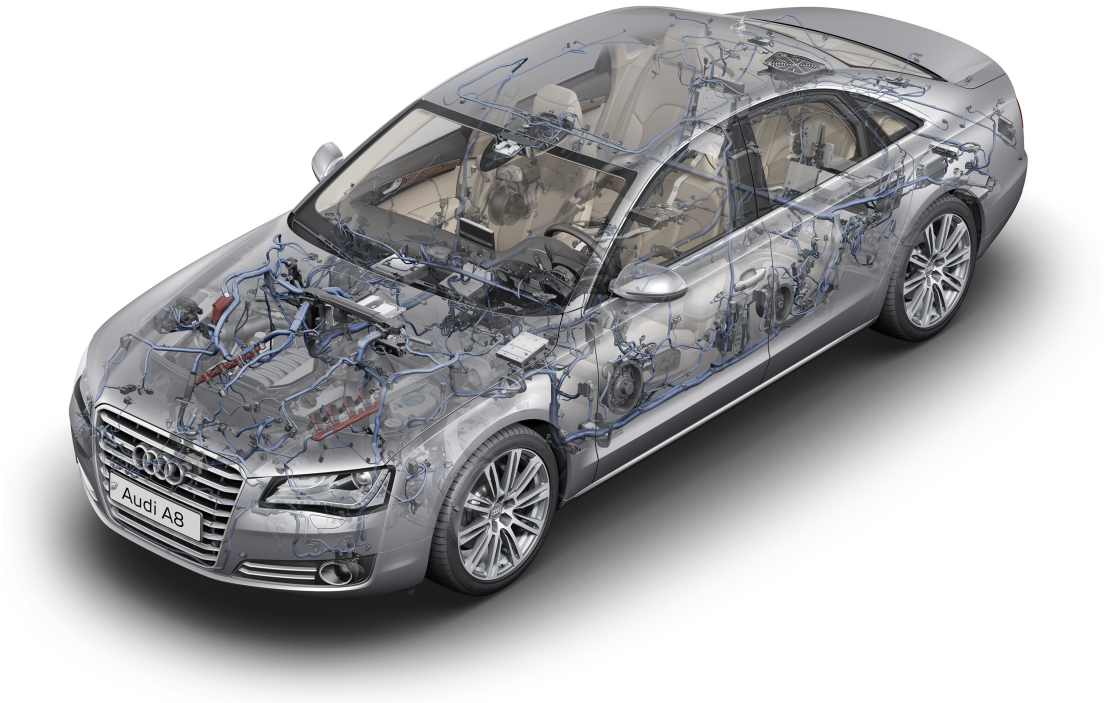

Abbildung 1.1: Bordnetz eines Audi A8. (Quelle[1]: VDI Wissensforum / AUDI AG)

Die Bandbreite der Busse ist limitiert, weshalb ein Netzwerk mit einer höheren Bandbreite benötigt wird, um zukünftige Technologien problemlos zu realisieren. Eine Variante zur Lösung dieses Problems stellt das heutzutage weitverbreitete Standard-Ethernet ‚IEEE 802.3' dar [6]. Doch um das standardisierte Netzwerk für ein Automobil nutzen zu können, muss es echtzeitfähig sein. Unter Echtzeit ist die Anforderung an ein Rechensystem gemeint, innerhalb einer kürzesten definierten Zeitspanne korrekt zu reagieren und dadurch Zeitgarantien einzuhalten. Dabei wird zwischen harten sowie weichen Anforderung unterschieden:

Harter Anforderungen: Ein Echtzeitsystem wird als hartes Echtzeitsystem (Hard-real-time-System) bezeichnet, wenn das Überschreiten der Zeitlimits bei der Reaktion erhebliche Folgen haben kann. So müssen beispielsweise Airbags oder die Motorsteuerung innerhalb der vorgegebenen Zeitschranke reagieren, da es andernfalls durch die Verzögerung zu Katastrophen für Mensch und Umwelt kommen kann.

Weicher Anforderungen: Eine Verletzung der Ausführungszeiten in einem weichen Echtzeitsystem (Soft-real-time-System) führt ausschließlich zu einer Verminderung der Qualität, nicht jedoch zu einer Beschädigung oder Gefährdung. Beispielsweise ist es bei der Eingabe einer Adresse in das Navigationssystem nicht relevant, ob die Routenplanung in einem bestimmten Zeitraum geschieht oder erst nach zwei bis drei Sekunden. Durch diese Verzögerung kann es zu keinen gravierenden Problemen kommen, sondern es verschlechtert sich lediglich die Qualität des Produktes.

Ein echtzeitfähiges Netzwerkprotokoll ist beispielsweise das von der Firma TTTech entwickelte Time-Triggered-Ethernet (TTE 2.3)[7]. Dieses benötigt meistens einen hohen Offline-Konfigurationsaufwand, der mit zunehmender Netzkomplexität durch neue Module weiter steigt. Um diesen Offline-Konfigurationsaufwand zu umgehen, werden Protokolle wie Virtual Local Area Network (VLAN) oder Audio/Video-Bridging (AVB) verwendet. Zurzeit beschäftigen sich Automobilhersteller und die IEEE TSN Task Group [8][9] mit der Erweiterung des AVB-Standards um eine zeitbasierte Kommunikation. TSN steht für ‚Time-Sensitive Networking' das seit November 2012 den Nachfolger von AVB bildet. Die Automobilhersteller sowie die TSN Task Group arbeiten an einem Konzept, das die Verbesserung der Genauigkeit und der Latenz von AVB < 2ms für höchst kritische Anwendungen ermöglichen soll, damit die zeitkritische Kommunikation im Fahrzeug nicht zu spät übermittelt wird.

Es wird angestrebt, dass sowohl echtzeitbasierte als auch nicht-echtzeitbasierte Anwendungen im selben Netzwerk miteinander kommunizieren. Allerdings darf die Echtzeitfähigkeit nicht durch den nicht-zeitkritischen Datenverkehr beeinträchtigt werden. So dürfen beispielsweise in einem Auto die Pakete zur Kommunikation mit der Bremse oder dem Lenkrad nicht durch das Infotainmentsystem des Fahrzeugs erheblich verzögert werden.

Das Konzept von ‚*Drive-by-Wire*'[10]bezeichnet in der Automobilindustrie das teilweise Fahren oder Steuern von Fahrzeugen ohne mechanische Kraftübertragung der Bedienelemente zu den entsprechenden Stellelementen wie etwa Drosselklappen. Die Steuerung von Funktionen findet über elektrische Leitungen und Servomotoren bzw. elektromechanische Aktoren statt. Die aktuelle Entwicklung im Kraftfahrzeugbau tendiert dazu, Fahrzeugbefehle rein elektrisch weiterzuleiten, wie bei ‚*Shift-by-Wire*', ‚*Steer-by-Wire*' oder ‚*Brake-by-Wire*'. So stellen beispielsweise ‚*Steer-by-Wire-Anwendungen*' besondere Anforderungen an die zeitgerechte Datenübertragung. Aufgrund der regelungstechnischen Anforderungen muss die Kommunikation eine obere Grenze für die Latenzen garantieren.

Bussysteme	Übertragungsrate
LIN	20 kbit/s
CAN	1 Mbit/s
MOST	23 Mbit/s
Flex-Ray	10 Mbit/s
Real-time Ethernet	100 Mbit/s - 1 Gbit/s

Tabelle 1.1: Bandbreitenvergleich zwischen Ethernet und Bussen

Diese Arbeit ist in der ‚Communication-over-Realtime-Ethernet-Arbeitsgruppe' (CoRE RG) [11] an der HAW Hamburg entstanden. Diese Arbeitsgruppe befasst sich mit der Realisierung von Echtzeit-Ethernet-Kommunikation in Flugzeugen und Autos. Den Schwerpunkt dieser Bachelorarbeit bildet die prototyp-basierte Analyse eines automobilen Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung. Dazu gehört die Implementierung von IEEE-802.1Q-Standards. Anhand von priorisiertem Virtual Local Area Networks (VLAN) und Best Efforts (BE) soll untersucht werden, wie sich die Kommunikation in einem gemeinsamen Netzwerk sowohl zwischen priorisierter als auch nicht-priorisierter Anwendung verhält. Dafür werden verschiedene Testszenarien erstellt, um zu analysieren, wie sich die Latenzen sowie der Jitter in unterschiedlichen Szenarien verhalten. Des Weiteren wird die Auswirkung mit priorisierter Anwendung und BE-Cross-Traffic sowie mit nicht-priorisierter Anwendung und BE-Cross-Traffic untersucht. Dabei soll festgestellt werden, ob und wann Verzögerungen im System auftreten.

Die vorliegende Bachelorarbeit ist wie folgt aufgebaut: In Kapitel 2 werden die Grundlagen erörtert, die für das Verständnis der Arbeit notwendig sind. Dazu gehört eine kurze Einführung zu Ethernet, IEEE 802.1Q (VLAN) sowie Time-Triggered Ethernet (TTE). In Kapitel 3 wird das Vorhandene System vorgestellt sowie die Aufgabenstellung erläutert. Anschließend werden in Kapitel 4 das Konzept sowie der Programmablauf vorgestellt. In Kapitel 5 geht es um die Realisierung der Kommunikation auf IEEE 802.1Q. Des Weiteren wird der Switch vorgestellt sowie auf die Implementierung eingegangen . Anschließend wird die Konfiguration des Systems erläutert. In Kapitel 6 werden die Tools, um die Anforderungen der Qualitätssicherung zu erfüllen, vorgestellt. Des Weiteren werden generelle Funktionen der Implementierung getestet. In dem Kapitel 7 geht es um die Evaluierung des Systems. Dazu wird die Hardware für die Messungen vorgestellt sowie die Hardware zu Erzeugung des Cross-Traffics. Danach wird der Messaufbau schematisch dargestellt. Anschließend werden die Sende- und Empfangsverzögerung des Hilscherboards ermittelt. Des Weiteren wird die analytische Berechnung der Bandbreite sowie der

Latenzen von Control-Flow- und Cross-Traffic-Kommunikation durchgeführt. Die Berechnung der Bandbreite dient der späteren Überprüfung der gemessenen Werte. Es werden Messungen mit und ohne Prioritäten durchgeführt. Alle Messungen werden mit und ohne Cross-Traffic gemessen. Abschließend werden im letzten Kapitel eine kurze Zusammenfassung und ein Fazit zur Arbeit gegeben.

2 Grundlagen

In diesem Kapitel werden die Grundlagen für das weitere Verständnis der Arbeit erläutert. Dazu zählen die grundlegenden Funktionsweisen von Ethernet, 802.1Q und Time-Triggered Ethernet.

2.1 Ethernet

Das Ethernet, das in der Norm IEEE 802.3 [6] standardisiert wurde, ist eines der am weitesten verbreiteten Netzwerkprotokolle, das die erste und zweite OSI-Schicht definiert. Es stellt eine Reihe von Standards für Soft- und Hardware (wie z. B. Switch oder Kabel) zur Verfügung, die für Local Area Networks (LAN) verwendet werden. Außerdem hat es sich gegen andere Netzwerktechnologien wie Token Ring oder Apple Talk durchgesetzt [12] und wird weitläufig verwendet. Ein Netzwerk besteht aus mehreren Knoten, auch Switches genannt, die miteinander verbunden werden, um Pakete von Host A nach Host B zu verschicken. Die Frames werden stets folgendermaßen aufgebaut: Der Header enthält die Informationen für den korrekten Transport über mehrere Knoten und die Payload enthält die eigentlichen Daten (siehe Abbildung 2.1). Dabei variieren auch die Übertragungsgeschwindigkeiten voneinander (siehe Tabelle 2.1).

Ethernet	**Übertragungsrate**
Fast Ethernet	1—100 Mbit/s
Gigabit Ethernet	1000 Mbit/s

Tabelle 2.1: Bandbreite

Abbildung 2.1: Aufbau eines Ethernetframes

2.2 IEEE 802.1Q / VLAN - Virtual Local Area Network

Virtual Local Area Networks (VLAN) sind virtuelle lokale Netze, die in IEEE 802.1Q standardisiert sind und auf Schicht zwei des OSI-Schichtenmodells arbeiten [13][14]. Die Virtual Local Area Network unterteilen ein bestehendes einzelnes physisches Netzwerk in mehrere logische Netzwerke. Jedes VLAN bildet dabei eine eigene Broadcast-Domain. Es existieren zwei Implementierungsvarianten für VLAN. Eine der beiden ist portbasiert: Netzwerke werden logisch in managebare Switche segmentiert, das heißt einzelne Ports werden dabei einem VLAN zugeordnet. So können die verbundenen Geräte nur im eigenen virtuellen Netzwerk miteinander kommunizieren. Beim tagged VLAN werden einzelne Frames durch einen zusätzlichen ‚Tag' gekennzeichnet, der die Zugehörigkeit zu einem VLAN festlegt. Dafür wurde der Ethernet-Standard entsprechend erweitert. Mit 802.1Q wird die Paketstruktur des Standard-Ethernet-Pakets um 4 Byte erweitert, und zwar zwischen dem Datenfeld für die Quelladresse (SA) und dem Typefeld, wie in Abbildung 2.2 dargestellt. Durch die zusätzlich hinzugefügten 4 Byte wird die maximale Länge eines Frames auf 1522 Byte vergrößert. Dadurch ist es möglich, dass ein Paket in verschiedene (Prioritäts-)Klassen und VLANs eingeordnet werden kann. Die übliche Queue von Ethernet wird bei 802.1Q in mehrere Queues aufgeteilt (siehe Abbildung 2.3). Jede Prioritätsklasse erhält eine eigene Queue.

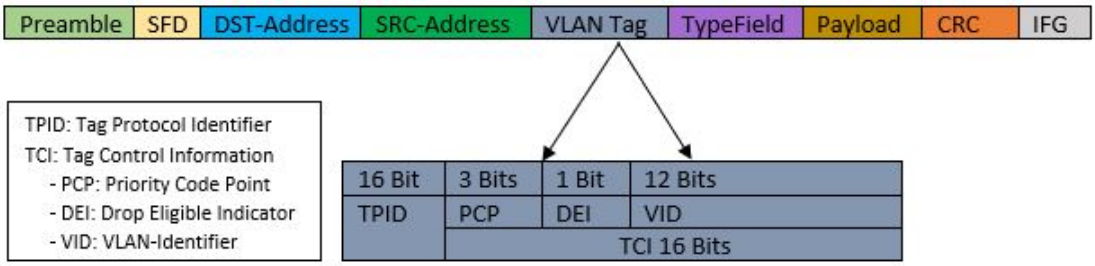

Abbildung 2.2: Aufbau eines Ethernetframes mit VLAN-TAG

Der VLAN-TAG besteht aus dem Tag-Protocol-Identifier (TPID) und der Tag-Control-Information (TCI) (siehe Abbildung 2.2). Der TPID besteht aus 2 Byte und wird immer auf den fest-reservierten Wert von 8100 (hex) gesetzt. Die TCI besteht aus dem Priority Code Point (PCP), in dem die Priorität des Frames festgelegt wird (drei Bits) und dem Drop Eligible Indicator (DEI), ehemals Canonical Format Indicator (CFI). Der Drop Eligible Indicator kann in Verbindung mit PCP oder separat verwendet werden und wird zum Anzeigen von Frames, die in der Gegenwart von Staus fallen gelassen werden, verwendet. Die letzten zwölf Bit werden für den VLAN Identifier (VID) verwendet. Mit dem TCI-Feld ist die Unterteilung in acht verschiedene Pakete möglich. Der Scheduler endscheidet anhand der Prioritätsklasse, in

welcher Reihenfolge die nächsten Pakete versendet werden. Pakete derselben Prioritätsklasse werden nach dem FIFO-Prinzip behandelt. Der Nachteil bei diesem Prinzip ist die Starvation. Das bedeutet, dass durch die unterschiedlichen Prioritätsklassen, die niedrig-priorisierten Pakete verzögert oder bei einem persistent auftretenden Strom (Burst) von Paketen einer höheren Prioritätsklasse komplett blockiert werden und so niemals ihr Ziel erreichen. Der Vorteil hingegen ist, dass Pakete nur auf Pakete gleicher oder höherer Prioritätsklasse warten müssen. Auch das Blockieren der Pakete durch niedrig-priorisierte Pakete ist möglich, sofern diese versendet wurden und nicht unterbrochen werden können. Bei AVB wird die Starvation durch den Credit-Based-Shaper (CBS) vermieden. Der CBS-Algorithmus sorgt unter anderem dafür, dass die vorher reservierten Bandbereiten durch das ‚Stream-Reservation-Protokoll' [15] eingehalten werden. Dieses beinhaltet sowohl die Zusicherung der Allokation als auch die Nichtüberschreitung des physikalischen Links. Der Credit wird beim Senden dekrementiert und anschließend, wenn der Wert negativ ist, bis zum nächsten Sendevorgang auf gleich null steigen. Die negative Steigung des Credits trägt den Namen *sendSlope* und die positive *idleSlope*. Ein AVB-Paket darf nur versendet werden, wenn der Credit einen Wert gleich null oder darüber aufweist. Wenn kein AVB-Sendevorgang stattfindet und der Credit kleiner null ist, wird der Wert des Credits mit dem *idleSlope* wieder bis auf null gesteigert. In diesem Zeitraum können Best-Effort-Nachrichten (BE-Nachrichten) versendet werden. Aus diesem Grund ergibt sich bei diesem Ansatz nicht das Problem mit der Starvation, da dadurch keine BE-Pakete blockiert werden können.

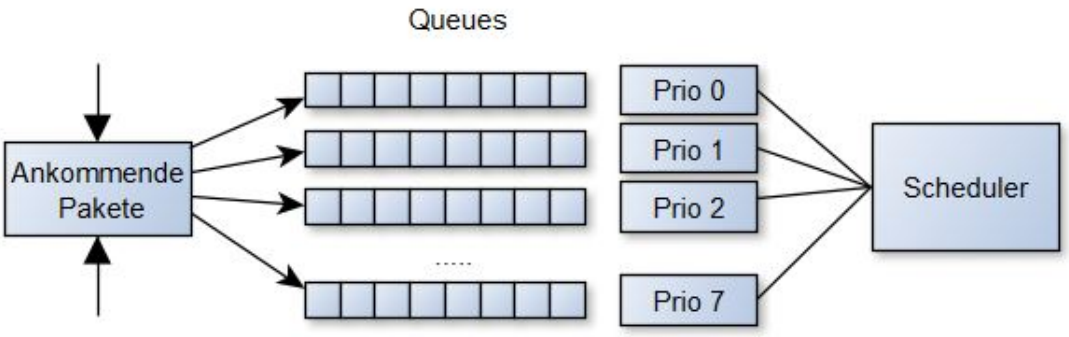

Abbildung 2.3: Aufbau der Warteschlange bei 802.1Q

2.3 Time-Triggered Ethernet

Beim Time-Triggered Ethernet handelt es sich um eine Netzwerktechnologie, die Scheduling verwendet, um deterministische Echtzeitkommunikation über Ethernet zu ermöglichen. Das Time-Triggered-Ethernet-Protokoll (TTE-Protokoll), das von TTech Computertechnik AG entwickelt wurde, ist eine Echtzeit-Erweiterung des ursprünglichen Ethernet-Standards, der in der IEEE-Norm 802.3 [6] spezifiziert ist. Diese ist im SAE-AS6802-Standard [16] festgelegt. Durch das Time-Triggered-Ethernet-Protokoll wird eine Übertragung mit niedriger Latenz und Jitter ermöglicht, wenn unterschiedliche Nachrichtenklassen über ein physikalisches Medium transportiert werden.

2.3.1 Nachrichtenklassen

Das TT-Ethernet unterstützt drei verschiedene Nachrichtenklassen. Time-Triggered (TT), Rate-Constrained (RC) und Best-Effort (BE), die unterschiedlich im System priorisiert werden. Den Inhalt von Nachrichten aus höheren Netzwerkschichten, zum Beispiel IP und TCP, behandelt TT-Ethernet transparent.

Time-Triggered-Nachrichten

Diese Nachrichtenklasse wird für zeitkritischen Netzwerkverkehr verwendet. Die Time-Triggered-Kommunikation hat im System die höchste Priorität, weshalb andere Nachrichtenklassen nicht gesendet werden dürfen, wenn sie sich mit TT-Nachrichten überschneiden. Dadurch verdrängen die TT-Nachrichten-Pakete alle anderen Nachrichtenklassen. Um dies zu ermöglichen, werden die Sende- und Empfangszeitpunkte offline Mikrosekunden genau geplant und zyklisch durch den Scheduler verschickt. Dafür muss das gesamte Netzwerk auf eine globale Zeit synchronisiert werden. Das Absenden von TT-Nachrichten wird durch die voreingestellte Zeit bestimmt, die in der Offline-Konfiguration definiert ist. In Hosts und Switches werden die Routen über die physikalischen Links statisch festgelegt.

Rate-Constrained-Nachrichten

Diese Nachrichtenklasse hat nach den TT-Nachrichten die zweithöchste Priorität. Bei RC handelt es sich um einen ereignisbasierten Nachrichtentyp, der keinem Scheduling unterliegt, d. h. die Übertragung ist unabhängig von der globalen Zeit. Die Routen sowie die Bandbreite werden offline konfiguriert. Abhängig von Priorität und Nachrichtenaufkommen erhöht sich die Latenz bei RC-Nachrichten dementsprechend. Da

RC-Nachrichten bandbreiten-basiert sind und eine Bandbreite garantiert wird, sind sie in der Charakteristik mit der AVB-Stream-Reservation vergleichbar.

Best-Effort-Nachrichten
Diese Nachrichtenklasse ist äquivalent zum Standard IEEE 802.3 Ethernet-Nachrichten und hat im System die niedrigste Priorität. Dadurch wird diese Nachrichtenklasse im Netzwerk von allen anderen verdrängt. Es gibt keine Garantie dafür, dass ein Paket erfolgreich übertragen wird. Die BE-Nachrichten werden bei einer nicht fehlerhaften Konfiguration von RC mit der restlichen, von TT und RC nicht genutzten Bandbreite übertragen.

2.3.2 Synchronisation

Bei TTE spielt die Synchronisation des Senders und Empfängers eine fundamentale Rolle. Damit ein zeitlich korrekter Ablauf der Aufgaben möglich wird, müssen alle Teilnehmer im Netzwerk ihre Systemzeit angleichen. TT-Ethernet sieht dafür eine fehlertolerante Zwei-Wege-Synchronisation vor, die durch eine Master-/Slave-Architektur realisiert wird. Dafür nehmen die Teilnehmer im Netzwerk unterschiedliche Rollen an wie ‚Synchronization-Master' (SM), ‚Compression-Master' (CM) und ‚Synchronization-Client' (SC).

Synchronisation-Master initialisieren die Synchronisation. Es wird ein Synchronisationsframe versendet sowie ein Protocol-Control-Frame (PCF), das die aktuelle Systemzeit des Masters enthält.

Compression-Master sind meistens Switche. Die gesendete Zeit von jedem SM wird vom CM empfangen, anhand dessen wird eine neue globale Systemzeit berechnet, die anschließend an alle Clients (SC) gesendet wird.

Synchronisation-Clients aktualisieren die durch den CM neu berechnete Systemzeit als ihre lokale Systemzeit.

Mittels eines PCF-Frames senden zuerst alle Synchronisation-Master ihre lokale Systemzeit zur Synchronisation an die Compression-Master. Diese berechnen anhand dieser Zeiten eine neue globale Systemzeit und versenden diese an die Synchronisation-Clients. Die SC passen ihre Uhrzeit auf die empfangene neue Uhrzeit an. Damit auch bei einem Ausfall die Synchronisation korrekt funktioniert, können im System mehrere Synchronisation-Master und Compression-Master definiert werden.

2.4 Latenz und Jitter

Latenz und Jitter sind Begriffe der Nachrichtentechnik und werden oft im Bereich der Signal- oder Nachrichtenübertragung verwendet. Dabei bezeichnet die Latenz normalerweise die Übertragungsdauer einer Nachricht vom Sender zum Empfänger. Der Jitter dagegen ist die Variation dieser Verzögerung. Die beiden Begriffe werden im weiteren Verlauf dieser Arbeit auf Ausführungszeiten des Systems ausgedehnt. Dadurch ist es möglich, das Verhalten des Programms und seiner Metriken zu beschreiben.

3 Anforderungen

Die Anforderung dieser Bachelorarbeit besteht darin, ein vorhandenes Kommunikations-Backbone einer Steer-by-Wire-Anwendung durch eine neue Kommunikationsmethode zu ersetzen. Die alte Kommunikation verlief über Time-Triggered Ethernet (TTE), das durch die neue Kommunikation 802.1Q ersetzt wurde.
Als weitere Hauptanforderung soll eine Analyse des neuen Kommunikations-Backbones anhand unterschiedlicher Testszenarien sowie mithilfe verschiedener Metriken für die Perfomancetests und die Lasttests durchgeführt werden.

3.1 Vorhandenes System

Das ursprüngliche System war eine auf Mikrocontroller basierte Steer-by-Wire-Anwendung, die auf echtzeitfähige Kommunikation ausgerichtet war. Dabei wurden die CAN- bzw. CANopen-basierten Komponenten wie der Lenkradmotor, der Motor des Lenkgetriebes sowie der Kraftaufnehmer zu einem Regelkreis geschlossen. Das System besteht aus vier Komponenten: einem Controller, einem Kraftaufnehmer, einem Lenkrad sowie einem Rad (siehe Abbildung 3.1). Der Controller liest in regelmäßigen Zeitabständen die Position des Lenkrades aus, dann berechnet er den daraus resultierenden Lenkwinkel und stellt diesen dementsprechend ein. Der Kraftaufnehmer des Rades nimmt die Querlenkkraft auf und übermittelt die Werte an den Controller, der diese wiederum an das Lenkrad weiterleitet. Diese Art von Kommunikation wird eingesetzt, um Force-Feedback-Verhalten zu simulieren. Weitere Erläuterungen hierzu finden sich in der Bachelorarbeit von Vitalij Stepanov [2].

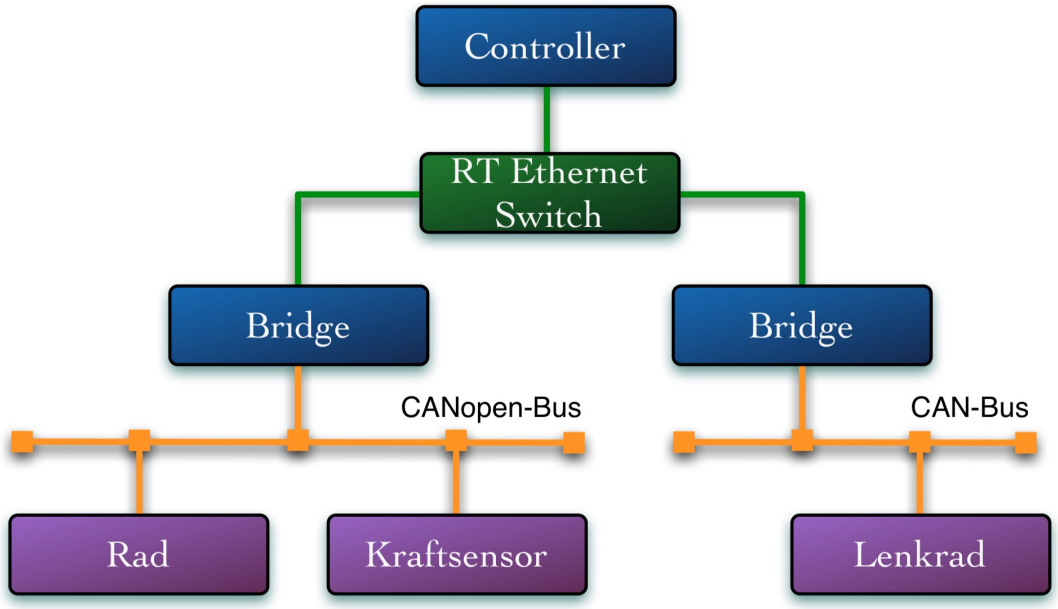

Abbildung 3.1: Konzept des Demonstrators (Quelle[2]: Vitalij Stepanov—Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen, 2011.)

Um das System echtzeitfähig zu machen, wurde die von Kai Müller [17] entwickelte TTE-API (siehe 2.3) eingesetzt, die den vorhandenen Ethernet-Stack um die Echtzeitfähigkeit erweitert. Des Weiteren wurde bei dem Aufbau ein TTE-Switch verwendet, der die Weiterleitung bestimmter Nachrichten zu vorgegebenen Zeiten ermöglicht. Dafür muss der TTE-Switch zuvor offline konfiguriert werden. Um die Anlage mit TT-Ethernet zu realisieren, wurde das System darüber hinaus um den Aspekt der Bridge erweitert. Die Bridge empfängt CAN-Nachrichten vom CAN-BUS, packt die Nachrichten in Ethernet-Pakete und leitet sie anschließend über den Backbone an eine andere Bridge weiter. Dort werden sie erneut in CAN-Nachrichten umwandelt, um die Nachrichten wieder auf den CAN-BUS zu legen. Weitere Erläuterungen hierzu finden sich in der Bachelorarbeit von Jan Kamieth [18].

3.2 Aufgabenstellung

Ein Aspekt dieser Arbeit besteht darin, das vorhandene System mithilfe des Kommunikations-Protokolls TT-Ethernet auf 802.1Q umzusetzen und anschließend mit unterschiedlichen Testszenarien zu evaluieren. Dafür muss der Frame-Aufbau der Ethernet-Pakete um VLAN erweitert

werden. Der TTE-Switch wird durch einen 802.1Q-fähigen Switch (siehe 5.1) ersetzt, um die Realisierung auf 802.1Q zu ermöglichen. Anhand der unterschiedlichen Prioritäten bei 802.1Q können Cross-Traffic-Nachrichten (CT-Nachrichten) auf dieselbe Leitung zugeführt und dabei die Hauptanwendung priorisiert versenden (siehe Abbildung 3.2). Dazu muss ein QoS-Profile in dem Summit X440 Switch angelegt und konfiguriert werden, damit der Control Flow (CF) priorisiert übertragen wird. Hiermit ist die Kommunikation von 802.1Q gemeint.

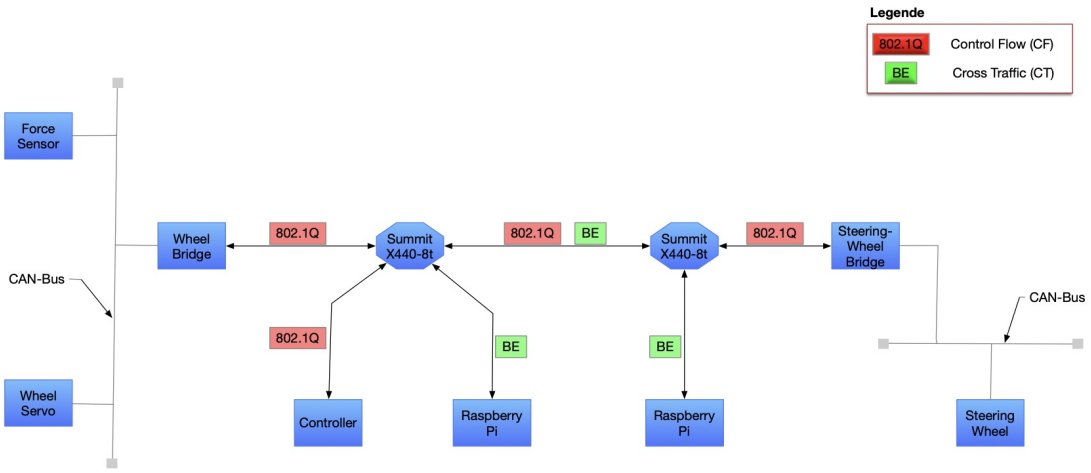

Abbildung 3.2: Schematische Darstellung des Demonstrators mit 802.1Q

Es müssen Aussagen getroffen werden, wie sich die Latenz des CF sowohl mit und ohne CT als auch mit oder ohne Priorität verhalten. Anschließend wird aus den Latenzen der Jitter ermittelt. Dabei sollte die 100-Mbit/s-Leitung mit CT voll ausgelastet werden, um Unterschiede zwischen den Prioritäten festzustellen, da die Anlage kleine Pakete versendet und dadurch die Bandbreite von 100 Mbit/s kaum genutzt wird. Es soll verdeutlicht werden, wann die Anlage nicht mehr funktionsfähig ist.

4 Konzept

In diesem Kapitel wird das vorhandene Konzept zur Umsetzung vorgestellt und erweitert. Anschließend wird ein Überblick über den Programmablauf gegeben.

4.1 Konzept und Programmablauf

Das Konzept der Bridge wurde von Jan Kamieth [18] übernommen und erweitert. Beispielsweise wurde der Frame-Aufbau in der Methode $can2eth$ erweitert. Dies wird in Kapitel 5.2 genauer erläutert.
Die von Jan Kamieth (2011) entwickelte Bridge übernimmt die Aufgabe, die ereignisgesteuerte CAN-Kommunikation auf die zeitgesteuerte Real-time-Ethernet-Kommunikation umzusetzen. Dafür wurde der gesamte Traffic in Real-time-Ethernet-Kanäle aufgeteilt. Man unterscheidet Time-Triggered- , Rate-Constrained- und Best-Effort-Kanäle.
Kamieth stellt in seiner Bachelorarbeit unterschiedliche Routingverfahren vor, die zur Gewährleistung eines effektiven Einsatzes der Bridge benötigt werden. Die Entscheidung fiel auf die vereinfachte Routingtabelle, die in dieser Arbeit auch angewendet wird. Für einen flexibleren Einsatz der Bridge wurde zusätzlich ein virtueller CAN-Stack implementiert. Dadurch ist der Wechsel zwischen realem und virtuellem CAN-Bus ohne größeren Implementationsaufwand möglich. Virtuell bedeutet, dass die CAN-Nachrichten nicht auf den CAN-Bus geschrieben, sondern von der Bridge direkt über den TT-Ethernet-Backbone weitergeleitet werden, beispielsweise an einen entfernten CAN-Bus. Es wurde für jeden CAN-Identifier ein virtueller Link eingerichtet, über den die Ethernet-Nachrichten übertragen werden. Um die Konfiguration zu erleichtern, wurden diese virtuellen Links mit derselben ID konfiguriert, wie die zugehörige CAN-Nachricht. Die CAN-Nachricht mit der ID 0x200 wird demnach über den virtuellen Link mit der ID 0x200 versendet. Um das Routingverhalten zu realisieren, wurde ein Header $bridge\_routing.h$ erstellt. Dieser Header enthält je zwei Tabellen für die drei Komponenten, für eingehenden $bridge\_rtable\_rx$- sowie ausgehenden $bridge\_rtable\_tx$-Verkehr (siehe Listings 4.1). Über die CAN-ID- und ETH-ID-Einträge wird eine Beziehung zwischen einer CAN-Nachricht und einer 802.1Q/Ethernet-Nachricht hergestellt. Das heißt, eine CAN-

Nachricht mit der ID X wird stets über die 802.1Q/Ethernet-Nachricht mit der ID Y versendet. Im Listing 4.1 ist das Routingverhalten des Lenkrades zu sehen.

```
#ifdef BRIDGE_STEER
static bridge_rtable_entry bridge_rtable_tx[] =
{
// {CAN ID, ETH ID}
  { 0x210, 0x210},
  { 0x211, 0x211},
  { 0x21A, 0x21A}
};

static bridge_rtable_entry bridge_rtable_rx[] =
{
// {CAN ID, ETH ID}
  { 0x200, 0x200},
  { 0x201, 0x201},
  { 0x20A, 0x20A}
};
#endif
```

Listing 4.1: Routingverhalten des Lenkrades

In der ersten Spalte der TX-Tabelle stehen die CAN-IDs, die über den virtuellen Link mit der ID aus der zweiten Spalte versendet werden. Die Einträge sind jeweils gleich, weil die IDs für die virtuellen Links so gewählt wurden, dass die Konfiguration der Routingtabellen erleichtert wird. Die RX-Tabelle ist entsprechend gegengesetzt zu interpretieren.

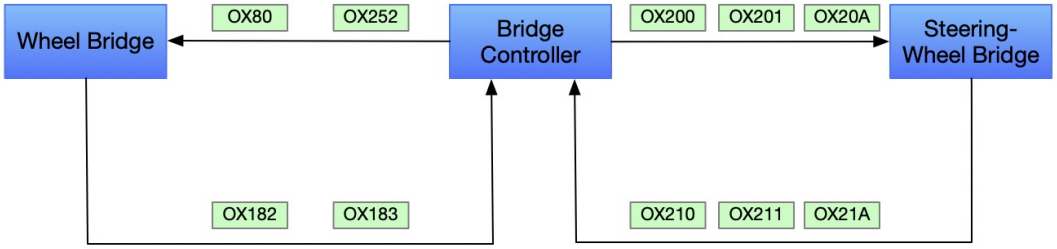

Abbildung 4.1: CF-IDs des Demonstrators

Die Nachrichten werden einzeln und periodisch in einem Takt von 5 ms versendet. Die virtuellen Links mit den IDs 0x200, 0x201 und 0x20A werden vom Controller an die Bridge des Lenkrades gesendet. Daraufhin versendet diese die Antworten mit den IDs 0x210, 0x211 und 0x21A an den Controller. Die IDs 0x80 sowie 0x252 werden vom Controller an die Bridge des Rades gesendet. Diese antwortet darauf mit den IDs 0x182 und 0x183. In der nachfolgenden Abbildung 4.1 sind die virtuellen Links abgebildet.

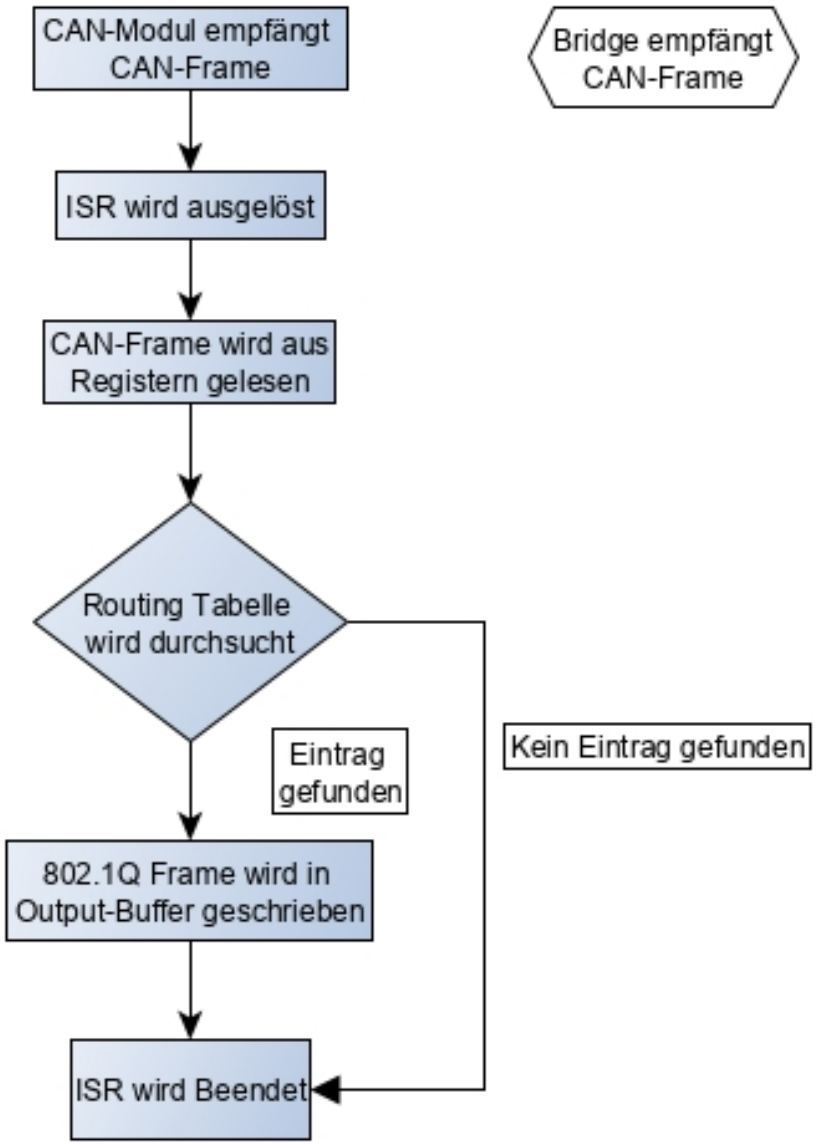

Abbildung 4.2: Ablauf beim Empfang von CAN-Frames in der ISR

In Abbildung 4.2 wird der Ablauf des Programms beim Empfang eines CAN-Frames dargestellt. Wenn das CAN-Modul ein CAN-Frame empfängt, wird die ISR ausgelöst. Daraufhin wird das CAN-Frame aus dem Register gelesen. Die Routing-Tabelle wird nach der empfangenen ID durchsucht. Wenn die ID vorhanden ist, wird die CAN-Nachricht in eine 802.1Q-Nachricht

umgewandelt und in den Output-Buffer geschrieben. Andernfalls wird sie verworfen und die ISR beendet.

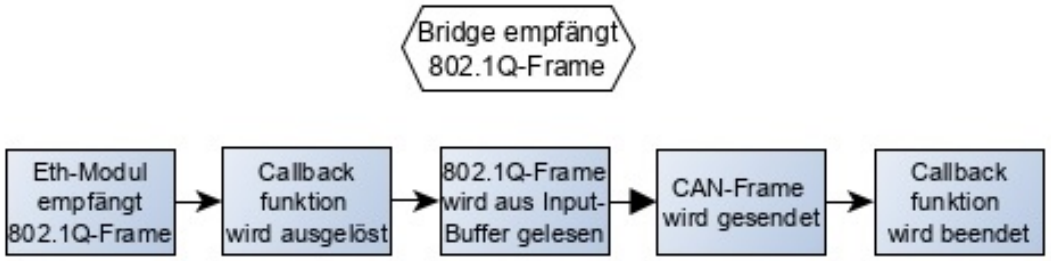

Abbildung 4.3: Ablauf beim Empfang von 802.1Q-Frames in der ISR

Beim Empfang eines 802.1Q-Frames sieht der Ablauf des Programms etwas anders als bei den CAN-Frames aus (siehe hierzu Abbildung 4.3). Wenn das Ethernet-Modul einen 802.1Q-Frame empfängt, wird die Callback-Funktion ausgelöst. Daraufhin wird der Frame aus dem Input-Buffer gelesen. Als nächstes wird aus dem 802.1Q-Frame die Payload mit der CAN-ID herauskopiert und in die ursprüngliche CAN-Nachricht umgewandelt. Anschließend wird der CAN-Frame in simulierte Register gelegt, die als Ringbuffer implementiert sind, woraufhin die Callback-Funktion beendet wird. Die Größe der Ringbuffer kann über die Konstante CAN_FRAME_BUFFER_SIZE konfiguriert werden. Da sich der CAN-Identifier mit im Payload der 802.1Q-Nachricht befindet, muss beim Empfang keine Abbildung mehr von 802.1Q-ID auf CAN-ID stattfinden. Deshalb kann die CAN-Nachricht sofort auf den CAN-BUS geschrieben werden.

5 Realisierung mit IEEE 802.1Q

Dieses Kapitel beschreibt die Realisierung der Software bzw. das Zusammenspiel mit weiteren Bachelorarbeiten im Projektumfeld. Darüber hinaus werden die Switche vorgestellt und es wird auf die Umsetzung eingegangen.

5.1 Summit X440-8t

Der Summit-X440-8t-Switch der Firma *Extreme Networks* besitzt eine CPU mit 500 MHz sowie 512 MB RAM und 512 MB Flash. Der Summit® X440 unterstützt das modulare und robuste ExtremeXOS®-Netzwerkbetriebssystem, das ermöglicht damit ein durchgängiges Betriebssystem mit einheitlichem Funktionsumfang in allen Bereiches des Netzes. Der Switch verfügt über 8-Gbit-Ethernet-Ports, die mit 10/100/1000 Mbit/s konfiguriert werden können. Des Weiteren enthält der Switch einen Port für eine serielle Konsole sowie einen 100-Mbit/s-Management-Port. Er besitzt umfangreiche QoS-Möglichkeiten für den Einsatz in Multimedia-Anwendungen und ist vielfältig konfigurierbar.

Abbildung 5.1: Summit X440-8t

5.2 Implementierung

Die von Kamieth [18] entwickelte Bridge wurde erweitert, damit das Verschicken von 802.1Q-Nachrichten ermöglicht wird. Dafür wurde ein Header für VLAN erstellt (vlan-header.h siehe Listing 5.1), der in die bridge-can.c implementiert wird. Die Datei vlan-header.h enthält den

Ethertype, die Prioritätsklassen, die DEI und die VLAN-ID. Insgesamt wird der Frame durch den vlan-header um 4 Byte erweitert. Die Implementierung für 802.1Q wurde in der Bridge in der Methode bridge-can2eth realisiert. Dort, wo ein Frame von CAN zu Ethernet eingebettet wird, wurde Control Flow (CF) in die Payload implementiert, wie es im Listing 5.2 zu sehen ist. In der Bridge mit der Methode bridge-can2eth werden der Ethernet-Header und anschließend die Payload zusammengebaut. Dort wurde der Ethernet-Frame um CF erweitert, was ebenfalls im Listing 5.2 zu sehen ist. Zuerst muss der Ethertype definiert werden, der das Format bzw. das Protokoll zur Interpretation des Datenblocks beschreibt. Im Anschluss folgt der DEI, der bei Ethernet Switches immer auf ‚0' gesetzt sein sollte, damit er an einen ungetaggten Port ausgegeben werden kann. Anschließend müssen die Prioritäten angelegt werden. Es kann zwischen sieben verschiedenen Prioritätsklassen mit aufsteigender Priorität gewählt werden. Bei den Prioritäten wurde $bitwise\ shift$ benutzt, um die Prioritätsklassen darzustellen. Anschließend wurde am Ende die VLAN-ID festgelegt, die dazu dient, dass nur Frames mit der gleichen VLAN (ID) im Netzwerk miteinander kommunizieren können.

```
#define VLAN_TAG 0x8100
#define ETHER_TYPE 0x0800    //IPv4

typedef struct
{
  uint16_t priority_DEI_VlanID;
  uint16_t ethertype;
}__attribute__((__packed__)) vlan_header;

#define VLAN_HEADER_PRIORITY_SHIFT    13

/*Priorisierungslevel
 * 0 (niedrigste)  BK  Hintergrund (Background)
 * 1   BE  Best Effort
 * 2   EE  Excellent Effort
 * 3   CA  Kritische Anwendungen (Critical Applications)
 * 4   VI  Video, < 100 ms Verzögerung
 * 5   VO  Sprache (Voice), < 10 ms Verzögerung
 * 6   IC  Internetwork Control
 * 7 (höchste)   NC  Network Control
 */
#define VLAN_HEADER_PRIO_0        0
#define VLAN_HEADER_PRIO_1        1
```

```
#define VLAN_HEADER_PRIO_2        2
#define VLAN_HEADER_PRIO_3        3
#define VLAN_HEADER_PRIO_4        4
#define VLAN_HEADER_PRIO_5        5
#define VLAN_HEADER_PRIO_6        6
#define VLAN_HEADER_PRIO_7        7

#define VLAN_HEADER_ID    2
```

Listing 5.1: VLAN-Header

```
//802.1Q Frame Aufbau in der Bridge
vlan_head = (vlan_header*) &frame.data[i];
vlan_head->ethertype = htons(ETHER_TYPE);
vlan_head->priority_DEI_VlanID = 0;
vlan_head->priority_DEI_VlanID |= VLAN_HEADER_PRIO_3
<< VLAN_HEADER_PRIORITY_SHIFT;
vlan_head->priority_DEI_VlanID |= VLAN_HEADER_ID;
vlan_head->priority_DEI_VlanID =
htons(vlan_head->priority_DEI_VlanID);
i += sizeof(vlan_header);
```

Listing 5.2: 802.1Q-Frame-Aufbau

Der oben vorgestellte Switch der Firma Extreme Networks muss konfiguriert werden, um priorisierten Traffic zu ermöglichen. Dazu muss der Switch mit der seriellen Konsole verbunden werden. Es wurde dabei *minicom* als Terminal-Emulator verwendet. Im Anschluss wurden die Verbindungseinstellungen geändert. In diesem Fall sieht die Konfiguration folgendermaßen aus: – 9600 bps – no parity – 8 data bits – 1 stop bit. Als nächstes wurde der AVB-Traffic abgeschaltet und ein neuer VLAN erstellt (Listing 5.3[1]). Mit dem nächsten Schritt wurde der VLAN-TAG gesetzt (Listing 5.3[2]). Anschließend wurden die jeweiligen Ports zugewiesen sowie die Frames als untagged oder tagged markiert (siehe Listing 5.3[3]). Als nächstes muss ein QoS-Profile angelegt werden für die gewünschten Prioritäten mit dem Befehl (Listing 5.3[4]) . Anschließend überträgt man die Konfiguration entweder an die jeweiligen Ports, die frei wählbar sind, oder an das vorhandene VLAN. Vorteil bei VLAN ist, dass die Ports vorkonfiguriert sind und nur die Ports, die VLAN-fähig sind, mit QoS gekennzeichnet werden. Mit dem Befehl $show\_port < port > qosmonitor$ können die QoS beobachtet werden. Dem Switch ist es jetzt möglich, priorisierten Traffic zu verarbeiten und weiterzuleiten.

```
1. create_vlan < name >

2. configure_vlan < name > tag < zahl >

3. configure_vlan < name > add_ports < 1 - 3 > tagged.

4. create_qosprofile < QP1||QP2||QP3||QP4||QP5||QP6||QP7||>
```

Listing 5.3: Konfigurationsbefehle des Summit-Switchs

5.3 Konfiguration

In diesem Abschnitt geht es um die Konfiguration der Bridges. Es wird hier nur die Konfiguration der Bridge des Lenkrades aufgeführt. In $config\_demonstrator\_main.h$ wird über Konstanten festgelegt, mit welchen Konfigurationsdateien die Bridge konfiguriert wird. Mit der Konstante $BRIDGE\_STEER$ wird die Zugehörigkeit der Bridge zum Lenkrad festgelegt und über die Konstante $CAN\_VIRTUAL$ wird das gewünschte CAN-Modul gewählt.

6 Qualitätssicherung

In diesem Kapitel werden Tools zur Erfüllung der Anforderungen der Qualitätssicherung vorgestellt. Des Weiteren werden generelle Funktionen der Implementierung getestet.

6.1 WireShark

Über den Ethernet-Port der Bridge wird mit einem Test Access Point (TAP) ein Laptop angeschlossen (siehe Abbildung 6.1), auf dem ein Packet-Sniffer installiert ist. Als Sniffer kommt WireShark zum Einsatz. WireShark zeigt bei einer Aufnahme sowohl die Protokoll-Header als auch den transportierten Inhalt an. Es erfasst den Netzwerkverkehr auf Paketbasis und stellt die Paketdaten detailliert dar. Dies geschieht entweder während oder nach der Aufzeichnung des Datenverkehrs einer Netzwerkschnittstelle. Dabei werden die Daten überschaubar mit dem entsprechenden, auf die jeweiligen Protokolle angepassten Filter aufbereitet. Dadurch kann der Inhalt der mitgeschnittenen Pakete betrachtet oder nach diesem gefiltert werden. Des Weiteren können mithilfe von WireShark Statistiken zum Datenfluss oder über spezielle Filter erstellt werden [19].

6.2 CANalyzer

Bei dem CANalyzer handelt es sich um ein Software-Analysewerkzeug für serielle Bussysteme der Firma Vektor, das in diesem Testsystem auf einem Notebook installiert ist. Neben dem reinen Bus-Monitoring bietet diese Software auch die Möglichkeit, selbst frei-konfigurierbare Nachrichten zu senden wie z. B. eine Nachricht mit der ID 0x200 sowie der dazugehörigen Payload. Der CANalyzer verfügt über zwei Anschlüsse an den CAN-Bus, einen RX- und einen TX-Kanal, um parallel Nachrichten zu empfangen und zu versenden. Dadurch ist die durchgängige Überwachung des Busses möglich [20].

6.3 Umwandeln von CAN Nachrichten

In diesem Abschnitt soll die Umwandlung der CAN-Nachrichten zu Control Flow-Nachrichten dargestellt werden. Der Messaufbau ist auf der Abbildung 6.1 zu sehen.

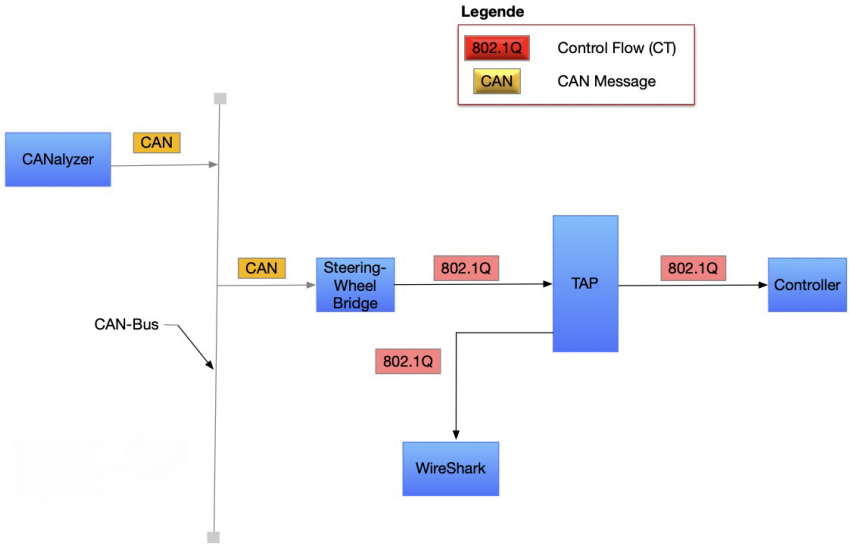

Abbildung 6.1: Messaufbau für QS von Can2Eth-Nachrichten

Für den Test der Umwandlung von CAN-Nachrichten zu Ethernet-Nachrichten wurde ein einfacher Versuchsaufbau gewählt, der die Anzahl der Komponenten im Testsystem gering hält. Deshalb wurde zum Beispiel auf die Verwendung des Summit-X440-Switches verzichtet, um Fehler in der Konfiguration auszuschließen.
In diesem Testszenario wurden nur zwei Komponente verwendet: die Bridge des Lenkrads sowie der Controller. Der CANalyzer wurde an den CAN-BUS angeschlossen. Mit dem Tool CANalyzer wurde manuell eine CAN-Nachricht mit der ID 0x200 erstellt und auf den CAN-BUS gelegt. Da der CANalyzer über zwei Kanäle verfügt (siehe Kapitel 6.2), kann geprüft werden, ob die generierte CAN-Nachricht auf dem CAN-BUS zu sehen ist. Die Steering-Wheel-Bridge nimmt die CAN-Botschaft vom CAN-BUS auf und wandelt diese in eine CF-Nachricht um. Anschließend versendet die Bridge den CF über einen Test Access Point (TAP) zum Controller. An der TAP ist ein Laptop mit dem Sniffer-Tool WireShark angeschlossen. In WireShark wird der gesendete CF betrachtet (siehe dazu Abbildung 6.2). In der Abbildung ist zu sehen, dass der 802.1Q-Frame-Aufbau korrekt übermittelt wurde. Des Weiteren ist ablesbar, dass die Prioritäten sowie die VLAN-ID richtig eingetragen wurden. Die mit dem WireShark empfangenen Daten

bestätigen, dass dieses CF-Paket erfolgreich von der Bridge weitergeleitet wurde. Anschließend wurde mit dem Debugger festgestellt, ob der Frame an der Bridge angekommen ist.

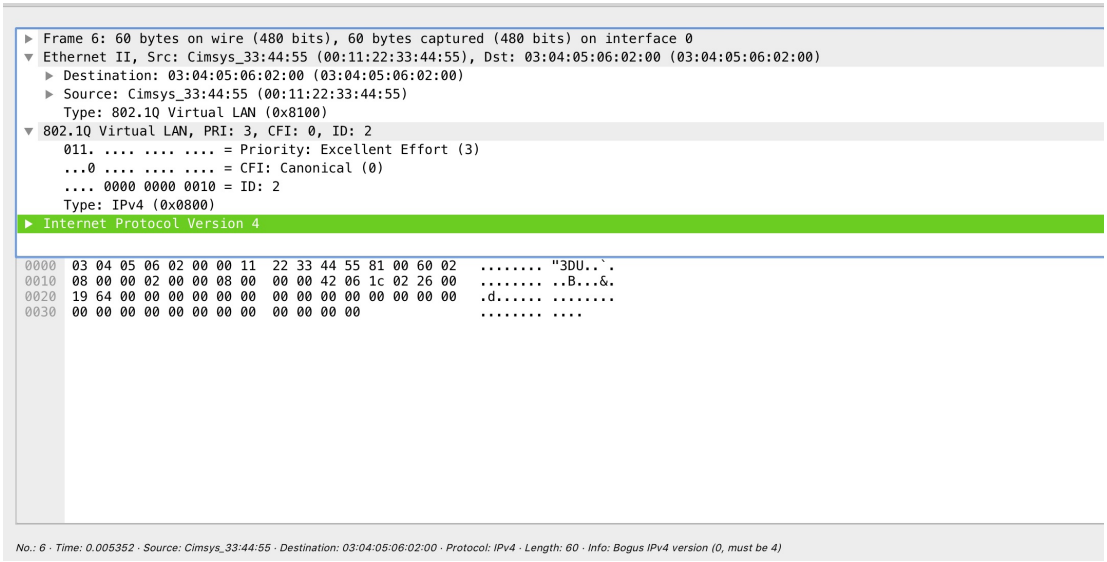

Abbildung 6.2: WireShark-Auszug

6.4 Umwandeln von Ethernet Nachrichten

In diesem Kapitel geht es um die Umwandlung der CF- zu CAN-Nachrichten. Der Messaufbau gliedert sich wie in Abbildung 6.3 dargestellt. In diesem Testfall wurde auch auf die Summit-X440-Switche verzichtet und das Testsystem mit lediglich zwei Komponenten klein gehalten. Um zu überprüfen, ob der CF korrekt in eine CAN-Botschaft umgewandelt wird, sendet der Controller einen 802.1Q-Frame an die Bridge des Lenkrads.

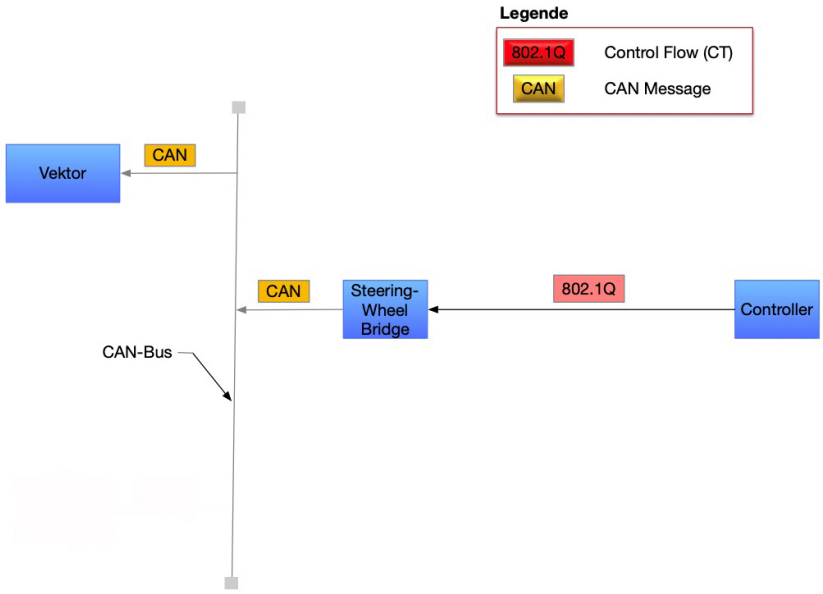

Abbildung 6.3: Messaufbau für QS von Eth2Can-Nachrichten

Die Bridge des Lenkrads ist an einem CAN-BUS angeschlossen. Das Messtool von Vektor wird ebenfalls an die CAN-BUS-Leitung angeschlossen, um die Umwandlung der CF-Nachrichten sowie die Ablegung auf den CAN-BUS nachzuprüfen. Die mit dem CANalyzer empfangenen Daten (siehe Abbildung 6.4) bestätigen, dass die CF-Pakete mit der ID 0x200 korrekt umgewandelt und von der Bridge erfolgreich weitergeleitet wurden. Es wurde in diesem Messaufbau auf WireShark verzichtet, da bereits im vorherigen Schritt festgestellt wurde, dass die Bridge CF-Nachrichten weiterleitet.

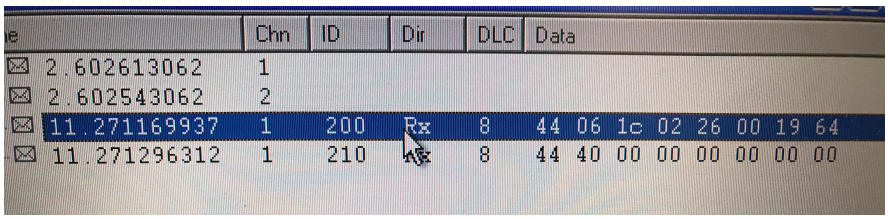

Abbildung 6.4: CAN-BUS-Auszug

7 Evaluierung

In diesem Kapitel werden die Hardware für die folgenden Messungen sowie der Messaufbau mit den dazugehörigen Komponenten detailliert vorgestellt. Des Weiteren wird in diesem Kapitel die analytische Berechnung der Bandbreite sowie der Latenzen von Control-Flow- und Cross-Traffic-Kommunikation durchgeführt. Die Berechnung der Bandbreite dient der späteren Überprüfung der gemessenen Werte. Hierfür sollen unterschiedliche Testszenarien erstellt und gemessen werden, um die Messungen anhand der Ergebnisse zu diskutieren.

7.1 Oszilloskop Tektronix MS04054B

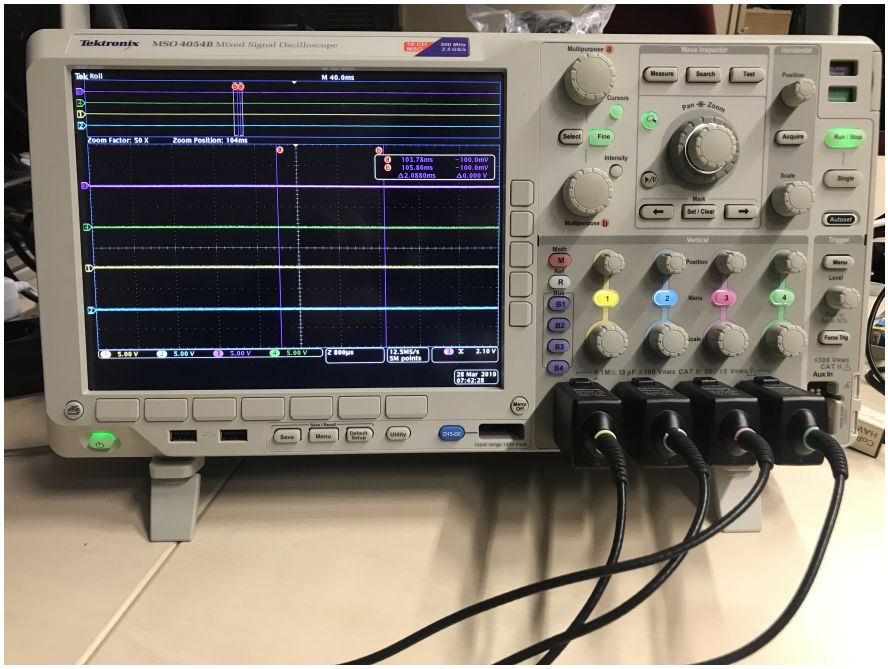

Abbildung 7.1: Tektonix MS04054B. (Quelle[3]: Tektronix. INC.)

Die nachfolgenden Messungen wurden mit dem Tektronix-Mixed-Signal-Oszilloskope der Serie MS04054B [3] durchgeführt. Mit diesem Oszilloskop können elektrische Schaltungen

überprüft, debuggt und charakterisiert werden. Es können bis zu vier analoge sowie 16 digitale Signale mit einem einzelnen Instrument analysiert werden. Um die sich schnell ändernden Signaldetails zu sehen, beträgt die Bandbreite bis zu 500 MHz und es findet eine mindestens 2,5-fache (2,5 GS/s) Überabtastung auf allen Kanälen statt, die für die erforderliche Leistung sorgen. Der MS04054B bietet eine umfangreiche Aufzeichnungslänge von 20 Mio. Punkten (Speichertiefe) auf allen Kanälen um unter Beibehaltung feinster Zeitauflösung lange Fenster von Signaltätigkeiten zu erfassen. Für die durchgeführten Messungen wurden passive Tastköpfe (TPP0502) genutzt.

7.2 Raspberry Pi 3 B+

Beim Raspberry Pi [21] handelt es sich um einen Single-Board-Computer, der für unterschiedliche Aufgaben genutzt werden kann. Er besitzt eine CPU, die mit 1,4 GHz getaktet ist, vier USB-Ports, 1 GB RAM und WLAN IEEE 802.11ac. Für die vorliegende Arbeit ist des Weiteren auch der Gbit-LAN-Port mit einem maximalen Durchsatz von 300 Mbit/s relevant.

7.3 Messaufbau

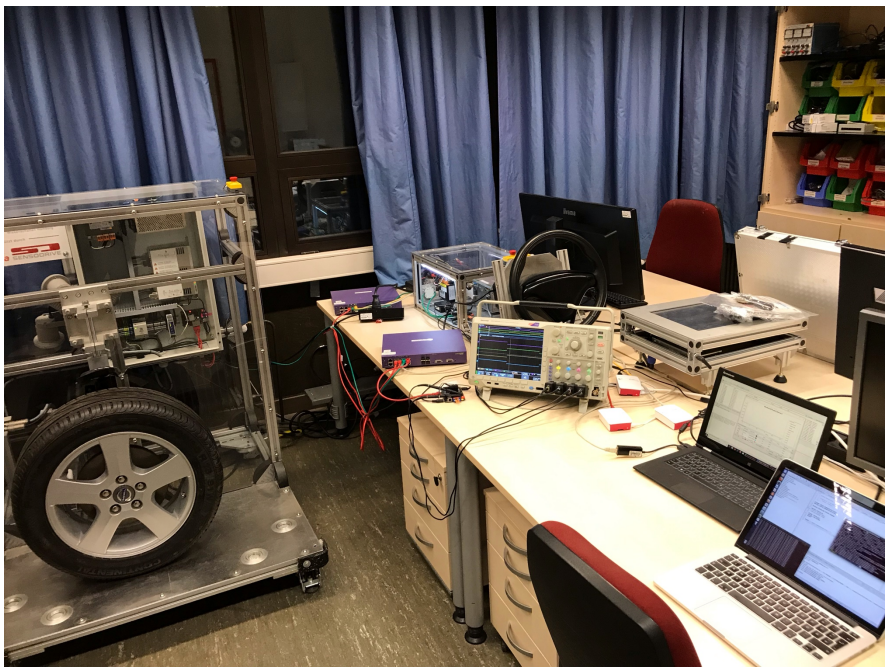

Abbildung 7.2: Messaufbau des Demonstrators.

Das Ziel der Messung war die Bestimmung der Latenz des CF zwischen dem Controller und dem Lenkrad. Des Weiteren sollte die Latenz mit dem Zuführen von Cross-Traffic ermittelt werden um Veränderung der Delays festzustellen, wozu der CF priorisiert verschickt wurde. Durch die Messungen wurden die Latenzen ermittelt und daraus der Jitter berechnet.
Der Messaufbau für den Demonstrator sieht wie folgt aus (siehe dazu Abbildung 7.2 und für die schematische Darstellung Abbildung 7.3):
Um Messfehler durch Paketverluste zu vermeiden, wurde eine Sequenznummer eingeführt. Für die Sequenznummer wurde eine Variable uint32 _t SQN erstellt. Beim Versenden eines Frames in der hal _ethernet.c mit dem Identifier ID 0x200 wird die SQN überprüft. Wenn der Wert bei 0 liegt, wird die entsprechende GPIO ausgelöst. Es wurde dafür eine Modulo-Funktion in bridge _can.c (bridge _can2eth) erstellt, die ab einem Wert von 32 die SQN wieder auf 0 setzt. Die Modulo-Funktion sieht wie folgt aus: ***SQN = (SQN + 1) % 32***.
Beim Empfangen wird in der bridge _can.c (bridge _can2eth) ebenfalls eine Variable uint32 _t SQNR erstellt erstellt. Mit memcpy wird aus der Payload des aktuellen Frames die Sequenznummer in die Variable SQNR kopiert. Wenn die Sequenznummer 0 ist, wird die entsprechende GPIO gesetzt. So kann es durch Paketverluste zu keinen Fehlmessungen kommen, da die jeweiligen Pakete einander mit der Sequenznummer zugeordnet werden können.
Die passiven Tastköpfe des Oszilloskops Tektronix MS04054 wurden am Controller an die General Purpose IOs (GPIOs) 9 und 13 und die anderen beiden Tastköpfe am Lenkrad ebenso an die GPIOs-9- sowie 13-Pins angeschlossen. Mit dem GPIO-9-Pin wird den Paketen die entsprechende Sequenznummer zugeordnet. Beim Controller wird der GPIO-13-Pin bei jedem Versenden eines Frames auf high oder low gesetzt. Beim Lenkrad wird lediglich beim Empfang eines Paketes der GPIO-13-Pin in der Bridge auf high oder low gesetzt. Der GPIO 9 am Lenkrad ist identisch zu dem des Controllers und dient nur zur Zuordnung der Sequenznummer.

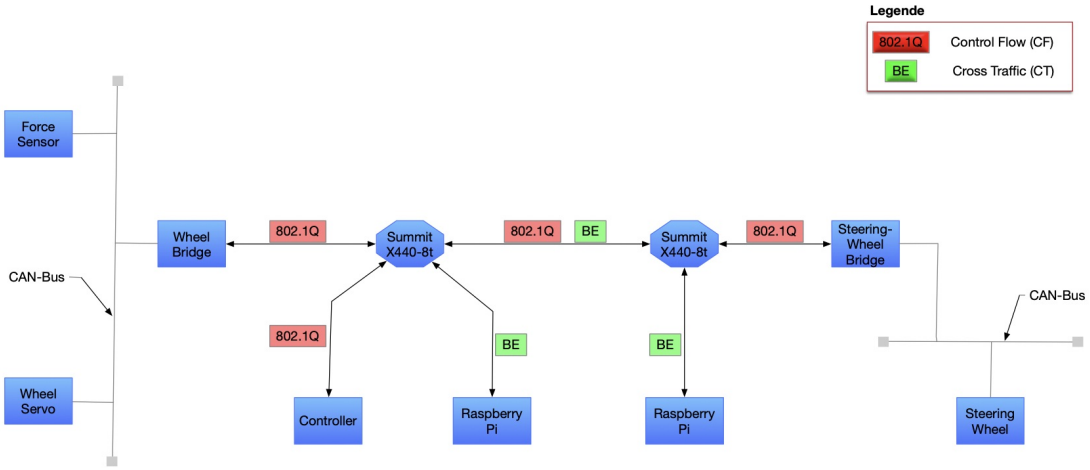

Abbildung 7.3: : Messaufbau des Demonstrators in schematischer Darstellung

Es wurden zwei Raspberry Pis an das Netzwerk für die Cross-Traffic-Generierung, angeschlossen. Die beiden Raspberry Pis schicken sich gegenseitig UDP-Pakete, um das Netzwerk zu belasten. Zwei Summit X440 der Firma *Extrem Networks* wurden als Switche eingesetzt. Diese ermöglichen den CF priorisiert zu verschicken. Es wurden insgesamt drei Messungen durchgeführt: vom Controller zum Lenkrad sowie zurück und vom Controller zum Rad. Die Messung vom Controller zum Rad diente der Ermittlung des Processing Delays der Switche. Bei sämtlichen Messungen war der CT auf die niedrigste Priorität festgelegt, nur beim CF wurde die niedrige Priorität in eine höhere umgewandelt, um Unterschiede in der Laufzeit festzustellen. In Folgenden werden nur die gemessenen Ergebnisse vom Controller zum Lenkrad mit der ID 0x200 präsentiert. Es wurde auf die Messungen vom Controller zum Rad verzichtet, da es sich identisch mit dem des Lenkrades verhält und dadurch keinen Mehrwert bietet.

7.4 Messungen zur Sende-und Empfangsverzögerung des Hilscherboard

Um die späteren Messergebnisse besser nachvollziehen zu können, muss zuerst festgestellt werden, wie lange ein Frame aus dem Stack bis zum Layer 1 braucht, bevor er aus dem Hilscherboard versendet wird. Um den Sendevorgang nachzuvollziehen sowie die Zeit festzulegen, wurde zwischen dem Controller und dem Switch ein Test Access Point (TAP) angeschlossen. An die TAP wurde auch der Oszilloskope über den Ethernet-Link angeschlossen. Dadurch kann gemessen werden, wann der Frame beim Versenden auf den physikalischen Link gelegt

wird (siehe Abbildung zum Messaufbau 7.4). Die Sendeverzögerung $T\_transmit$ liegt bei 2,7 µs, bis der Frame auf die physikalische Leitung gelegt wird (siehe Abbildung 7.5). Es wurden jeweils zehn Messungen durchgeführt.

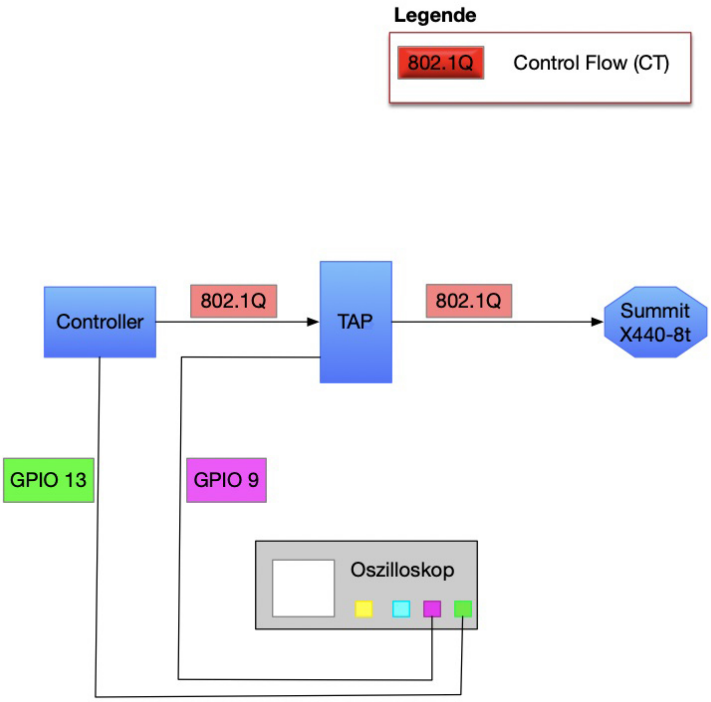

Abbildung 7.4: Messaufbau der Sendeverzögerung

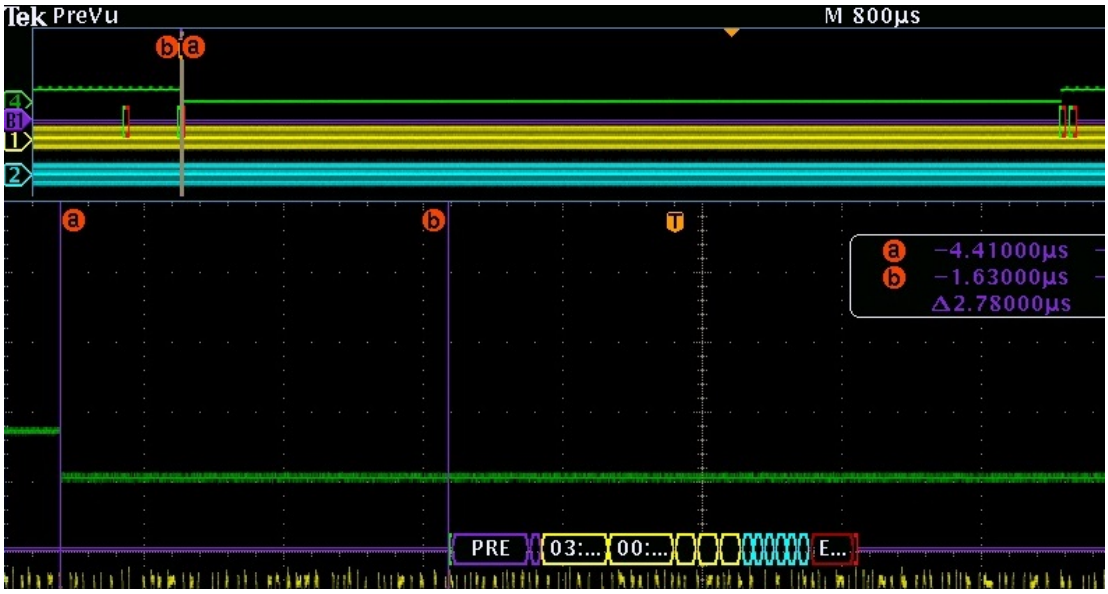

Abbildung 7.5: Sendeverzögerung des Hilscherboards

In Abbildung 7.5 ist der versendete Frame mit seinem Frame-Aufbau zu erkennen. Der GPIO-13-Pin ist in der Grafik als grüne Linie dargestellt und wird bei jedem Versenden einer Nachricht mit der ID 0x200 auf high oder low gesetzt wird (mehr dazu in Kapitel 7.3). Die lilafarbige Linie stellt den Ethernet-Link dar, auf dem der Frame zu sehen ist. Der Frame wurde versendet und dadurch der GPIO-13-Pin gesetzt, jedoch wurde der Frame erst nach 2,7 µs auf dem Link sichtbar. Dadurch konnte eine Sendeverzögerung von 2,7 µs festgestellt werden.

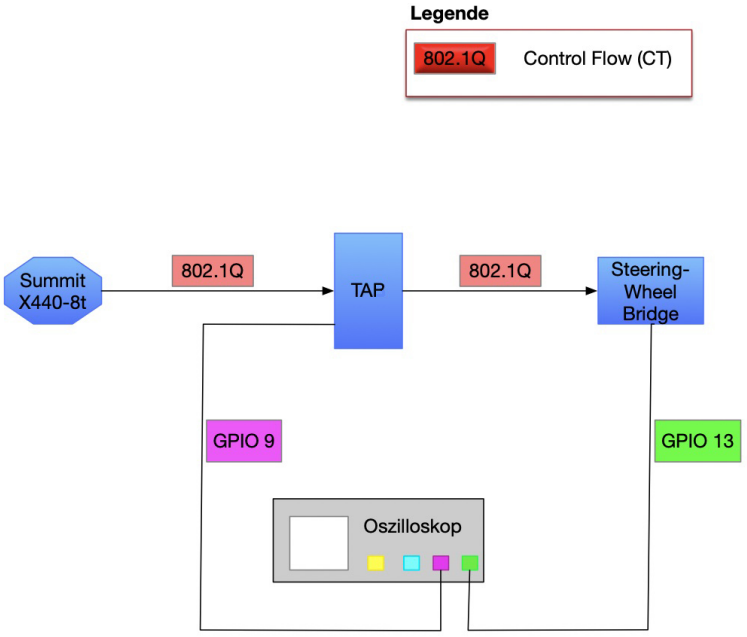

Abbildung 7.6: Messaufbau der Empfangsverzögerung

Das gleiche Verfahren wurde auch beim Empfänger eingesetzt (siehe dazu Abbildung 7.6 zum Messaufbau), hier allerdings auf der Lenkradseite, um die Versanddauer eines Frames von der Hard- bis zu der Software in der Bridge zu messen.
In Abbildung 7.7 sind die empfangenen Frames als kleines Rechteck dargestellt. Der GPIO-13-Pin ist auf dem Bild als grüne Linie zu sehen und wird bei jeder empfangenen Nachricht mit der ID 0x200 auf high oder low gesetzt. Der Ethernet-Link wird in der Grafik durch die lilafarbene Linie gekennzeichnet. Es wird erkennbar, dass die Nachricht zuerst auf dem Link empfangen, der Zustand des GPIO-13-Pins jedoch erst später geändert wurde. Die Empfangsverzögerung $T_{receive}$ liegt bei 29,9 µs, bis der Frame an der Bridge angekommen ist (siehe Abbildung 7.7). Es wurden jeweils zehn Messungen durchgeführt. Im Worst Case benötigte der Frame 65 µs bis zur Bridge.

Abbildung 7.7: Empfangsverzögerung des Hilscherboards

Durch diese beiden Messverfahren wurden die Sende- und Empfangsverzögerungen ermittelt. Die unterschiedlichen Werte erklären sich dadurch, dass bei der Sendeverzögerung in der HAL die GPIO beim Senden gesetzt werden. Dagegen wird bei der Empfangsverzögerung in der Bridge-Software die GPIO beim Empfangen gesetzt. Diese Messung war notwendig, um später die Messergebnisse mit den berechneten Werten vergleichen zu können.

7.5 Berechnungen

In diesem Unterkapitel geht es um die Berechnungen der Bandbreite des Netzwerks sowie der Latenz. Anschließend wird der Jitter aus den Ergebnissen der Latenz ermittelt. Die Berechnung ist notwendig, um die gemessenen Werte für die Plausibilitätsüberprüfung auszuwerten.

7.5.1 Berechnung der Bandbreite

Um die geforderte Bandbreite des Control Flows (CF) zu ermitteln, wird die Bandbreite analytisch berechnet. Diese wird aus $Preamble$ (7 Byte), $StartofFrameDelimiter$ (SFD (1 Byte)),

$Header$ ((14 Byte) und $802.1QTag$ (4 Byte)), $Payload$ (46 Byte), $cyclicredundancycheck$ (CRC (4 Byte)) und $InterpacketGap$ (IFG (12 Byte)) berechnet.

$$\begin{aligned} CF_{Frame\_size} &= (Preamble + SFD) + (Header + 802.1Q) + Payload + \\ &\quad CRC + IFG \\ CF_{Frame\_size} &= (7 + 1Byte) + (14 + 4Byte) + 46Byte + 4 + 12Byte = 88Byte \\ &= 88Byte * 8 \\ &= \underline{704Bit} \end{aligned} \tag{7.1}$$

$$bandwidth_{CF\_Frame} = \frac{CF_{Frame\_size}}{100Mbit/s} = \frac{704Bit}{1000000} = \underline{0,000704Mbit/s} \tag{7.2}$$

Es werden alle 5 ms periodisch Pakete versendet:

$$Frame_{count} = \frac{1s}{5ms} = \frac{1s}{5 * 10^{-}3s} = \underline{200} \tag{7.3}$$

$$\begin{aligned} bandwidth_{CF\_Frame1} &= bandwidth_{CF\_Frame} * Frame_{count} \\ &= 0,000704Mbit/s * 200 \\ &= \underline{0,1408Mbit/s} \end{aligned} \tag{7.4}$$

Für ein Ethernet-Paket, das alle 5 ms versendet wird, benötigt das Netzwerk eine Bandbreite von 0,1408 Mbit/s. Der Controller schickt 5 (M) Nachrichten hintereinander ab:

$$\begin{aligned} bandwidth\_all_{CF\_Frames} &= bandwidth_{CF\_Frame1} * M = 0,1408Mbit/s * 5 \\ &= \underline{0,704Mbit/s} \end{aligned} \tag{7.5}$$

Das System benötigt 0,704 Mbit/s Bandbreite, um die fünf Nachrichten periodisch zu versenden.

7.5.2 Latenz von Control Flow (CF)

Es wurde die Latenz des CFs vom Controller zum Lenkrad gemessen. Um das Transmission Delay zu berechnen, werden eine Bandbreite (R) von 100 Mbit/s sowie eine Länge von CF-Frame (704 Bit) benötigt.

$$T_{transmission} = \frac{CF_{Frame\_size}}{R} = \frac{704Bit}{100Mbit/s} = \underline{7,04\mu s} \tag{7.6}$$

Durch die analytische Berechnung ergibt sich die Verzögerung in den Übertragungskomponenten. Im untersuchten Netzwerk existieren drei Links (L) bis zum Ziel, weshalb das Ergebnis mit drei multipliziert werden muss, um das Propagation Delay zu berechnen.

$$T_{propagation} = T_{transmission} * L = 7,04\mu s * 3 = \underline{21,12\mu s} \tag{7.7}$$

Es werden 18,72 µs benötigt, um über drei Links die Nachricht zu versenden. Hinzu kommen noch die Delays des Hilscherboards beim Versenden $T_{transmit}$ 2,7 µs (gemessen) sowie beim Empfangen $T_{receive}$ 30 µs (gemessen) (siehe hierzu Kapitel 7.4).
Um die Delays der Switche $T_{processing}$ zu ermitteln, wurde die Latenz des CFs zwischen Controller und Rad gemessen. In der Berechnung wird nicht auf die $T_{queuing}$ eingegangen, da die Queue leer ist. Die gemessene Latenz mit einem Switch, zwischen dem Controller und dem Rad, beträgt 49,9 µs.

$$\begin{aligned} T_{processing} &= 49,9\mu s - T_{transmit} - T_{transmission} - T_{transmission} - T_{receive} \\ T_{processing} &= 49,9\mu s - 2,7\mu s - 7,04\mu s - 7,04\mu s - 29,9\mu s = \underline{3,22\mu s} \end{aligned} \tag{7.8}$$

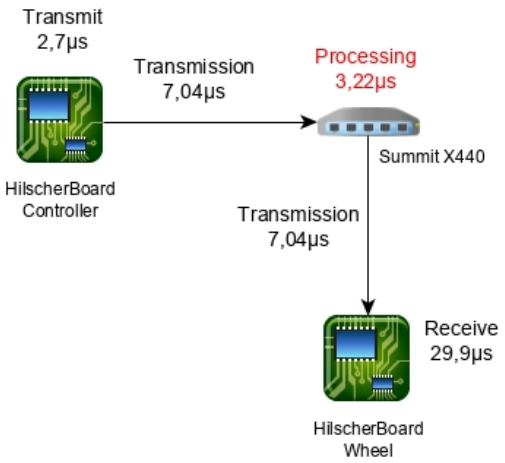

Abbildung 7.8: Processing Delay

Anhand der Abbildung 7.8 wird erkennbar, dass nur eine unbekannte Konstante existiert, nämlich das Processing Delay. Dadurch ist es möglich, den Delay der Switche zu berechnen. Die Processing-Verzögerung, die bei dem Switch auftritt, liegt bei 3,22 µs.

$$T_{CF_{end-end}} = T_{transmit} + T_{processing} + T_{propagation} + T_{processing} + T_{receive} \tag{7.9}$$
$$T_{CF_{end-end}} = 2,7\mu s + 3,22\mu s + 21,72\mu s + 3,22\mu s + 29,9\mu s = \underline{60,76\mu s}$$

Die Latenz des Control Flows (CF) über drei Hops liegt bei 60,76 µs.

7.5.3 Maximale Latenz von CF Nachrichten

In diesem Unterkapitel wird der Worst Case durch den Cross Traffic (CT) berechnet. Mit dem Rapsberry Pi wurden CT-Frames mit einer Payload von 1518 Byte generiert. Um die maximale Latenz von Frames zu berechnen, werden die Bandbreite (R), die maximal Länge von CT (D) 1518 Bit sowie das Interframegap T_{IFG}. benötigt.

$$\begin{aligned} maxCT_{Frame} &= 1518Byte \\ &= 1518Byte * 8 \\ &= \underline{12144Bit} \end{aligned} \tag{7.10}$$

$$\begin{aligned} T_{CF_{max}} = \frac{D}{R} + T_{transmission} + T_{IFG} &= \frac{12144Bit}{100Mbit/s} + 7,04\mu s + 0,96\mu s \\ &= \underline{129,44\mu s} \end{aligned} \tag{7.11}$$

Damit ist die Berechnung des Worst Cases eines Schedulingvorgangs von CF mit CT im System möglich. Dafür müssen die Sende- und Empfangsverzögerungen sowie die Switche (processing) mit in die Berechnung aufgenommen werden. In der vorherigen Berechnung von $T_{CF_{max}}$ wurde das T_{IFG} miteinbezogen, weshalb es nicht erneut berücksichtigt werden muss.

$$\begin{aligned} T_{Max\_Latenz} = T_{transmit} + T_{transmission} + T_{processing} + T_{CF_{max}} + T_{processing} \\ + T_{transmission} + T_{receive} \end{aligned} \tag{7.12}$$
$$\begin{aligned} T_{Max\_Latenz} &= 2,7\mu s + 7,04\mu s + 3,22\mu s + 129,44\mu s + 3,22\mu s + 7,04\mu s + 65\mu s \\ &= \underline{217,66\mu s} \end{aligned}$$

Durch die analytische Berechnung konnte der Worst Case des Control Flows ermittelt werden. Dafür wurde das Delay eines Switches berechnet, indem ein einfacher Messaufbau mit

einem Controller, einem Switch und einem Lenkrad gewählt wurde. Dadurch konnte analytisch ermittelt werden, welche Latenz im schlimmsten Fall bei CF mit CT auftreten kann.

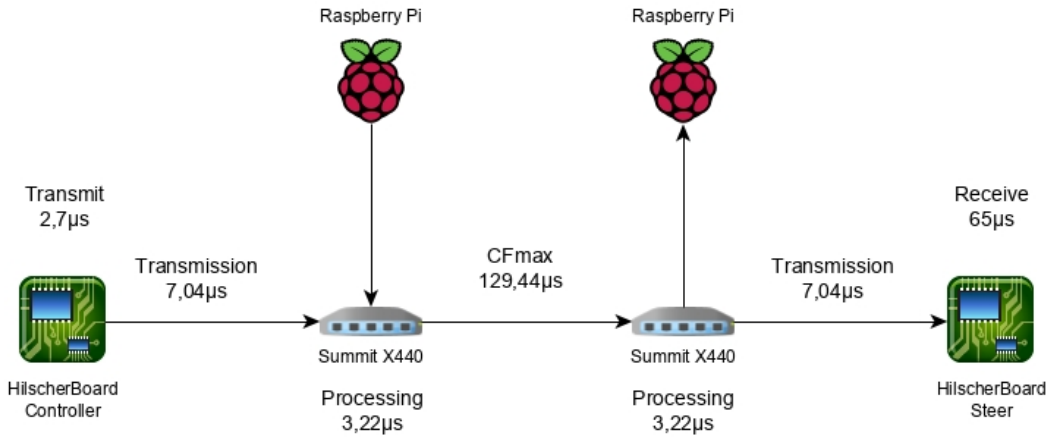

Abbildung 7.9: Maximale Verzögerung von CF mit CT.

In Abbildung 7.9 wird die Berechnung bei 7.12 visuell dargestellt. Anhand dieser kann die Berechnung besser nachvollzogen werden. Die maximale Latenz eines CF-Frames mit CT liegt bei 217,66 µs.

7.6 Messungen ohne Prioritäten

In den folgenden Szenarien wurde die Priorität der CF gleich gesetzt mit der der CT (0), damit alle Pakete durch den Switch gleichbehandelt werden. Die erste Messung wurde ohne Cross-Traffic durchgeführt. Da kein anderer Traffic im Netzwerk stattfand, sind die Prioritäten in diesem Fall zu vernachlässigen. Es wurde vom Controller zum Lenkrad die ID 0x200 gemessen. Die maximale Latenz betrug 65,2 µs und die minimale Latenz 62 µs. Aus der maximalen und minimalen Latenz wurde der Jitter berechnet. In diesem Fall lag er bei 3,2 µs (siehe Tabelle 7.2). Dazu wurden zwanzig Messungen durchgeführt und in eine Tabelle 7.1 eingetragen.

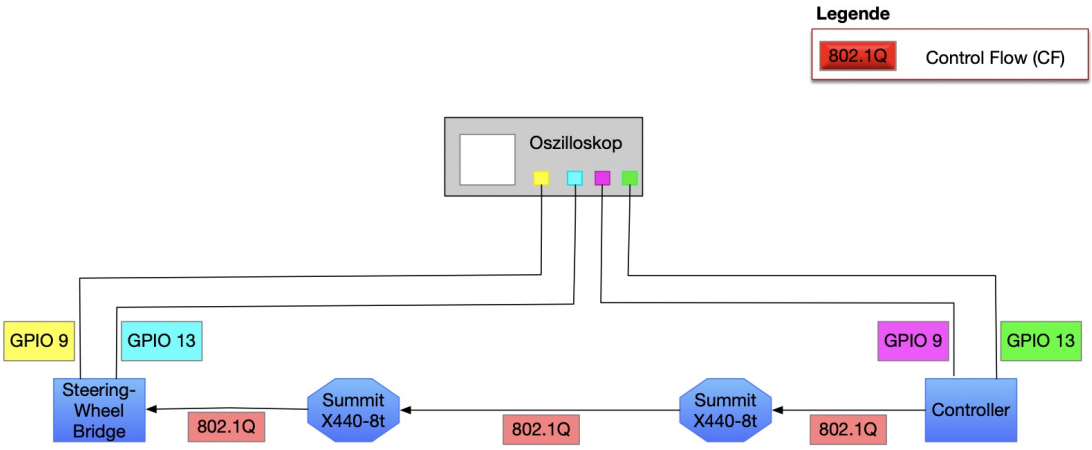

Abbildung 7.10: Messaufbau von Controller zum Lenkrad ohne CT.

Die folgende Messung wurde mit dem Oszilloskop durchgeführt (siehe hierfür Abbildung 7.11). Die beiden lilafarbenen und grünen Messpunkte wurden auf der Controller-Seite angeschlossen. Der lilafarbene Messpunkt steht für die Sequenznummer und der grüne für die ID 0x200, die zwischen high und low beim Versenden wechselt. Die gelbe und der türkisfarbene Punkt stehen auf der Empfängerseite bereit. Gelb stellt die Sequenznummer dar und Türkis steht für jedes empfangene ID 0x200-Paket (siehe schematische Darstellung 7.10). Wie aus der Abbildung 7.11 erkennbar wird, beträgt die Übertragungszeit eines Paketes mit der ID 0x200 um die 63.2 µs beim Versand ohne Cross-Traffic. Es wurden für die Messungen weitere 19 Tests durchgeführt, um den Jitter zu ermitteln. Diese können der Tabelle 7.1 entnommen werden. Die gemessene Latenz stimmt mit der vorherigen Berechnung überein.

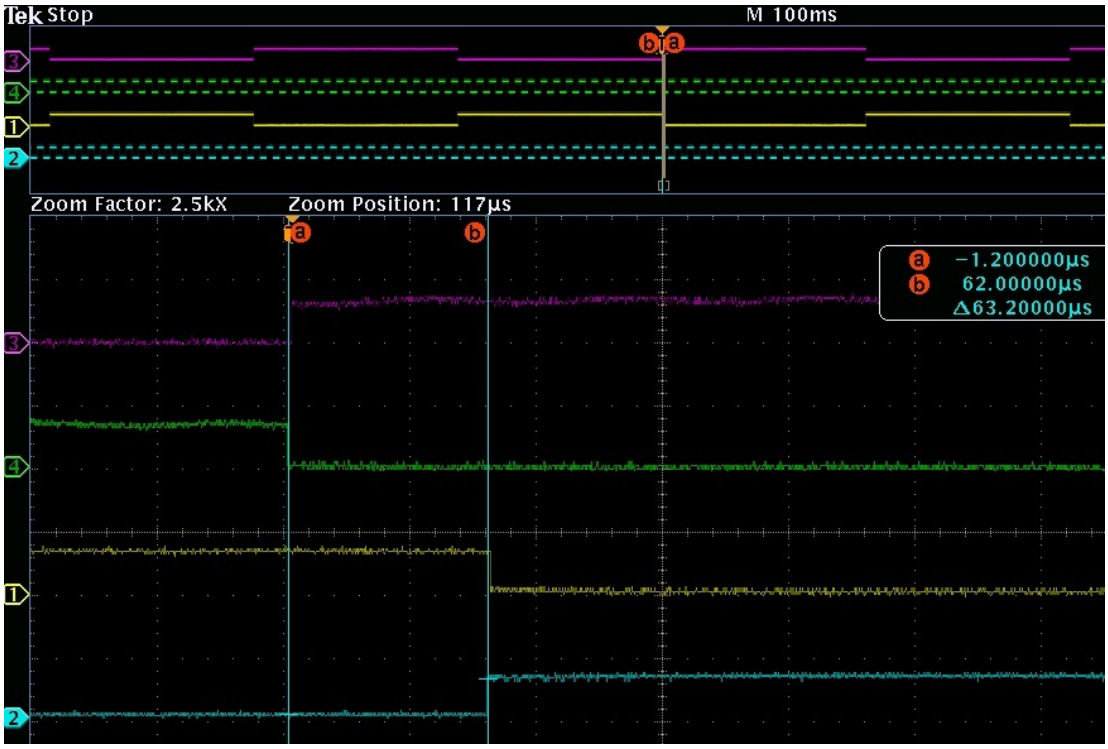

Abbildung 7.11: Messung von ID 0x200 von Controller zum Lenkrad

7.6.1 Messung mit Cross Traffic (CT)

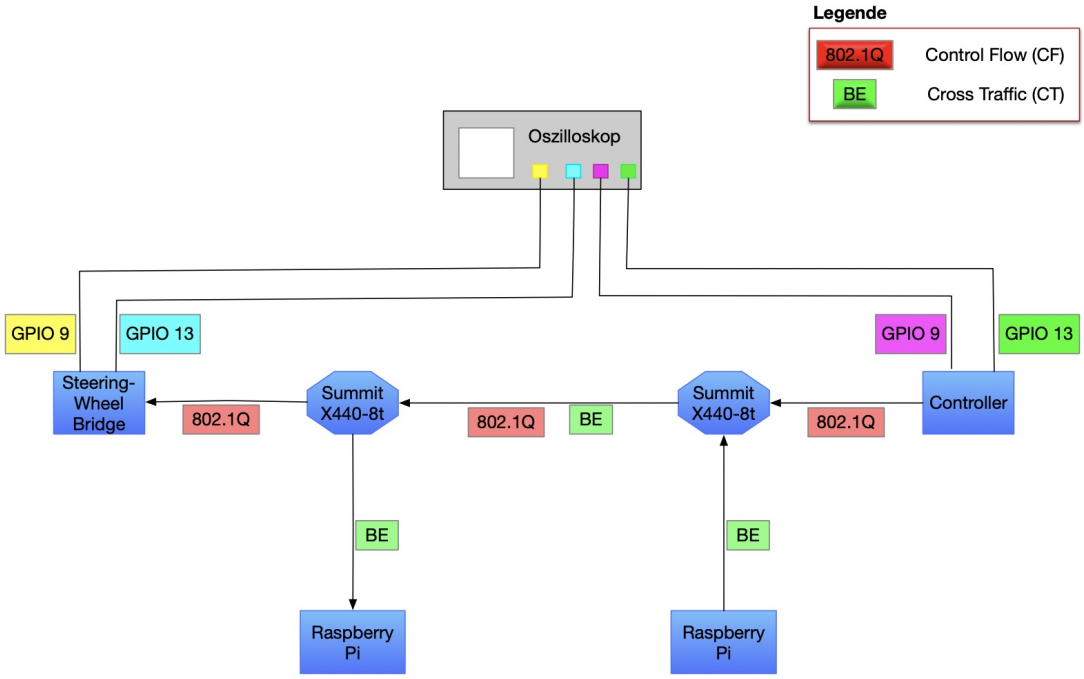

Abbildung 7.12: Messaufbau von Controller zum Lenkrad mit CT

Der gleiche Messaufbau wurde auch im nächsten Fall angewendet, diesmal allerdings mit hohem CT-Anteil und weiterhin mit der niedrigsten Priorität wie beim CF-Traffic (siehe Abbildung 7.12 zum Aufbau). Bei diesen Messungen kam ein Raspberry Pi zur CT-Generierung zum Einsatz. Die Bandbreite wurde voll ausgelastet. Der Raspberry Pi überlastete das Netzwerk mit einer maximalen Übertragung von 98 Mbit/s, damit Pakete verworfen wurden. Wie in Abbildung 7.13 an der türkisfarbenen Linie zu erkennen ist, werden auch Pakete des CF-Traffics durch den hohen Anteil des CT verworfen. Wenn keine CF-Pakete verworfen werden, ist die türkisfarbene Linie identisch mit der grünen. Außerdem ist zu sehen, dass die Latenzen überproportional angestiegen sind (siehe Tabelle 7.1).

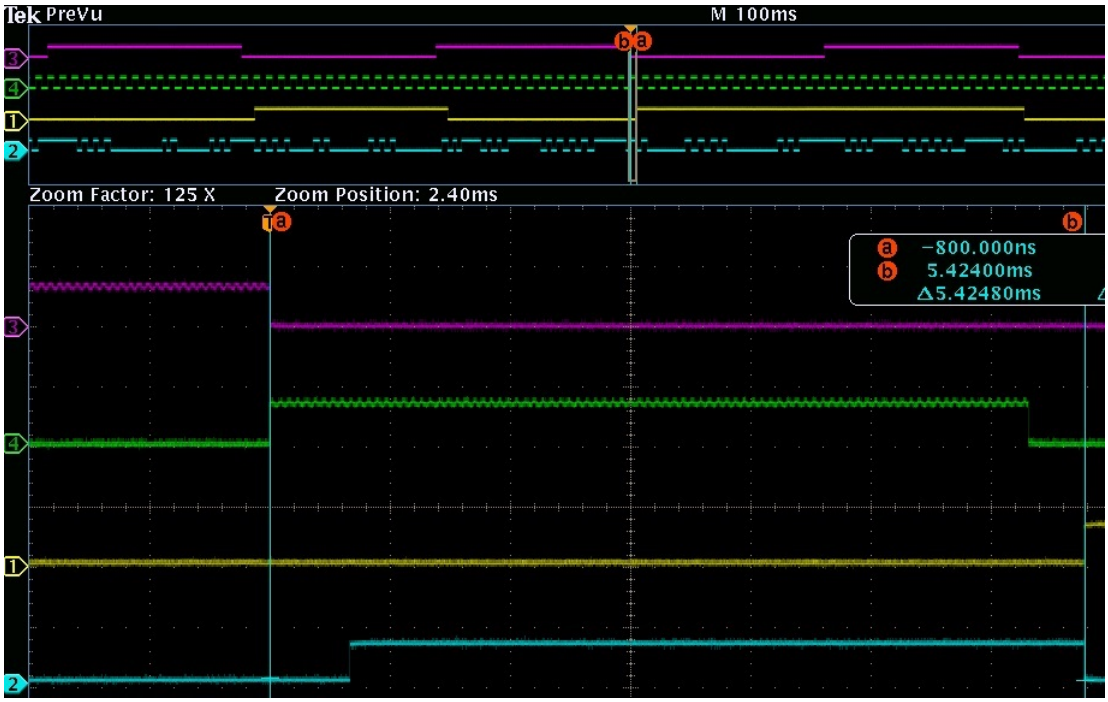

Abbildung 7.13: Messung von ID 0x200 mit CT von Controller zum Lenkrad

Mit dem Zuführen von Cross-Traffic (CT) steigt die maximale Latenz bis zu 5400 µs und die minimale auf 1450 µs. Anhand der Abbildung 7.13 wird erkennbar, dass die Latenz durch den CT angestiegen ist. Dieser Wert ist im Vergleich zu vorher hoch. Durch die unterschiedlichen Latenzen steigt der Jitter an.

Controller ⇒ Lenkrad	**Controller ⇒ Lenkrad mit CT**
63,2 µs	2540 µs
63,2 µs	3780 µs
62 µs	5420 µs
63,2 µs	1490 µs
64,4 µs	2890 µs
64,4 µs	2890 µs
64,4 µs	4370 µs
63,6 µs	1610 µs
64 µs	5420 µs
64,4 µs	2330 µs
64,8 µs	3530 µs
64,8 µs	3270 µs
65,2 µs	2900 µs
62,8 µs	1540 µs
62,8 µs	3200 µs
62,8 µs	4100 µs
63,2 µs	3530 µs
64,8 µs	2170 µs
62,8 µs	2610 µs
64,2 µs	3200 µs

Tabelle 7.1: Messungen ohne Prioritäten.

	ohne_CT	mit_CT
Jitter	3,2 µs	3880 µs

Tabelle 7.2: Jitter ohne und mit CT.

Es wurden in diesem Abschnitt Messungen ohne Prioritäten sowie mit oder ohne CT durchgeführt (siehe hierzu die Tabelle 7.1). Bei den Messungen ohne CT waren die Latenzen erwartbar niedrig. Anschließend wurde der CT mit auf die Leitung gelegt, was die Latenzen stark ansteigen ließ. Der daraus resultierende Jitter ergab 3880 µs. Zuvor lag der Jitter ohne CT bei 3,2 µs (Tabelle 7.2). Dadurch ist die Anlage nicht stabil und nicht funktionsfähig, denn mit langen Verzögerungen zwischen Lenkrad und Rad schaukelt sich das System auf. Zudem werden systemkritische Nachrichten verworfen.

7.7 Messung mit Prioritäten

In den folgenden Szenarien wurde die Priorität der CF auf den Wert 3 (Critical Message) gesetzt, damit priorisierte CF-Nachrichten anhand der Prioritätsklassen priorisiert durch den Switch behandelt werden. Die erste Messung wurde ohne Cross-Traffic (CT) durchgeführt. Es wurde dabei vom Controller zum Lenkrad die ID 0x200 gemessen. Der Messaufbau war der gleiche wie zuvor, allerdings ohne Priorisierung. Die maximale Latenz betrug 65,3 µs und die minimale 62,4 µs. Aus der maximalen und minimalen Latenz wurde in diesem Fall ein Jitter von 2,9 µs berechnet. Dazu wurden zwanzig Messungen durchgeführt (siehe hierzu Tabelle 7.3). Es war zu erwarten, dass sich die Ergebnisse sowohl mit als auch ohne Priorisierung ohne den zusätzlichen Cross-Traffic (CT) ähneln.

7.7.1 Messung mit Cross Traffic (CT)

Der bereits in Kapitel 7.6.1 vorgestellte Messaufbau wurde auch hier verwendet. In diesem Fall aber mit der Ausnahme, dass die CF-Pakete mit höheren Prioritäten versehen sind als die CT-Pakete. Wie erwartet, wurden bei diesem Szenario nur CT-Pakete verworfen und keine CF-Pakete. Dies ist an der Abbildung 7.14 sehen: Die türkisfarbene Linie verhält sich gleich mit der grünen Linie. Anhand der Tabelle 7.3 ist zu erkennen, dass in diesem Fall nur die Latenzen der Pakete gestiegen sind (siehe Abbildung 7.14). Die maximale Latenz betrug 223 µs und die minimale 114 µs. Die gemessenen Ergebnisse stimmen mit den berechneten Latenzen überein, wodurch Fehler ausgeschlossen werden können, die bei den Messungen zustande kommen könnten. In diesem Szenario betrug der Jitter 109 µs, wie in der Tabelle 7.4 zu sehen ist. Im Vergleich zur gleichen Messung ohne Prioritäten, mit einem Jitter von 3880 µs, fiel dieser klein aus. Mit dem Jitter von 109 µs kann die Anlage problemlos betrieben werden. Außerdem werden die CF-Pakete priorisiert behandelt und somit werden keine systemkritischen Nachrichten verworfen.

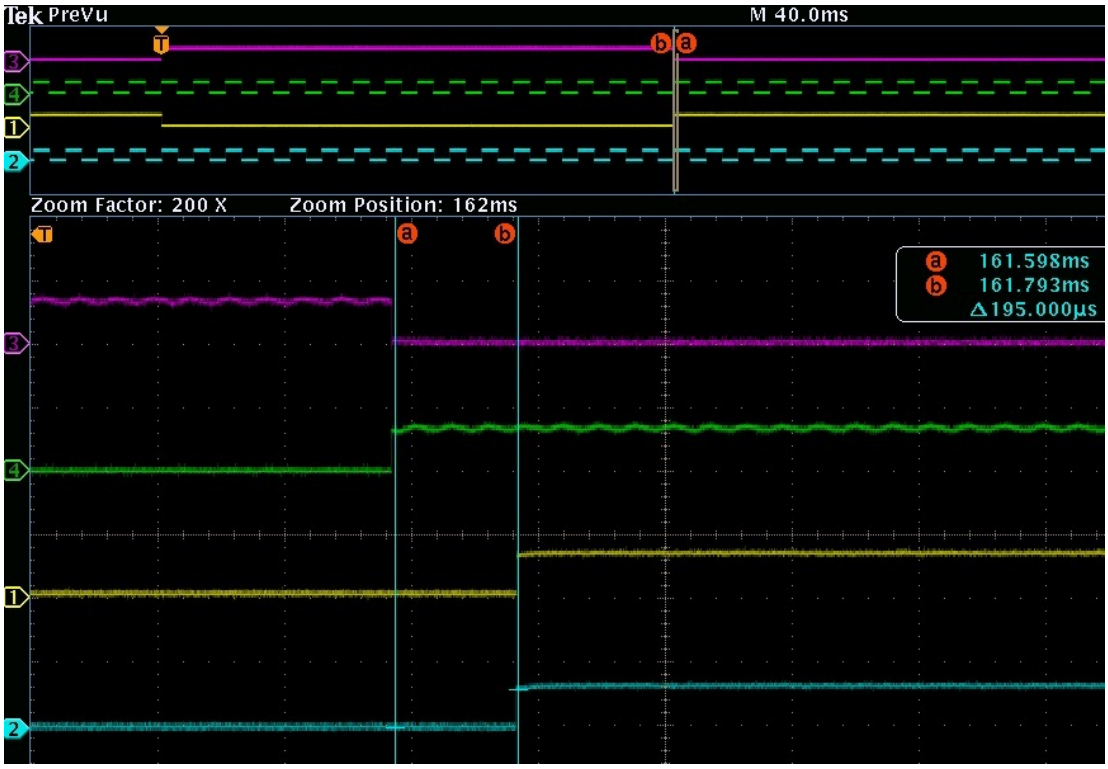

Abbildung 7.14: Messung von ID 0x200 mit CT und Priorität von Controller zum Lenkrad

Controller ⇒ Lenkrad	**Controller ⇒ Lenkrad mit CT**
63,2 µs	130 µs
64 µs	189 µs
63,2 µs	130 µs
65,1 µs	205 µs
63,2 µs	195 µs
63,6 µs	145 µs
63,2 µs	194 µs
62,2 µs	210 µs
62,8 µs	200 µs
63,2 µs	156 µs
62,8 µs	118 µs
63,6 µs	141 µs
63,6 µs	124 µs
63,6 µs	172 µs
62,8 µs	154 µs
62,4 µs	133 µs
63,4 µs	114 µs
65,3 µs	223 µs
63,2 µs	171 µs
63,2 µs	180 µs

Tabelle 7.3: Messungen mit Prioritäten.

	ohne_CT	mit_CT
Jitter	2,9 µs	109 µs

Tabelle 7.4: Jitter ohne und mit CT.

Im letzten Abschnitt wurden mehrere Messungen mit Prioritäten durchgeführt, jeweils mit und ohne CT. Bei den Messungen ohne CT waren die Ergebnisse wie zu erwarten niedrig. Durch das Zuführen von CT stiegen die Latenzen von CF an (siehe Tabelle 7.3). Die gemessenen Ergebnisse stimmen mit den berechneten Latenzen überein, wodurch Fehler ausgeschlossen werden können, die bei den Messungen zustande kommen könnten. Aus den Latenzen wurde der Jitter berechnet. In diesem Szenario beträgt der Jitter 109 µs, wie in der Tabelle 7.4 zu sehen ist. Im Vergleich zu den gleichen Messungen ohne Prioritäten, mit einem Jitter von 3880 µs, fiel

dieser klein aus. Mit dem Jitter von 109 µs kann die Anlage problemlos betrieben werden. Die CF-Pakete werden priorisiert behandelt und somit werden keine systemkritischen Nachrichten verworfen.

8 Fazit und Ausblick

In diesem Kapitel findet die Arbeit zur prototyp-basierten Analyse eines automobilen Kommunikations-Backbones am Beispiel einer Steer-by-Wire-Anwendung ihren Abschluss. Dazu werden die Aussagen der einzelnen Kapitel zusammengefasst, ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten gegeben.

8.1 Zusammenfassung und Fazit

Die Automotive-Anwendungen werden in Zukunft stetig in ihrer Anzahl und Komplexität ansteigen. Bei der Entwicklung moderner Automobile wird zunehmend auf den Einsatz komplexer elektronischer Systeme gesetzt, wie zum Beispiel auf X-By-Wire-Systeme oder auf Komfort- und Informationssysteme. Diese Systeme stellen unterschiedliche Anforderungen an das zugrundeliegende Netzwerk. Sensorbasierte Systeme benötigen eine hohe Bandbreite für die Daten von Kameras und Laserscannern, während sicherheitskritische Anwendungen eine niedrige Latenz mit einem niedrigen Jitter voraussetzen. Die Automobilindustrie versucht, die Menge an unterschiedlichen Bussystemen, und damit auch die Komplexität der Vernetzung, zu reduzieren.
In dieser Bachelorarbeit wird 802.1Q als Alternative zu TTE vorgestellt. In Kapitel 2 wurde das grundlegende Hintergrundwissen zu dieser Arbeit vermittelt. Dazu gehört ein Einblick in Virtual Local Area Networks (VLANs), die in IEEE 802.1Q standardisiert sind [13][14]. Des Weiteren wurde eine knappe Übersicht über TTE gegeben sowie eine kurze Einführung der Begriffe Latenz und Jitter. Anschließend wurden die Anforderungen an das System festgelegt sowie eine Übersicht über das vorhandene System vor der Änderung aufgezeigt. Im Anschluss wurden das vorhandene Konzept der Bridge vorgestellt und der Programmablauf anhand einer visuellen Darstellung erklärt. In Kapitel 5 folgte die Realisierung mit 802.1Q, wozu der 802.1Q-fähige Switch vorgestellt wurde. Im darauffolgenden Kapitel wurde die Implementierung anhand der Qualitätssicherung getestet. Durch diesen einfachen Testaufbau konnte festgestellt werden, dass die Implementierung korrekt und fehlerfrei funktioniert. Im Kapitel Evaluierung wurden die Bandbreite sowie die erwartete Latenz theoretisch berechnet, um diese mit den ermittelten Messwerten zu vergleichen. Der Messaufbau wurde schematisch dargestellt

und erläutert. Dazu wurden Messungen mit und ohne Priorität durchgeführt. Außerdem wurde bei den Messungen CT eingesetzt, der das Netzwerk belasten sollten, um die Einhaltung der Prioritäten zu gewährleisten und die unterschiedlichen Merkmale darzustellen. Alle Ergebnisse der Latenzen wurden in eine Tabelle eingetragen und daraus der Jitter berechnet. Abschließend wurden die gemessenen Werte, wie die Latenzen, mit den theoretisch berechneten Werten geprüft.
Die gemessenen Ergebnisse stimmten mit den berechneten Werten überein. Des Weiteren wurde festgestellt, dass durch die Priorisierung des Control Flows die Hauptanwendung trotz des Zuführens von Cross-Traffic weiterhin funktionsfähig ist. Es wurde festgestellt, dass Pakete ohne eine höhere Priorisierung aus der Hauptanwendung verloren gehen. Bei den Messungen ohne Prioritäten konnte festgehalten werden, dass die Latenzen durch das Zuführen von CT nicht konstant, sondern mit großer Schwankung auftreten. Deshalb entstand durch die unterschiedlichen Latenzen ein zu hoher Jitter, wodurch die Anlage nicht mehr funktionsfähig war. Die Verwendung von Prioritäten löste in diesem Fall das Problem.

Mit Ethernet in Fahrzeugen findet in absehbarer Zeit ein Technologiewechsel von Feldbussen zu modernen Kommunikationsstrukturen statt. Der Infotainment-Bereich wird zunehmend umfassender und profitiert von diesem Wechsel. An dieser Arbeit wird erkennbar, dass für eine kritische Anwendung auch die Priorisierung einsetzbar ist. Doch wenn zahlreiche Kommunikationsprozesse über die hohe Prioritätsklasse laufen, werden die Grenzen überschritten. Diese Technologie kann in Fahrzeugen auch für den Infotainment-Bereich eingesetzt werden, für den eine hohe Bandbreite benötigt wird. Unter anderem werden durch Priorisierung niedrige Jitter-Zeiten erreicht. Der Jitter lag bei 109 µs. Durch das Zuführen von mehr Traffic mit gleicher Priorität stieg der Jitter weiter. Es gibt keine Garantie für eine Echtzeitfähigkeit, aber durch die unterschiedlichen Priorisierungsmöglichkeiten kann mit 802.1Q so ein Verhalten hervorgebracht werden. Daher ist 802.1Q in unterschiedlichen Bereichen einsetzbar, wie zum Beispiel bei sensorbasierten Systemen, bei denen Kameras oder Laserscannern eine hohe Bandbreite fordern. Durch die niedrige Latenz und den daraus entstandenen Jitter ist es möglich, 802.1Q priorisiert in sicherheitskritische Anwendungen zu implementieren.

8.2 Ausblick

Die Arbeiten der CoRE-Arbeitsgruppe werden sich auch in Zukunft mit der weiteren Entwicklung von Time-Triggered Ethernet, Audio/Video-Bridge und TDMA sowie mit 802.1Q beschäftigen. Weiterhin muss auch auf die Sicherheit der Netzwerke Wert gelegt werden, wenn

derartige Technologien in Zukunft in Fahrzeuge eingebaut werden.
Die Ergebnisse dieser Bachelorarbeit können mit Peripheriegeräten vom Demonstrator erweitert werden. Als Peripheriegeräte eignen sich Scheinwerfer oder das Infotainmentsystem. Anstatt mit den Raspberry Pis Cross-Traffic zu erzeugen, waren ursprünglich das Infotainmentsystem sowie ein Notebook dafür vorgesehen. Das Infotainmentsystem versendet ein Video, das am Notebook abgespielt wird. Daraufhin überträgt das Notebook die Webcam-Aufnahmen an das Infotainment-System. Beispielweise zu vergleichen mit der Rückfahrkamera eines Autos, die Video-Daten beim Parken in das Infotainmentsystem überträgt. Der Vorteil der Raspberry Pis ist dagegen, dass der Traffic beliebig einstellbar ist, um unterschiedliche Testszenarien daraus zu entwickeln. Es konnte in dieser Bachelorarbeit festgestellt werden, dass bei diesem Prinzip ein Nachteil durch die Starvation entsteht. Das bedeutet, dass durch die unterschiedlichen Prioritätsklassen niedrig priorisierte Pakete verzögert oder bei einem persistent auftretenden Strom (Burst) von Paketen einer höheren Prioritätsklasse komplett blockiert werden und so niemals ihr Ziel erreichen. Deshalb wäre der nächste Schritt, das gesamte System auf Audio/Video-Bridge (AVB) umzusetzen. Voraussetzung hierfür ist ein funktionsfähiger AVB-Stack. Bei AVB wird das Problem mit der Starvation, die in dieser Arbeit erwähnt wurde siehe Kapitel 2.2, durch den Credit-Based-Shaper (CBS) gelöst.

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

Literaturverzeichnis

[1] *VDI Wissensforum / AUDI AG.* https://www.pressebox.de/pressemitteilung/vdi-wissensforum-gmbh/Komplexe-Bordnetze-entwickeln/boxid/498067 - Zugriffsdatum 12.12.2018.

[2] Vitalij Stepanov. *Mikrocontroller und CAN-basierte verteilte Regelung einer Steer-by-Wire Lenkung mit harten Echtzeitanforderungen*, 2011. http://edoc.sub.uni-hamburg.de/haw/volltexte/2011/1406/pdf/Vitalij_Stepanov_Steer_By_Wire.pdf.

[3] TEKTRONIX. INC. *TEKTRONIX, INC. Homepage.* https://www.tek.com/oscilloscope/mso4054b - Zugriffsdatum 15.03.2019.

[4] Reinhold Dörfler. *CAN Bus... ganz einfach!: Einführung in die CAN-Bus Technik.* Verlag Reinhold Dörfler, 2011.

[5] Till Steinbach. *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil.* Springer-Verlag, 2018.

[6] *IEEE Std 802.3-2018, IEEE Standard for Ethernet*, Aug 2018. Electronic ISBN: 978-1-5044-5090-4.

[7] *TTEthernet Specification. TTTech Computertechnik AG. November 2008.* https://www.tttech.com - Zugriffsdatum 10.01.2019.

[8] IEEE 802.1 TSN Task Group. *802.1Qbu - Frame Preemption.* http://www.ieee802.org/1/pages/802.1bu.html - Zugriffsdatum 6.01.2019.

[9] IEEE 802.1 TSN Task Group. *802.1Qbv - Enhancements for Scheduled Traffic.* http://www.ieee802.org/1/pages/802.1bv.html - Zugriffsdatum 6.01.2019.

[10] C. Rossi, A. Tilli, and A. Tonielli. Robust control of a throttle body for drive by wire operation of automotive engines. *IEEE Transactions on Control Systems Technology*, 8(6):993–1002, Nov 2000.

[11] CoRE RG. *CoRE RG: Communication over Real-time Ethernet.* http://core.informatik.haw-hamburg.de - Zugriffsdatum 9.12.2018.

[12] Wolfgang Riggert. *Rechnernetze: Grundlagen-Ethernet-Internet.* Carl Hanser Verlag GmbH Co KG, 2014.

[13] V. Rajaravivarma. *Virtual local area network technology and applications, March 1997 - Date Added to IEEE Xplore: 06 August 2002 - Print ISBN: 0-8186-7873-9.*

[14] *Institute of Electrical and Electronics Engineers, Inc. IEEE 802.1: 802.1Q - Virtual LANs.*, 2014. http://www.ieee802.org/1/pages/802.1Q-2014.html - Zugriffsdatum 21.01.2019.

[15] *IEEE Std 802.1Qat-2010, IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)*, 2010. Electronic ISBN: 978-0-7381-6501-1.

[16] AS6802, SAE. *SAE-AS-2D Time Triggered Systems and Architecture Committee, "Time-Triggered Ethernet (AS 6802).* URLhttp://www.sae.org - Zugriffsdatum 2.02.2019.

[17] Kai Mueller. *Time-Triggered Ethernet fuer eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfaehigen Netzwerkstack-Architektur*, 2011. http://edoc.sub.uni-hamburg.de/haw/volltexte/2011/1414/pdf/bachelorthesis.pdf.

[18] Jan Kamieth. *Entwurf einer Mikrocontroller basierten Bridge zur Kopplung von CAN Bussen ueber Time Triggered Realtime Ethernet*, 2011. http://edoc.sub.uni-hamburg.de/haw/volltexte/2011/1395/pdf/Jan_Kamieth_CAN_ETH_Bridge.pdf.

[19] Wireshark Foundation. *WIRESHARK FOUNDATION: Wireshark.* https://www.wireshark.org/ - Zugriffsdatum 18.04.2019.

[20] Vector Informatik GmbH. *Vector Informatik GmbH, SteuergerÃ¤te-Analyse mitCANalyzer.* https://www.vector.com/de/de/produkte/produkte-a-z/software/canalyzer/ - Zugriffsdatum 17.04.2019.

[21] Raspberry Pi Foundation. *Raspberry Pi 3 B+.* https://www.raspberrypi.org/ - Zugriffsdatum 30.03.2019.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 6. Juni 2019 Anushavan Melkonyan