Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

**Lukas Wendt**

## Effiziente regelbasierte CAN-Ethernet Gateways im Automotivebereich

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Lukas Wendt

# Effiziente regelbasierte CAN-Ethernet Gateways im Automotivebereich

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Andreas Meisel

Eingereicht am: 18. April 2017

**Lukas Wendt**

**Thema der Arbeit**

Effiziente regelbasierte CAN-Ethernet Gateways im Automotivebereich

**Stichworte**

CAN, Feldbus, Gateway, Realtime Ethernet, Regelwerk, Automotive, Migration

**Kurzzusammenfassung**

Im Automotivebereich werden häufig Feldbusse wie der CAN-Bus verwendet, die über ein zentrales Gateway miteinander verbunden sind. Alternativ dazu könnten mehrere CAN-Ethernet Gateways diese CAN-Busse über einen Ethernet-Backbone verbinden. So ist es möglich, vorhandene CAN-Steuergeräte an einem Ethernet-Backbone weiter zu verwenden. Ein solches CAN-Ethernet Gateway wird entwickelt, iterativ optimiert und im Hinblick auf seine Effizienz mit alternativen Gateways verglichen. Um die Ausführungsdauer klein zu halten, wird zusätzlich ein Code Generator entwickelt, der die Konfiguration (Regelwerk) einliest und daraus einen minimalistischen Gateway-Quellcode erstellt. Eine Validierung im Prototypenfahrzeug liefert Messergebnisse aus dem realen Einsatz und belegt die Funktionstüchtigkeit. Es wird gezeigt, dass die Übertragung eines CAN-Frames über Ethernet von einem zu einem anderen CAN-Bus mit der gleichen Effizienz möglich ist wie mit einem zentralen CAN-Gateway.

**Title of the paper**

Efficient rule-based CAN-Ethernet gateways in the automotive sector

**Keywords**

CAN, fieldbus, gateway, real-time Ethernet, rules, automotive, migration

**Abstract**

In the automotive sector, fieldbuses like the CAN bus are often used and interconnected via a central gateway. Alternatively, several CAN-Ethernet gateways could connect CAN buses via an Ethernet backbone. Such a CAN-Ethernet gateway is developed, iteratively optimized and compared with alternative gateways for its efficiency. To keep the execution time small, a code generator is also developed that reads the configuration (rules) and creates a minimal gateway source code. The validation in the prototype vehicle provides test results from the real enviroment and prove the operational capability. It is shown that the transmission of a CAN frame from one CAN bus to the other via Ethernet is possible with the same efficiency as via central CAN gateway.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Formelverzeichnis

# 1 Einleitung

In modernen Kraftfahrzeugen befinden sich viele Steuergeräte, die Electronic Control Units (ECUs), die miteinander vernetzt sind. Für die herkömmliche Kommunikation wird meistens der Controller Area Network (CAN)-Bus eingesetzt. Dieser ist kostengünstig, robust und weit verbreitet. Für viele moderne Anforderungen ist der CAN-Bus jedoch technisch nicht mehr ausreichend. Für Assistenzsysteme, wie die Rückfahrkamera, sind höhere Bandbreiten erforderlich als es der CAN-Bus ermöglicht. Um in Zukunft hohe Bandbreiten und trotzdem harte Echtzeitanforderungen zu erfüllen, arbeitet z.B. die Communication over Realtime Ethernet (CoRE) Research Group an der Hochschule für Angewandte Wissenschaften (HAW), an der Entwicklung von ECUs auf Basis von Realtime Ethernet (RTE) im Automotivebereich. Langfristig soll so eine Alternative zu vorhandenen Feldbussystemen wie CAN, Flexray und weiteren Feldbussen im Automotivebereich entstehen (CoRE-Arbeitsgruppe, 2014).

### Problemstellung

Für eine Übergangsphase wird ein Parallelbetrieb mehrerer Kommunikationsnetze im Fahrzeug vonnöten sein. Verschiedenartige ECUs sind vorhanden und können nicht ohne Weiteres auf eine neue Technologie umgestellt werden, so beispielsweise die Motorsteuergeräte. Außerdem würden Kosten ohne einen direkten Mehrwert entstehen, da die Funktionalität eines Steuergerätes durch die Migration auf eine andere Technologie nicht zunimmt. Durch verschiedene Netze im Fahrzeug müssen Bereiche im Fahrzeug doppelt verkabelt werden, in denen ein Zugriff auf beide Netze erforderlich ist. Um eine doppelte Verkabelung zu vermeiden und eine Kommunikation zwischen den Netzen zu ermöglichen, z.B. die Anzeige der Öltemperatur im Infotainment-System des Fahrzeugs, wird ein Gateway zwischen diesen Netzen benötigt. Solche Gateways gibt es bereits in modernen Fahrzeugen. Das Prototypenfahrzeug der CoRE Gruppe besitzt ein zentrales CAN-Gateway, das Nachrichten zwischen den CAN-Bussen im Fahrzeug transportiert. Im Rahmen dieser Arbeit wird ein CAN-Ethernet Gateway entworfen, das die volle Bandbreite des CAN-Busses unterstützt und eine möglichst geringe Latenz haben soll. Da in Zukunft eine schrittweise Migration zu einem Ethernet-Backbone möglich sein soll, muss diese Lösung eine ähnliche Performance wie ein zentrales Gateway erreichen. Im weiteren Verlauf dieser Arbeit wird dieses neue Gateway als Gateway-2017 bezeichnet.

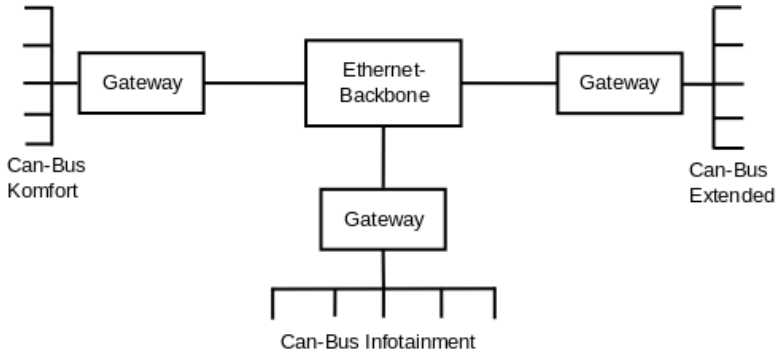



Abbildung 1.1: Exemplarische Darstellung eines verteilten CAN-Busses über einen Ethernet-Backbone im Prototypenfahrzeug

In der Abbildung 1.1 ist die Vernetzung von CAN-Netzen exemplarisch dargestellt. Diese sind häufig in thematische Domänen aufgeteilt und sollen über einen Ethernet-Backbone verbunden werden.

In jedem Gateway sollen sich Regeln festlegen lassen, welche CAN-Frames weitergeleitet werden und wie lange diese vor der Weiterleitung maximal im Gateway zwischengespeichert werden.

Mehrere CAN-Frames sollen in einem Ethernet-Frame gebündelt werden.

An der HAW Hamburg wurde bereits ein solches Gateway (Gateway-2015) von Jan Depke in seiner Masterarbeit entwickelt (Depke, 2015). Das Gateway wurde flexibel gehalten, da zur Laufzeit die Regeln vom „Parser“ interpretiert werden. Der „Parser“ ist die Komponente im Gateway-2015, die passenden Regeln (Verarbeitungsvorschriften) einer Nachricht zuordnet. So konnte mit diesem CAN-Ethernet Gateway keine komplette Auslastung des CAN-Busses erreicht werden. Außerdem war die Latenz deutlich größer als die des Volkswagen (VW)-Gateways. In dieser Arbeit wird daher ein Code Generator entwickelt, der einen Teil des Source-Code für das Gateway-2017 generiert. So können einige Berechnung bereits im Voraus durchgeführt werden und der zur Laufzeit auszuführende Code wird statischer, was sich positiv auf die Laufzeit auswirken wird.

Durch den Code Generator geht der dynamische Teil des Gateways-2015 verloren, was bedeutet, dass jede Änderung der Regeln ein erneutes Kompilieren erfordert. Dies sollte im Automotivebereich kein Problem darstellen, da bereits beim Entwurf festgelegt wird, welche CAN-Identifier (ID) auf einem CAN-Bus vorkommen.

### Aufbau der Arbeit

Im ersten Teil der Arbeit werden die technischen Grundlagen, die für diese Arbeit erforderlich sind, erklärt. Dazu gehören die Funktionsweise des RTE und der verschiedenen Feldbusse. Zunächst wird eine Ist-Analyse erstellt, bevor im dritten Kapitel ein Konzept erarbeitet wird, welches die Anforderungen an das Gateway-2017 und die Planung der Umsetzung beinhaltet. Außerdem werden die verschiedenen Datenstrukturen und die Nachrichtenbündelung erläutert.

Danach folgt eine Optimierung des Konzepts, um die Regelsuche zu verbessern.

Im nächsten Kapitel wird beschrieben, wie das Konzept umgesetzt wurde und wie das Gateway-2017 und der Code Generator verwendet werden können.

In der Evaluierung und Validierung werden zunächst die Tests, die mit dem Gateway-2017 durchgeführt wurden, beschrieben. Anschließend wird überprüft, ob die Anforderungen des Konzepts erfüllt wurden. Zuletzt werden die Laufzeiten der einzelnen Funktionen vermessen und dargestellt. Dementsprechend kann das Schedule eingestellt und die Latenzen des Gateways-2017 vermessen werden. Optimierungsmöglichkeiten werden erläutert und miteinander verglichen. Außerdem wird das Gateway-2017 im Prototypenfahrzeug verbaut, getestet und mit dem Gateway-2015 und dem VW-Gateway im Hinblick auf die Latenz verglichen.

Im Schlussteil soll deutlich werden, ob ein Ethernet-Backbone für die Übertragung von CAN-Frames eine ähnliche Performance erreicht wie ein zentrales Gateway und welche Überlegungen sich für die Zukunft anschließen.

# 2 Grundlagen

In diesem Kapitel werden die Grundlagen zu CAN und Ethernet erläutert.

## 2.1 Feldbussysteme

In modernen Fahrzeugen gibt es verschiedene etablierte Feldbussysteme:

| Feldbussystem | Topologie | Bandbreite | Anzahl Geräte |
|---|---|---|---|
| CAN | Bus- und Stern-Topologie | $\leq$ 1 MBit/s | maximal 128 Geräte |
| LIN | Bus-Topologie | $\leq$ 20 KBit/s | maximal 16 Slaves + Master |
| FlexRay | Bus- und Stern-Topologie | $\leq$ 10 MBit/s | Anzahl der Slots |
| Most | Ring- und Stern-Topologie | $\leq$ 150 MBit/s | maximal 64 Geräte |

Tabelle 2.1: Übersicht der Feldbussysteme nach LIN Consortium, 2010, MOST Cooperation, 2011. International Organization for Standardization, 2015, FlexRay Consortium, 2010

Da der CAN-Bus in der Automotivebranche am stärksten verbreitet ist (CAN in Automation, 2014), wird sich in dieser Arbeit auf die Entwicklung eines CAN-Ethernet Gateways fokussiert.

Die Integration von Local Interconnect Network (LIN) würde sich relativ simpel lösen lassen, da der Master im LIN Netzwerk meist auch eine Verbindung zum CAN-Netz hat und so bereits als Gateway zwischen LIN und CAN dient. So kann das CAN zu Ethernet Gateway diese Nachrichten aus dem CAN-Netz in das Ethernet-Netz übermitteln. Die Software wird sich einfach zu einem Flexray-Ethernet Gateway portieren lassen, weil sich in beiden Frames (CAN und Flexray) ein Data-Feld und ein Feld, in dem die Länge des Data-Feldes steht, befindet.

Die Portierung zu einem Media Oriented Systems Transport (MOST)-Ethernet Gateway wird schwieriger durchzuführen sein, da der MOST-Frame ein synchrones und ein asynchrones Data-Feld enthält und diese Felder im Gateway unterschiedlich behandelt werden müssten.

## 2.2 CAN

Der CAN-Bus wurde 1983 von der Bosch Gruppe entwickelt. Die maximale Bandbreite des CAN-Bus von 1 MBit/s ist abhängig von der Länge der Leitung. Als physikalisches Medium wird ein Unshielded-Twisted-Pair-Kabel mit einem Abschlusswiderstand von 120 Ω empfohlen (International Organization for Standardization, 2015). Auf dem CAN-Bus ist eine logische 1 rezessiv und eine logische 0 dominant. Sobald ein Controller eine 0 (dominant) auf dem Bus anlegt, der davor auf dem Pegel 1 (rezessiv) war, nimmt der Pegel nun den Wert 0 (dominant) an. Im weiteren Text werden im Zusammenhang mit dem CAN-Bus die Begriffe rezessiv und dominant für die Pegel auf dem CAN-Bus verwendet.

### 2.2.1 Aufbau CAN-Frame

Es gibt einen Standard CAN-Frame (CAN 2.0A) und einen Extended CAN-Frame (CAN 2.0B). Der Extended CAN-Frame hat einen erweiterten Identifier. CAN-Frames mit einer kleineren CAN-ID haben eine höhere Priorität, da ein dominantes Bit ein rezessives Bit überschreibt und der Sender des rezessiven Bits daraufhin die Übertragung abbricht. Falls beide Standards auf einem CAN-Bus vorkommen, wird der Standard CAN-Frame höher priorisiert behandelt, da das Identifier Extension (IDE) Bit Feld dort ein dominantes Bit ist.

| SoF | Identifier | RTR | CTRL | Daten | CRC Checksum | CRC Delimiter | Ack-Slot | Ack-Delimiter | EoF | Ifs |
|---|---|---|---|---|---|---|---|---|---|---|

Tabelle 2.2: Aufbau eines CAN-Frames (CAN 2.0A) nach ISO 11898-2 International Organization for Standardization, 2003

| SoF | Identifier | SRRB | CTRL | Erweiterter Identifier | Daten | CRC Checksum | CRC Delimiter | Ack-Slot | Ack-Delimiter | EoF | Ifs |
|---|---|---|---|---|---|---|---|---|---|---|---|

Tabelle 2.3: Aufbau eines CAN-Frames (CAN 2.0B) nach ISO 11898-2 International Organization for Standardization, 2003

#### Data Frame

Das Feld **Start of Frame (SoF)** steht am Beginn jedes CAN-Frames und hat ein dominant gesetztes Bit.

Das **Identifier** Feld ist die Kennung des CAN-Frames und ist 11 Bit groß.

Das **Substitute Remote Request Bit (SRRB)** wird im extended-Frame ebenso wie das **Remote Transmission Request (RTR)** Bit im Standard-Frame genutzt, damit ein Teilnehmer einen anderen auffordern kann, seine Daten zu senden.

Der **Erweiterte CAN Identifier** erweitert den Identifier des CAN-Frames um 18 Bit, sodass insgesamt 29 Bit für die Kennung eines CAN-Frames im Extendend Format zur Verfügung stehen.

Im **Control** Feld steht in Bit 0-3 die Größe des Datenfeldes (DLC), das zwischen 0 und 8 Byte groß sein kann, ein weiteres Bit im Control Feld steht für **IDE**. Hiermit kann der CAN-Controller erkennen, ob es sich um einen Standard oder Extended CAN-Frame handelt.

Die **cyclic redundancy check (CRC) Summe** ist die Checksumme, die sich aus dem CAN-Frame berechnet. Der **CRC Delimiter** ist ein rezessives Bit, das das Ende der Checksumme anzeigt. Der **End of Frame (EoF)** ist 7 Bit lang und besteht nur aus rezessiven Bits. Diese Kombination kann nur einmal im Frame auftreten (Erklärung in Abschnitt 2.2.1 bit stuffing) und markiert so eindeutig das Ende eines Frames. Das **Interframe spacing (Ifs)** besteht ebenfalls aus rezessiven Bits. Sobald hier ein dominanter Pegel folgt, beginnt ein neuer Frame.

#### Remote Frame

Mit dem Remote Frame kann das Senden eines Data Frames angefordert werden. Hierzu wird ein CAN-Frame mit einem rezessiven RTR (Standard-Frame) Bit oder einem rezessiven SRRB (Extended-Frame) Bit gesendet. Ein Remote Frame enthält keine Daten, aber das DLC Feld kann trotzdem größer 0 sein.

#### Error Frame

Wenn ein Empfänger einen Fehler bei der Übertragung eines Frames feststellt, sendet der Empfänger einen Error Frame. Damit der CAN-Bus nicht von dem Error Frame belegt wird, gibt es einen Empfänger- und einen Sender-Zähler, der bei Fehlern hochgezählt wird. Wenn dieser ein eingestelltes Limit erreicht, verlässt der Teilnehmer den CAN-Bus. Wenn der Fehler nur bei diesem Teilnehmer auftrat, können alle anderen problemlos weiter kommunizieren.

#### Acknowledge-Slot

Um festzustellen, ob ein Frame ordnungsgemäß übertragen wurde, setzt der Sender das Acknowledge (ACK)-Feld auf rezessiv. Ein Empfänger kann das Bit auf dominant setzen und so den Empfang bestätigen. Der ACK-Slot sagt jedoch nichts darüber aus, ob alle Teilnehmer die Daten auf dem Bus empfangen haben. Es braucht nur ein Teilnehmer das Bit auf dominant zu

setzen, obwohl möglicherweise niemand anderes den Frame empfangen hat. Der Sender kann auch nicht erkennen, welcher Empfänger den Frame bestätigt hat.

#### Bit stuffing

Da keine seperate Clock-Leitung auf dem CAN-Bus existiert, wird die Clock zwischen den CAN-Controllern über die Flanken auf der Datenleitung synchronisiert.

Damit es keine langen Ketten an rezessiven oder dominaten Bits gibt, bei denen keine Flanke auftreten würde, ergänzt der CAN-Controller den CAN-Frame am Ende von 5 identischen Bits um ein invertiertes Bit. Da der Empfänger dieses Verfahren kennt, kann er das zusätzliche Bit wieder entfernen und gleichzeitig seine Clock mit der des Senders synchronisieren. Eine Ausnahme bildet hierbei das EoF-Feld, da es aus 7 rezessiven Bits besteht. Es ist somit im Bit-Stream eindeutig, weil das bit stuffing überall sonst ein dominantes Bit einfügen würde. Um die Anzahl an Bits zu ermitteln, um die sich der CAN-Frame im Worst Case Fall vergrößert, lässt sich folgende Formel verwenden (Medhat, 2015):

$$bSB = \frac{AB - 1}{4} \tag{2.1}$$

Maximale Anzahl an bit stuffing Bits

| Variable | Verwendung |
|---|---|
| bSB | bit Stuffing Bits |
| AB | Anzahl der Bits |

Vom bit stuffing betroffen sind beim Standard-Frame 34 Bits und bei einem Extended-Frame 52 Bits. Dies sind weniger Bits als in einem CAN-Frame, da einige Felder nicht vom bit stuffing betroffen sind, wie z.B. das EoF-Feld. Hinzu kommt die Anzahl der Bits, die für die Nutzlast verwendet werden. Der Divisor 4 erklärt sich dadurch, dass beim bit stuffing das eingefügte Bit mit zu den 5 Bits zählt und daher im Worst Case nach jedem 4. Frame Bit ein Stuff Bit eingefügt wird.

### 2.2.2 Arbitrierung & Größe

Unter Arbitrierung wird im Bereich der Feldbusse der Vorgang bezeichnet, der den Zugriff mehrerer CAN-Teilnehmer auf den CAN-Bus regelt. Die Arbitrierungsdauer umfasst daher die Zeit, die ein CAN-Teilnehmer warten muss, bis kein höher priorisierter Frame mehr übertragen wird und die eigene Übertragung abgeschlossen ist.

Die Größe eines Frames hängt von verschiedenen Faktoren ab, einmal sind Std.- und Ext.-Frames unterschiedlich groß, außerdem variiert die Größe der übertragenden Nutzdaten. Die 4 Ergebnisse in der linken Tabelle 2.4 entsprechen daher einem Std.-Frame mit 8 Byte Daten und

ohne Daten und einem Ext.-Frame mit 8 Byte Daten und ohne Daten. In der rechten Tabelle 2.5 werden zu den 4 Ergebnissen die maximale Anzahl an bit stuffing Bits hinzugefügt.

| Feld | Std.-Frame | Ext.-Frame |
|---|---|---|
| SoF | 1 | 1 |
| Identifer | 11 | 11 |
| RTR/SRRB | 1 | 1 |
| CTRL | 6 | 6 |
| CRC | 15 | 15 |
| CRC Delimiter | 1 | 1 |
| ACK-Slot | 1 | 1 |
| ACK Delimiter | 1 | 1 |
| EOF Bits | 7 | 7 |
| IFS | 3 | 3 |
| Erweiterter Identifier | 0 | 18 |
| = Summe min | **47** | **65** |
| +8 Byte | 64 | 64 |
| = Summe max | **111** | **129** |

Tabelle 2.4: Größe eines Extended (Ext.)- und eines Standard (Std.)-Frames - ohne bit stuffing

| Payload | Std.-Frame | Ext.-Frame |
|---|---|---|
| Sum. min | 47 | 65 |
| Sum. max | 111 | 129 |
| +0 Byte | 8 | 12 |
| +8 Byte | 24 | 28 |
| Sum. 0 B. | **55** | **77** |
| Sum. 8 B. | **135** | **153** |

Tabelle 2.5: Größe eines Extended (Ext.)- und eines Standard (Std.)-Frames - mit bit stuffing

Wenn in dieser Arbeit der Begriff „**minimaler Frame**“ verwendet wird, ist ein Standard-Frame ohne Nutzlast und ohne bit stuffing gemeint. Ein „**maximaler Frame**“ ist ein Extended-Frame mit voller Nutzlast und Worst Case bit stuffing.

## 2.3 Ethernet

Ethernet ist der Standard für die Vernetzung von Computernetzwerken und wurde 1985 von der Ethernet Group (IEEE Ethernet Group, 1985) veröffentlicht. Durch die große Verbreitung ist eine Massenproduktion verfügbar und dementsprechend ist der einzelne Ethernet-Controller sehr kostengünstig. Nach Stasch, 1997, kostete ein Ethernet-Controller 1988 4000$ und 1993 1000$. Am 21.12.2016 kostete ein 1GBit/s Ethernet-Controller nur noch 5,95 €. Außerdem ist Ethernet unabhängig vom physikalischen Medium. Es ist z.B. standardisiert für twisted pair-, coaxial- und Glasfaser-Verbindungen (IEEE 802.3). Twisted Pair wird auch von CAN verwendet. Dieselben Kabel können für Ethernet genutzt werden. Verwendet wird heute eine

Stern-Topologie. Es gibt aber auch Standards, wie 10Base2 und 10Base5, die eine Bus-Topologie unterstützen. Hierfür ist keine Hardware mehr auf dem Markt verfügbar. Hervorzuheben ist außerdem die Bandbreite, die mit Ethernet erreicht werden kann. Diese liegt bei bis zu 100GBit/s, spezifiziert im Standard 802.3ba (D'Ambrosia, 2010), was Ethernet auch für Multimedia und weitere Entwicklungen in der Zukunft prädestiniert.

### 2.3.1 Aufbau Ethernet-Frame

Im folgenden Kapitel wird der Aufbau eines Ethernet-Frames nach Standard IEEE Ethernet Group, 1985, beschrieben.

| Preamble | SFD | Destination Address | Source Address | Length/Type | Data | Pad | FCS | IPG |
|---|---|---|---|---|---|---|---|---|

Tabelle 2.6: Aufbau eines Ethernet-Frames nach IEEE 802.3 IEEE Ethernet Group, 1985

Die **Preamble** ist ein 7 Byte großer Block, dessen Inhalt sich dauernd ändert, aber periodisch verläuft. So kann die Clock der Empfänger im Ethernet-Netzwerk synchronisiert werden.

Der **Start Frame Delimiter (SFD)** ist eine statische Sequenz, die immer nach der Preamble folgen muss, und 1 Byte groß ist.

Danach kommen die **Destination/Ziel**- und die **Source/Quell**-Adressen. Diese sind jeweils 6 Byte groß und werden im weiteren als Media Access Control (MAC)-Adresse bezeichnet. Im **Length/Type**-Feld steht die Größe des Data-Feldes, wenn diese kleiner gleich 1500 beträgt. Falls dieser Wert größer als 1500 ist, handelt es sich nicht um einen Standard Ethernet-Frame (IEEE 802.3). Das Feld selbst ist 2 Byte groß. Das **Data**-Feld kann zwischen 46 und 1500 Bytes enthalten.

Das **Pad** füllt einen Ethernet-Frame bis zu seiner minimalen Größe auf, wenn das Data-Feld weniger als 46 Bytes beinhaltet.

Das **Frame Check Sequence (FCS)**-Feld ist ein CRC Wert, der 32 Bit groß ist, sich aus den Address-, Data-, Length- und Pad-Feldern berechnet und so die Erkennung von Fehlern bei der Übertragung ermöglicht.

Zwischen zwei Frames muss eine Idle-Zeit eingehalten werden das wird mit Inter-Packet Gap (IPG) erfüllt, dieses Feld hat eine Größe von 12 Byte. Alle diese Felder addiert ergeben die minimale Frame-Größe von 84 Bytes und die maximale Frame-Größe von 1538 Bytes.

### 2.3.2 CSMA/CD

Damit mehrere Teilnehmer ein physikalisches Übertragungsmedium nutzen können, befindet sich im Standard (IEEE 802.3 IEEE Ethernet Group, 1985) das Verfahren Carrier Sense Multiple Access/Collision Detection (CSMA/CD). Bei diesem Verfahren prüft ein Teilnehmer, ob das Medium frei ist. Sobald es frei ist, wartet der Teilnehmer eine zufällige Zeit ab. Wenn das Medium dann immer noch frei ist, versendet der Teilnehmer seinen Frame. Kommt es dennoch zu einer Kollision zweier gleichzeitig versendeter Frames, bricht die Übertragung aller Teilnehemer ab und das Verfahren beginnt wieder von vorn. Das Problem in echtzeitkritischen Anwendungen ist, dass nicht garantiert werden kann, dass ein Teilnehmer bis zum Zeitpunkt X seine Nachricht verschickt hat und sie tatsächlich beim Kommunikationspartner angekommen ist. Außerdem wird CSMA/CD in einem geswitchten Ethernet-Netzwerk nicht mehr benötigt, wenn kein gemeinsames Medium mehr genutzt wird.

### 2.3.3 TDMA

Das Time Division Multiple Access (TDMA) wird häufig in drahtlosen Netzwerken verwendet um den Zugriff mehrerer Teilnehmer auf ein physikalisches Medium zu kontrollieren. Dazu wird ein Zeitintervall (Zyklus), beispielsweise 1 ms, festgelegt und in mehrere Slots aufgeteilt. Es wird nicht eine zufällige Zeit gewartet, sondern es wird ein Slot ausgewählt, der im letzten Zyklus unbenutzt war. Falls zwei Teilnehmer den gleichen Slot verwenden, kann es hier immer noch zu einer Kollision kommen. In diesem Fall wählen beide einen neuen, zufälligen Slot, der im letzen Zyklus nicht verwendet wurde. Auch hier kann so noch nicht garantiert werden, dass ein Teilnehmer seine Nachricht bis zum Zeitpunkt X versendet hat. Sollten alle Slots belegt sein oder es wählen zwei Teilnehmer durch Zufall den gleichen Slot, kommt es zu einer zeitlich nicht vorhersagbaren Verzögerung. Nimmt man diesem Verfahren die Dynamik (zufällige Slot-Wahl) und weist allen Teilnehmern spezifische Slots zu, kann ein Senden innerhalb eines Zeitraum X garantiert werden.

### 2.3.4 Realtime Ethernet/Scheduler

Dieses Verfahren nutzt die Firma TTTech Computertechnik AG für ihr Realtime Ethernet Protokoll TTEthernet. Mithilfe dieses Protokolls hat Kai Müller in seiner Bachelorarbeit (Müller, 2011) einen kooperativen Scheduler entwickelt. Dieser Scheduler wird im Rahmen der folgenden Bachelorarbeit für die statische Koordinierung der Tasks im Gateway-2017 genutzt. Mit einem Scheduler können verschiedene Tasks zu bestimmten Zeitpunkten geplant werden. Um eine Synchronisierung aller Sende Events im Netzwerk zu erreichen, muss ein Gerät als Sync Master sogenannte „Sync Frames“ verschicken. Alle anderen Geräte werden als Sync Client

konfiguriert und passen ihr Schedule entsprechend an, sodass der Sendevorgang im passenden Zeitslot ausgeführt wird. Dieser Synchronisationsvorgang muss einmal pro Scheduler-Periode aufgerufen werden.

## 2.4 Subnetting

Der Begriff Subnetting kommt vom Internet Protocol (IP) und bezeichnet ein Verfahren, mit dem ein Netz in Teilnetze aufgeteilt werden kann. Bei der Aggregation werden diese Teilnetze wieder zu großen einzelnen Netzen zusammengefasst um die Anzahl der Routing Einträge gering zu halten. IP ist ein Protokoll, das sich auf der Vermittlungsschicht(3) des Open Systems Interconnection (OSI)-Modells befindet. Weitere Informationen zum IP finden sich im Buch von Tanenbaum und Wetherall, 2012. Für diese Bachelorarbeit wird das Subnetting- und das Aggregations-Verfahren zur Aufteilung und Zusammenfassung von CAN-Netzen adaptiert.

# 3 IST-Analyse

Der CoRE-Gruppe steht ein Prototypenfahrzeug (VW Golf 7) zur Verfügung. Dieses Fahrzeug enthält im Originalzustand mehrere CAN-Busse, die über ein CAN-Gateway von VW miteinander verbunden sind. Im weiteren Verlauf dieser Arbeit wird dieses Gateway als VW-Gateway bezeichnet. 3 CAN-Busse, der Infotainment (I)-, der Komfort (K)- und der Extended (E)-Bus, wurden für die Tests ausgewählt, weil sie für die Fahrzeugsicherheit weniger relevant sind. Es befinden sich außerdem 3 nachgerüstete Gateways (2015 I, 2015 E, 2015 K) im Prototypenfahrzeug. Diese werden über eine CAN-Diode an das Fahrzeug angeschlossen. Diese Diode sorgt dafür, dass die Gateways auf die 3 CAN-Busse nicht schreiben können, da sonst die Zulassung des Prototypenfahrzeugs erlischt. Zur Kontrolle schicken diese Gateways ihre Ethernet-Frames an ein Logging System. Zu Testzwecken können die Gateways ohne Diode mit Schreibzugriff angeschlossen werden.

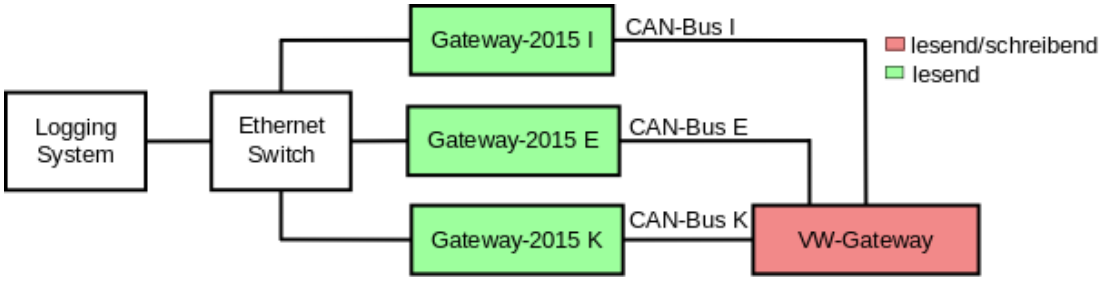



Abbildung 3.1: Schematische Darstellung des Ist-Aufbaus im Prototypenfahrzeug

## 3.1 Hardware

Bei den im Prototypenfahrzeug nachgerüsteten Gateways handelt es sich um Microcontroller (µC) der Firma Hilscher des Typs NXHX 51-ETM. Das Board besitzt einen 32 Bit Advanced Reduced Instruction Set Computer (RISC) Machines (ARM)-Prozessor, der mit 200 MHz getaktet ist. Er zeichnet sich speziell durch seine zwei Ethernet-Controller aus. Außerdem kann das Board mit 2 Feldbus-Anschlüssen (z.B. CAN, Radio Sector (RS)-232 und Profibus) bestückt werden. (Hilscher GmbH, 2013)

## 3.2 Verwandte Arbeiten zu CAN-Ethernet Gateways

Es werden drei Ausarbeitungen zum Thema CAN-Ethernet Gateways diskutiert. Hierbei handelt es sich um zwei an der HAW durchgeführte Arbeiten sowie um eine externe Veröffentlichung zum Thema CAN-Ethernet Gateways.

### Inkrementelle Konsolidierung automobiler Bussysteme auf Basis eines Realtime Ethernet Backbones

Die an der HAW erstellte Masterarbeit von Jan Depke beschreibt die Entwicklung eines CAN-Ethernet Gateways, dessen Regeln zur Laufzeit ausgewertet werden (Interpretationsansatz). Es wird untersucht, wie sich ein Gateway, das Feldbussysteme über einen Ethernet-Backbone verbindet, auf die Feldbussysteme auswirkt im Hinblick auf die Features des CAN-Busses (Data-, Remote-, Error-Frame und Acknowledge Slot).

Es wird erläutert, was für eine Migration zu einem MOST oder FlexRay zu Ethernet Gateway erforderlich ist. Ein verteilter CAN-Bus ist bis auf wenige Einschränkungen gleichwertig zu verwenden wie ein einzelner CAN-Bus. Mit dem in dieser Arbeit entwickelten Gateway lässt sich jedoch keine volle CAN-Bus Auslastung erreichen.

### Erprobung von Echtzeit Ethernet basierten Automobil-Gateways in einem Prototypfahrzeug

In der an der HAW von Patrick Kuncke erstellten Bachelorarbeit (Kuncke, 2016) wurde ein Filter entwickelt um das CAN-Ethernet Gateway von Jan Depke im Prototypenfahrzeug zu testen. Die Latenzen des VW-Gateways und des CAN-Ethernet Gateways wurden vermessen und anschließend verglichen. Außerdem wurde auf einer Teststrecke validiert, ob das Prototypenfahrzeug mit dem CAN-Ethernet Gateway fahrtüchtig bleibt.

Das Prototypenfahrzeug erwies sich als fahrtüchtig, aber die Latenzen vom CAN-Ethernet Gateway waren deutlich höher als mit dem VW-Gateway. Die maximalen Werte sind sehr hoch und der Mittelwert ist 3 mal höher als beim VW-Gateway. Eine genauere Auswertung findet sich im Kapitel „Zeitliches Verhalten des Gateways“.

### CAN over Ethernet Gateway - A Convenient and flexible Solution to Access Low Level Control Devices

Diese Ausarbeitung (Thomas, Davids und Holme, 2015) testet ein CAN-Ethernet Gateway der Firma AnaGate auf seine Performance und Stabilität für den Einsatz in der Forschungseinrichtung CERN. Hierfür werden verschiedene Fehler provoziert und überprüft, ob das Gateway

selbstständig in einen stabilen Betrieb zurückkehrt. Die Latenz wird gemessen und der maximale Durchsatz wird ermittelt bis es zu Frame-Verlusten kommt. Dieses Gateway verlässt selbständig die Fehlerzustände und kehrt in einen sicheren Betriebszustand zurück. Die Latenz liegt im einstelligen ms Bereich.

Eine volle Auslastung wurde bei einer CAN-Schnittstelle erreicht. Bei zwei Schnittstellen kam es ab 60 % (9600 Frames/s) Auslastung zu Frame-Verlusten.

### 3.2.1 Abgrenzung/Schlussfolgerung

Gateways, die auf dem Markt verfügbar sind, wie das Gateway von AnaGate, verwenden häufig Transmission Control Protocol (TCP) oder User Datagram Protocol (UDP) für die Übertragung von CAN-Frames. Außerdem wird häufig ein Linux Unterbau verwendet. Beides führt zu einem höheren Protokolloverhead und zu höheren Laufzeiten im Gateway.

Das im Laufe dieser Arbeit entwickelte Gateway-2017 wird daher nur mit dem CAN-Ethernet Gateway (Gateway-2015) und dem VW-Gateway verglichen. Auch das Gateway-2015 nutzt Ethernet-Frames (ohne TCP oder UDP) und verwendet kein Betriebssystem, das zusätzlichen Overhead erzeugen würde.

# 4 Konzept

Es folgt die Ausarbeitung des Soll-Konzepts und die Auswahl der Methoden und Verfahren, mit denen das Gateway-2017 arbeiten und implementiert werden soll.

## 4.1 Soll-Konzept

Im Soll-Konzept werden die Muss-, Wunsch- und Abgrenzungs-Kriterien erläutert.

### 4.1.1 Muss-Kriterien

Dieses Kapitel beschreibt Anforderungen, die das Gateway-2017 erfüllen muss. Um den Overhead auf dem Ethernet-Backbone möglichst gering zu halten, sollen mehrere CAN-Frames zwischengespeichert und als eine Nachricht verschickt werden (Aggregation). Durch die Aggregation wird die Weiterleitung der CAN-Frames jedoch verzögert. Außerdem kommen unter Umständen viele Frames gleichzeitig an, die anschließend sequentiell weiter versendet werden müssen.

Das Gateway-2017 soll im Hinblick auf die Effizienz so optimiert werden, dass die maximale Bandbreite des CAN-Busses verarbeitet werden kann. Unter der maximalen Bandbreite oder der vollen Auslastung ist zu verstehen, dass die maximale Anzahl pro Sekunde an minimalen CAN-Frames verarbeitet werden kann. Außerdem muss gewährleistet sein, dass beim Empfang von Ethernet-Frames die maximale Anzahl an minimalen CAN-Frames vom Gateway erzeugt wird. CAN-Frames müssen bei gleichem Timeout in der richtigen Reihenfolge bleiben.

Das Daten-, DLC- und das ID-Feld dürfen vom Gateway-2017 nicht verändert werden.

Über ein Regelwerk sollen Nachrichten unterschiedlich sortiert, gefiltert und zwischengespeichert werden. Dieses Regelwerk schreibt vor, an welche MAC-Adresse eine bzw. mehrere CAN-IDs weitergeleitet werden sollen. Eine exemplarische Darstellung des zu erreichenden Aufbaus, findet sich in der Abbildung 1.1.

### 4.1.2 Wunsch-Kriterien

Es wäre wünschenswert, wenn Funktionen, wie der ACK-Slot, Remote Frame oder der Error Frame, vom Gateway unterstützt würden.

Eine Filterung nach Duplikaten (identischer ID, oder identischer Inhalt) im Buffer würde zusätzlich die Auslastung auf dem Ethernet-Backbone und dem CAN-Bus verringern.

### 4.1.3 Abgrenzung

Obwohl die Hardware zwei CAN- und zwei Ethernet-Schnittstellen hat, wird das Gateway-2017 nur eine CAN- und eine Ethernet-Schnittstelle unterstützen. Folgende Szenarien werden also nicht direkt unterstützt:

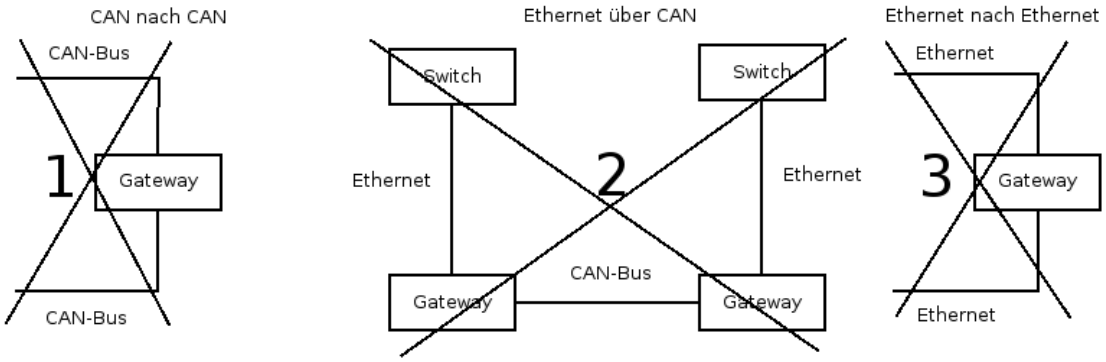



Abbildung 4.1: Darstellung der Netzwerkarchitekturen, die vom Gateway nicht unterstützt werden sollen

Abbildung 4.1 Nr. 1: Das Übertragen von einem CAN-Bus zu einem anderen CAN-Bus wird nicht unterstützt. Hierfür wären erweiterte Regeln nötig: Regeln, die einen empfangenen CAN-Frame an die andere CAN-Schnittstelle weiterleiten und Regeln, die beim Empfang von Ethernet-Frames entscheiden, auf welcher CAN-Schnittstelle die entpackten CAN-Frames verschickt werden müssten.

Nr. 2: Das Tunneln von Ethernet-Frames über den CAN-Bus ist z.B. von Nutzen, wenn über weite Distanzen kommuniziert werden soll. So kann ein Ethernet-Frame über den CAN-Bus übertragen werden, der eine Reichweite von bis zu 6,7 km unterstützt, obwohl 100 m bereits die maximale Kabellänge bei Twisted Pair Kabeln für die Ethernet-Kommunikation ist. Diese Funktion wird von dem zu entwickelnden Gateway-2017 nicht unterstützt. Da Ethernet-Frames größer als CAN-Frames sind, müssten die Ethernet-Frames in Teile zerlegt und später wieder zusammengesetzt werden. Das würde einen zusätzlichen Buffer beim Empfänger

benötigen, worauf beim Gateway-2017 verzichtet wird. Außerdem kommen solche Kabellängen im Automotivebereich nicht vor.

Nr. 3: Das Verbinden von zwei Ethernet-Netzen könnte genutzt werden um die Broadcast Domain auf Layer 2 zu verringern und um die Kommunikation zwischen verschiedenen Teilnehmern zu beschränken. Diese Funktion wird vom Gateway-2017 nicht unterstützt, da hierfür zusätzliche Regeln beim Empfang von Ethernet-Frames durchsucht werden müssten. Es wäre zu unterscheiden, ob es sich um einen Ethernet-Frame handelt, der weitergeleitet werden soll, oder um einen Ethernet-Frame, der CAN-Frames enthält.

### Tabellarische Zusammenfassung

Um später die Erfüllung der Anforderungen zu prüfen, wurden die Anforderungen in einer Tabelle zusammengefasst:

| **Typ** | **Beschreibung** |
|---|---|
| Muss | maximale Bandbreite CAN |
| | CAN-Frames bündeln |
| | Reihenfolge bleibt unverändert |
| | Daten, DLC- und ID-Feld unverändert |
| Wunsch | Ack-Slot CAN-Bus übergreifend |
| | Error Frame CAN-Bus übergreifend |
| | Remote Frame CAN-Bus übergreifend |
| | Duplikate filtern |
| Abgrenzung | kein CAN- nach CAN-Gateway |
| | kein Ethernet- nach Ethernet-Gateway |
| | kein Ethernet- über CAN-Tunneln |

Tabelle 4.1: Darstellung der Kriterien, die das Gateway-2017 erfüllen soll

## 4.2 Regeln

Das Gateway-2017 soll ein Regelwerk verwenden, das beim Empfang von CAN-Frames angewendet wird.

Da ein CAN-Frame einem Ethernet-Frame zugeordnet werden soll, muss eine Regel die Parameter MAC-Adresse und CAN-ID enthalten. Da eine Nachrichtenbündelung von CAN-Frames durch das Gateway-2017 unterstützt werden soll, wird eine Regel zusätzlich um den Parameter „Timeout“ ergänzt, der angibt, wie lange eine solche Nachricht maximal im Gateway-2017 gespeichert werden soll, bevor diese versendet wird.

Sollten auf einen CAN-Frame mehrere Regeln zutreffen, werden im Gegensatz zum Verhalten (Longest Prefix Match) beim IP-Routing alle Regeln ausgeführt. Das hat den Vorteil, dass ohne Veränderungen an den Steuergeräten ein CAN-Frame an mehrere CAN-Busse verschickt werden kann. Daraus folgt: Sollte es sich bei den verschiedenen Regeln um unterschiedliche Ethernet-Adressen handeln, würde der CAN-Frame dupliziert werden und an alle entsprechenden Ethernet-Empfänger verschickt werden. Sollte es sich um die gleiche Ethernet-Adresse, aber um unterschiedliche „Timeouts“ handeln, würde der kleinere „Timeout“ verwendet.

Um die Anzahl der Regeln im Regelwerk klein zu halten, wird festgelegt, dass es eine Möglichkeit geben muss, Regeln für mehrere CAN-IDs zusammenzufassen. Hierfür hat eine Regel zusätzlich einen prefix. Dieser Wert kann zwischen 0 (alle) und 29 (ein Teilnehmer) liegen, da eine CAN-ID maximal 29 Bit groß ist. Das „subnetzmask“ -Feld ist ein Platzhalter und kann beim Einlesen der Regeln automatisch aus dem prefix berechnet werden. Der Vorteil an dem Subnetting-Verfahren aus dem Layer 3 IP besteht darin, dass die Berechnung, ob eine CAN-ID sich im gleichen Subnetz befindet, einfach mit dem „Und“ Operator durchgeführt werden kann.

$$NetzID = CanId \& Subnetzmaske \tag{4.1}$$

Berechnung der Netz ID

Zusätzlich soll zwischen Remote und Data Frames unterschieden werden können, sodass ein Teilnehmer auf einem CAN-Bus einen Remote Frame über das Gateway-2017 verschicken kann und der Teilnehmer auf dem anderen CAN-Bus diese Anfrage mit einem Data Frame beantworten kann. Schlussendlich lässt sich mathematisch festhalten, dass auf eine CAN-ID 0 bis $\infty$ Regeln ausgeführt werden könnten, während einer Regel 1 bis $2^{29}$ CAN-IDs zugeordnet sind.

### 4.2.1 Darstellung der Regeln

Um die Regeln für das Gateway-2017 zu beschreiben, wurde das Comma-separated values (CSV)-Format mit einem Semikolon als Separator gewählt. Das hat den Vorteil, dass die Regeln komfortabel in einem Tabellenkalkulations-Programm erstellt und verändert werden können. Außerdem kann eine CSV Datei einfach von Code Generator eingelesen werden. Eine exemplarische Konfiguration sieht folgendermaßen aus:

| MAC-Adresse | CAN ID | Prefix | Mask | Timeout | Data | Remote |
|---|---|---|---|---|---|---|
| ff:2c:f5:26:58:11 | 9459 | 27 | | 50 | 1 | 0 |
| d7:78:ed:af:e3:a5 | 2047 | 29 | | 0 | 0 | 1 |

Tabelle 4.2: Beispiel einer Gatewaykonfiguration

### 4.2.2 Algorithmen

Alle Regeln müssen im Speicher des Gateway-2017 abgelegt sein und die passenden Regeln müssen schnell gefunden werden. Daher werden im diesem Kapitel zwei Verfahren zur Suche verglichen.

#### Lineare Suche

Die lineare Suche ist eine einfache Lösung zum Suchen einer Regel in einer Liste. Dazu wird jedes Element mit dem zu suchenden Element verglichen. Die Laufzeit der linearen Suche ist im Vergleich zu anderen Verfahren schlecht und liegt bei O(n). Ein Vorteil dieser Variante wäre, dass Regeln zur Laufzeit des Gateways nachgeladen werden könnten, da Regeln unsortiert vorgehalten werden können.

#### Binäre Suche

Bei der binären Suche muss die Liste sortiert sein. Dazu erfolgt eine fortlaufende Teilung in zwei Hälften. Das Problem bei der binären Suche nach einer CAN-ID ist, dass die gesuchte CAN-ID in der Liste vorhanden sein muss, da die binäre Suche sonst kein Ergebnis liefert. Die Laufzeit liegt bei O(log*n).

### 4.2.3 Entscheidung

Um eine Entscheidung zu treffen, wird mit folgenden Vereinfachungen gerechnet:

Eine CAN-ID ist 3 Bit groß und der prefix liegt zwischen 0 und 3.

Für die weitere Darstellung wird eine Regel wie folgt beschrieben:

(NetzID/prefix).

Folgende Regeln sind vorhanden und sollen durchsucht werden :
(1/3),(4/3),(6/3),(4/2),(7/3),(0/0)

Wenn nun nach der CAN-ID 5 gesucht werden soll, müssten im Ergebnis zwei Regeln gefunden werden:

(0/0) und (4/2),

da die CAN-ID 5 mit dem prefix (0 und 2) UND verknüpft als Ergebnis die NetzID (0 und 4) liefert.

#### Lineare Suche

Die lineare Suche durchsucht hierfür alle Elemente in der Liste. Die CAN-ID 5 wird mit dem jeweiligen prefix UND verknüpft. Wenn das Ergebnis gleich der NetzID des durchsuchten Elementes ist, ist eine zutreffende Regel gefunden worden. Bei der Linearen Suche werden beide Regeln gefunden. Die Laufzeit liegt bei O(n).

#### Binäre Suche

Für die binäre Suche wird nun die Liste aufsteigend nach der CAN-ID sortiert:
(0/0),(1/3),(4/2),(4/3),(6/3),(7/3)
Die Suche nach der CAN-ID 5 liefert kein Ergebnis, da es keine konkrete Regel für die CAN-ID 5 in der Liste gibt. So muss ab dem Punkt, wo die binäre Suche abbricht, eine lineare Suche gestartet werden. Dies hat zur Konsequenz, dass die Laufzeit sehr stark schwankt und die binäre Suche daher nur bei Regeln, mit dem prefix 29, eine vorhersagbare Laufzeit hat. So würde z.B. die Suche nach der 5 bei der (4/3) abbrechen und von dort eine lineare Suche bis zur (4/2) beginnen. Im besten Fall (Best Case) liegt die Laufzeit bei O(log*n) und im schlechtesten Fall (Worst Case) bei O(log*n+n).

#### Entscheidung

Die lineare Suche wird implementiert werden. Da sie aber mit einer Laufzeit von O(n) deutlich langsamer als die binäre Suche ist, wird sie nur zum Einsatz kommen, wenn die binäre Suche bei der Suche kein Ergebnis liefert. Die binäre Suche wird ebenfalls implementiert, da sie beim Einsetzen von Regeln mit dem prefix 29 die passenden Regeln schneller finden wird.

### 4.2.4 Datenstrukturen

Das folgenden Kapitel beschreibt die Datenstrukturen des Gateway-2017.

#### Buffer

Für die gewünschte Funktionalität des Gateways-2017 sind mehrere Buffer erforderlich. Eingehende CAN-Frames werden von der interrupt service routine (ISR) in einem Buffer zwischengespeichert. Für jede Ziel-MAC-Adresse wird ein Buffer mit 1500 Byte (maximale Nutzlast eines Ethernet-Frames) angelegt. Diese Buffer sind doppelt verkettet und nach dem Timeout sortiert. Wenn der Timeout des ersten Buffers kleiner gleich der Zeit des Gateways-2017 ist, wird dieser

Buffer verschickt. Da die Verkettung der Buffer bereits zum Start des Gateway-2017 sortiert ist, kann das Sortieren der Buffer mithilfe des Insertion Sort in einer Laufzeit von O(n) durchgeführt werden. Ein weiterer Buffer wird benötigt, um beim Empfang eines Ethernet-Frames die daraus entpackten CAN-Frames zu speichern. Die Größe dieses Buffers muss mindestens der maximalen Anzahl der zu entpackenden CAN-Frames entsprechen. Zusätzlich besitzt das Hilscher Board noch Hardware (HW)-Buffer für den Ethernet- und den CAN-Controller. Der Ethernet-Buffer kann in der derzeitigen Konfiguration maximal 2 Ethernet-Frames speichern, während der CAN-Controller 16 CAN-Frames speichern kann.

#### Management

Eine weitere Datenstruktur enthält alle Regeln, einschließlich der Dummy-Knoten (siehe Optimierung Konzept), die für den binären Suchbaum benötigt werden. Um die Suche in dieser Datenstruktur zu vereinfachen, sind die Regeln zusätzlich in einem binären Suchbaum verknüpft.

## 4.3 Nachrichtenbündelung

Der folgende Abschnitt behandelt das Bündeln der CAN-Frames in einem Ethernet-Frame. Das Problem beim Verpacken von CAN-Frames in einen Ethernet-Frame ist folgendes: Ein minimaler CAN-Frame hat eine Daten-Nutzlast von minimal 11 Bit (CAN-Identifier), während ein Ethernet-Frame minimal 84 Byte groß ist. Das bedeutet, eine solche Nachricht hätte einen Protokoll-Overhead von 97,4% . Selbst ein maximaler CAN-Frame im Extended-Format hätte mit Identifier (29Bit) und Datenfeld (8 Byte) einen Overhead von 77,9%. Um den Overhead eines solchen Ethernet-Frames zu verringern, könnten mehrere CAN-Frames in einem Ethernet-Frame gebündelt werden.

### 4.3.1 Aufbau der Nachricht

Nun soll das Verfahren zur Bündelung von Nachrichten ausgewählt werden. Der Overhead setzt sich aus dem Overhead des Ethernet-Frames und dem Overhead der einzelnen CAN-Frames zusammen.

$$overhead[\%] = \frac{Packetsize - Payload}{Packetsize} \tag{4.2}$$

Berechnung des Protokoll-Overheads

Im Folgenden werden vier verschiedene Verfahren in Bezug auf ihren Overhead und die Fehleranfälligkeit verglichen. Die Anzahl der CAN-Frames, die maximal in einen Ethernet-Frame passen, lässt sich mithilfe dieser Formel berechnen:

$$ACF = \frac{Payload}{SOCF} \tag{4.3}$$

| Variable | Verwendung |
|---|---|
| ACF | Anzahl CAN-Frames |
| SOCF | Größe eines CAN-Frames |

Maximal bündelbare CAN-Frames in einem Ethernet-Frame

Die Nutzlast ist die Anzahl an Bytes, die in einem Frame, abzüglich der Protokolldaten, für Daten zur Verfügung stehen.

### Gleiche Größe mit Header

Zunächst eine simple Lösung: Jeder CAN-Frame ist gleich groß und beginnt mit einem eindeutigen Header. Danach kommt die CAN-ID, die 4 Byte groß ist, um die 29 Bit CAN-ID eines erweiterten CAN-Frames zu speichern. Anschließend kommt das Datenfeld, das immer 8 Byte groß ist.

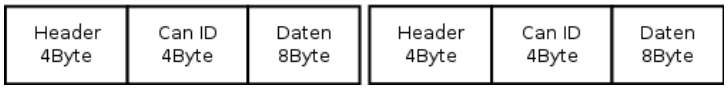



Abbildung 4.2: CAN-Frame mit fester Größe und Header vor jedem Frame im Ethernet-Datenfeld

Ein Vorteil dieser Lösung ist, dass der Ethernet-Dump während der Entwicklungsphase leicht vom Entwickler zu lesen ist. Der Beginn eines weiteren CAN-Frames ist mithilfe des Headers einfach zu finden. Ein Nachteil ist, dass das Verfahren aus CAN-Frames mit einem Datenfeld kleiner als 8 Byte, CAN-Frames mit einem 8 Byte großen Datenfeld macht. Dem Empfänger fehlt die Information über die korrekte Größe des Datenfeldes. Daher scheidet dieses Verfahren direkt aus, da sich das DLC- und das Daten-Feld wie im Soll-Konzept definiert, nicht verändern dürfen. Außerdem kann nur die gleiche maximale Anzahl an minimalen wie maximalen CAN-Frames gebündelt werden.

| Frame-Größe | Anzahl Nachrichten | Protokoll-Overhead |
|---|---|---|
| minimal | 93 | 91,67% |
| maximal | 93 | 29,11% |

Tabelle 4.3: Overhead und Anzahl Nachrichten „Gleiche Größe mit Header“

### Dynamisch mit mehreren Headern

Im Gegensatz zum vorherigen Verfahren kommt in dieser Variante ein weiteres Byte hinzu. Es soll die Größe des Datenfeldes dynamisch halten.


Abbildung 4.3: CAN-Frame mit dynamischer Größe und Header vor jedem Frame im Ethernet-Datenfeld

Somit ergibt sich der Vorteil, dass aus minimalen CAN-Frames keine maximalen CAN-Frames mehr werden. Auch lässt sich ein CAN-Frame gut im Ethernet-Dump finden, weil weiterhin vor jeder Nachricht ein Header steht. Wenn nur maximale CAN-Frames versendet würden, hätte dieses Verfahren den Nachteil, dass das Size Byte zusätzlicher Overhead wäre und daher nur noch 88 maximale CAN-Frames in einen Ethernet-Frame passten. Dieses Verfahren wurde bereits von Jan Depke für das Gateway-2015 verwendet.

| Frame-Größe | Anzahl Nachrichten | Protokoll-Overhead |
|---|---|---|
| minimal | 166 | 85,14% |
| maximal | 88 | 33,39% |

Tabelle 4.4: Overhead und Anzahl Nachrichten „Dynamisch mit mehreren Headern“

### Dynamisch ohne Header

Wenn feststeht, dass keine anderen Ethernet-Frames an das Gateway-2017 gehen (MAC-Adresse des Gateway), kann auf den Header verzichtet werden.


Abbildung 4.4: CAN-Frame mit dynamischer Größe ohne Header vor jedem Frame im Ethernet-Datenfeld

Dadurch verringert sich der Overhead und es können bis zu 300 minimale CAN-Frames in einem Ethernet-Frame gespeichert werden. Falls es doch zu Nachrichten kommt, die keine CAN-Frames enthalten, können diese nicht zuverlässig erkannt werden. Nur wenn ungültige Werte im Size-Feld (Wert größer 8) erkannt werden, könnten diese Nachrichten verworfen werden. Ansonsten sendet das Gateway-2017 willkürlich und unvorhersehbare CAN-Frames auf den CAN-Bus.

| **Frame-Größe** | **Anzahl Nachrichten** | **Protokoll-Overhead** |
|---|---:|---:|
| minimal | 300 | 73,14% |
| maximal | 115 | 12,67% |

Tabelle 4.5: Overhead und Anzahl Nachrichten „Dynamisch ohne Header“

### Dynamisch mit globalem Header

Bei diesem Verfahren wird vor alle CAN-Frames ein Header gesetzt.

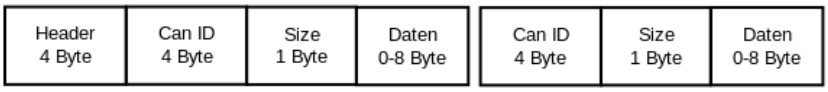



Abbildung 4.5: CAN-Frame mit dynamischer Größe und einem globalen Header vor allen Frames im Ethernet-Datenfeld

Dies hat den Vorteil, dass der Frame nur einmal den Protokoll-Overhead um 4 Byte erhöht. Es können weniger CAN-Frames versendet werden als beim Verfahren ohne Header, aber mehr als bei allen anderen Verfahren. Als Nachteil bleibt zu nennen, dass sich das Verfahren schwieriger implementieren lässt, da der Ethernet-Dump nur noch mit Mühe (Zählen der Bytes) vom Entwickler zu lesen ist. Bei einem minimalen Ethernet-Frame kann auch das Ende des letzen CAN-Frames nicht erkannt werden.

| Frame-Größe | Anzahl Nachrichten | Protokoll-Overhead |
|---|---|---|
| minimal | 299 | 73,22% |
| maximal | 115 | 12,91% |

Tabelle 4.6: Overhead und Anzahl Nachrichten „Dynamisch ohne Header“

#### Entscheidung

Für die Entwicklung des Gateways-2017 wurde das Verfahren **„Dynamisch mit mehreren Headern“** gewählt, da es einen geringen Protokoll-Overhead besitzt und sich relativ simpel implementieren lässt. Außerdem kann das Gateway auch in Zukunft leicht auf weitere Protokolle, wie Flexray oder Most, portiert werden. So könnte der Header die Information beherbergen, welchen Typs der nachfolgende Frame ist. Außerdem ist das Gateway-2017 damit kompatibel mit dem Gateway-2015.

## 4.4 Fragmentierung gebündelter Nachrichten

Im folgenden Teil wird sich mit der Aufteilung von CAN-Frames auf mehrere Ethernet-Frames beschäftigt. Das kann hilfreich sein, um den Protokoll-Overhead noch weiter zu verringern. So passt z.B. ein maximaler CAN-Frame mit der Größe von 17 Byte nicht mehr in einen Ethernet-Frame, der bereits mit 1484 Byte gefüllt ist. Der Protokoll-Overhead erhöht sich dementsprechend um 1%. Wenn man den CAN-Frame aufteilt, muss dem Empfänger mitgeteilt werden, dass es sich um ein Fragment handelt und nicht um einen kompletten Frame. Hierzu wäre mindestens ein zusätzliches Bit nötig, was den Protokoll-Overhead wiederum um 11 (88 Frames)-20,75 (166 Frames) Bytes erhöhen würde, der somit sogar teilweise über dem Gewinn von 16 Byte liegt. Außerdem müssten beim Empfänger zusätzliche Buffer eingerichtet werden um die Fragmente zwischenzuspeichern. Zu klären wäre, wie viele solcher Buffer verfügbar sein sollen, ob dynamisch Speicherplatz alloziert werden soll, der für Angriffe missbraucht werden könnte, und wann und ob ein Fragment aus dem Speicher gelöscht werden soll, wenn vermutlich kein zugehöriges Fragment mehr folgt.

Aufgrund beschriebener Probleme und dem geringen Gewinn oder sogar Verlust wurde sich gegen eine solche Implementierung entschieden. Sollte das Gateway-2017 für ein anderes Feldbussystem portiert werden, könnte es sich mit größeren Frames rentieren. So erhielte man beispielsweise bei Flexray mit einem 254 Byte großen Datenfeld und einer 11 Bit großen Frame ID einen zusätzlichen Overhead von bis zu 16,9%, wenn keine Fragmentierung durchgeführt wird.

## 4.5 Duplikate

In der Konzeptionsphase wurde die Ausfilterung doppelter Nachrichten erörtert.

Annahme: eine ECU sendet zyklisch Informationen auf den CAN-Bus. Wenn auf einem zweiten CAN-Bus diese Nachricht nur mit einer größeren Periode benötigt wird, könnte das Gateway-2017 einen Teil der Nachrichten herausfiltern. Wenn zusätzlich zur CAN-ID auch noch das Datenfeld geprüft würde, könnten die Nachrichten der zyklisch sendenden ECU so gefiltert werden, dass Nachrichten nur bei deren Veränderung weitergeleitet werden. Das Filtern von Duplikaten würde aber die Laufzeit der Gateway-Logik deutlich erhöhen. Es müsste jede eingehende Nachricht mit allen bereits zwischengespeicherten Nachrichten verglichen werden. Aus diesem Grund wurde sich gegen die Umsetzung einer solchen Funktion entschieden.

## 4.6 CAN-Features

In diesem Abschnitt wird beschrieben, wie sich die Kommunikation zwischen zwei CAN-Bussen über einen Ethernet-Backbone im Bezug auf die Features des CAN-Busses verändert.

### Error Frame

Ein Error Frame wird gesendet, wenn ein CRC-Fehler auf dem Bus auftritt, was meist bei physikalischen Problemen auf dem Bus oder beim Senden von CAN-Frames mit dem selben Identifier, aber unterschiedlichem Datenfeld, auftritt. Daher macht es keinen Sinn, einen Error Frame an einen anderen CAN-Bus weiterzuleiten, der ja einen anderen physikalischen Link verwendet. Das Gateway-2017 wird keine Error Frames weiterleiten. Um zu verhindern, dass CAN-Frames verloren gehen, kann das Gateway-2017 so eingestellt werden, dass es CAN-Frames mehrfach versendet, bis sie zugestellt werden konnten.

### Ack-Slot

Der Acknowledge-Slot gibt weiterhin an, dass ein Teilnehmer den Frame empfangen hat. Teilnehmer ist nun auch das Gateway, das den CAN-Frame abhängig von der Konfiguration gegebenenfalls nicht weiterleitet. Ein dominantes Acknowledge Bit bedeutet so nur die Bestätigung des Gateways-2017, einen Frame empfangen zu haben ohne dass ein „anderer“ Teilnehmer den CAN-Frame empfangen hat.

### Remote Frame

Der Remote Frame wird unterstüzt, indem bei der Nachrichtenbündelung im Size Byte zusätzlich ein Bit gesetzt wird, das die Information enthält, ob es sich bei diesem Frame um ein Data

oder einen Remote Frame handelt. So kann ein Remote Frame einen Data Frame auf einem anderen CAN-Bus anfordern.

## 4.7 Interpretation oder Kompilation

Im folgenden Abschnitt werden die Vor- und Nachteile von Interpretation und Kompilation beschrieben und die Entscheidung erläutert.

### 4.7.1 Interpretation

Die Interpretation liest die Konfiguration des Gateway zur Initialisierungsphase oder zur Laufzeit ein und interpretiert die Regeln daraufhin.

- Vorteile
  - Bei der Veränderung von Regeln muss der Code nicht neu kompiliert werden.
  - Regeln können zur Laufzeit nachgeladen werden.
  - Nur ein Programm ist nötig.
- Nachteile
  - Regelwerk muss zur Laufzeit sortiert werden oder linear durchsucht werden.
  - Speicher-Allozierung erfolgt zur Laufzeit.

### 4.7.2 Kompilation

Bei der Variante „Kompilation“ wird ein Regelsatz von einer Software eingelesen und aus diesem Regelsatz der Source-Code generiert.

- Vorteile
  - Regelwerk kann vom Code Generator sortiert werden.
  - Größe und Anzahl der Buffer sind zur Kompilationszeit bekannt.
- Nachteile
  - Bei der Veränderung von Regeln muss der Code neu kompiliert werden.
  - Zwei Programme sind nötig.

### 4.7.3 Entscheidung

Für die Umsetzung des Gateway-2017 wurde sich für die **Kompilationsvariante** entschieden.

Diese Entscheidung wurde aus verschiedenen Gründen getroffen:

Eine Embedded Software (SW), die zur Laufzeit Speicher alloziert, zeigt die daraus resultierenden Probleme erst zur Laufzeit, im besten Fall beim Starten, im schlimmsten Fall erst Jahre nach der Inbetriebnahme. Eine Embedded SW, die vor der Laufzeit Speicher alloziert, zeigt resultierende Probleme bereits beim Kompilieren oder spätestens beim Starten der Software. Durch die Interpretation der Regeln vom Code Generator werden Code-Pfade, die für die vorhandenen Regeln nicht benötigt werden, bereits vor der Laufzeit vom Code Generator gar nicht erst erzeugt oder durch den Preprocessor auskommentiert. Durch die vorherige Sortierung der Regeln ist das Suchen der zugehörigen Regel zur Laufzeit schneller durchzuführen.

Da das Gateway-2015 als Verfahren „Interpretation“ einsetzte und damit nicht die maximale Bandbreite eines CAN-Busses erreicht wurde, wird das Gateway-2017 als Verfahren „Kompilation“ einsetzen.

## 4.8 Gateway

Die SW des Gateways-2017 besteht aus dem Code, der vom Code Generator generiert wird, aus der Konfiguration des Schedulers und dem Management von Regeln und Buffern.

Genutzt werden die vorhandenen application programming interfaces (APIs) für Ethernet und CAN.

Als Hardware wird der in der IST-Analyse vorgestellte µC beibehalten.

## 4.9 Generator

Der Code Generator generiert statischen Code für die in der Konfigurationsdatei (Konfigurationen/config.csv) vorgegebene Konfiguration. Der Code Generator wird in der Programmiersprache JAVA entwickelt. Diese Sprache ist bekannt und zur Bearbeitung von Zeichenketten (String Bibliotheken) besonders gut geeignet. So ist es einfach, Strings zu konkatenieren oder mit regulären Ausdrücken zu bearbeiten.

Über eine zweite Konfigurationsdatei sollen Einstellungen für den Code Generator möglich sein.

# 5 Optimierung Konzept

Erste Messungen der Laufzeit in Abhängigkeit von der Anzahl der Regeln, ergaben stark schwankende Ergebnisse. Die maximalen Laufzeiten waren nicht zufriedenstellend. Daher wurde das Konzept um einen weiteren Algorithmus zur Regelsuche erweitert: den binären Suchbaum.

### Binärer Suchbaum

Anstelle einer Liste sind hier alle Elemente in einen Baum verknüpft. Der binäre Suchbaum hat eine Laufzeit von O(log*n). Durch den prefix liegt die maximale Höhe des Suchbaumes bei 30. Es müssen Dummy-Regeln hinzugefügt werden um den Suchbaum zu komplettieren. Das hat zur Folge, dass der Speicherverbrauch durch die zusätzlichen Regeln steigt. Zur besseren Veranschaulichung wurde der Suchbaum im Folgenden auf die Tiefe 4 reduziert:

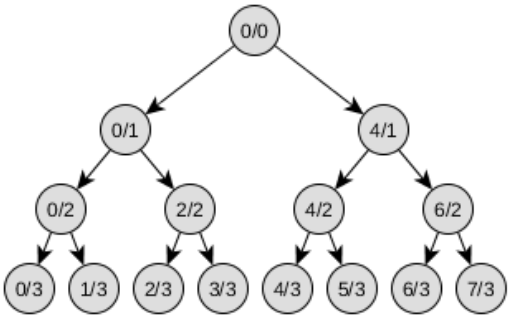
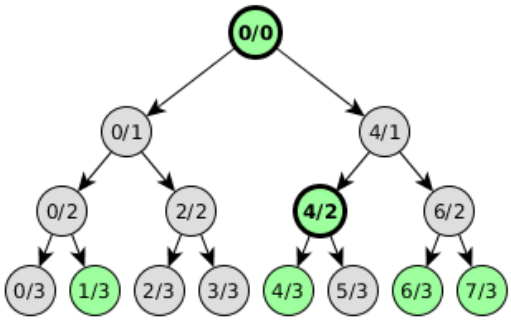



Abbildung 5.1: Binärer Suchbaum mit der Suchtiefe 4

Der Algorithmus zur Suche im binären Suchbaum wird mit den gleichen Regeln und der gleichen Suche durchgeführt (Abschnitt 4.2.3, Seite 19). Die Regeln werden im nächsten Bild als grüne Knoten markiert.



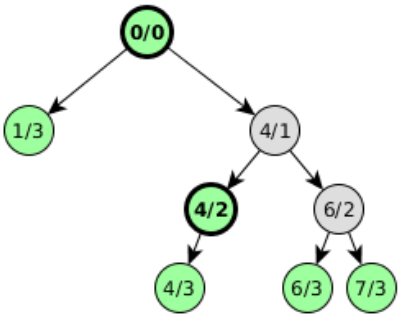

**Legende**

| | |
|---|---|
| Grün | Regeln |
| Grau | Dummy-Knoten |
| Fett | gesuchte Regeln |

Abbildung 5.2: Binärer Suchbaum mit eingefügten Regeln

Zunächst startet die Suche im Startknoten (0/0). Da dieser eine Regel enthält, wird diese ausgeführt. Um zu entscheiden, ob der linke oder der rechte Pfad weiterverfolgt wird, wird die gesuchte CAN-ID 5 mit dem prefix 1 „Und“ verknüpft. Diese Berechnung ergibt die Zahl 4, woraus sich ergibt, dass die Suche im rechten Knoten(4,1) weitergeht. Von hieraus geht es nach links in den Knoten (4/2). Die dort enthaltene Regel wird ausgeführt. Das Ende des Algorithmus ist mit dem Knoten (5/3) erreicht. Beide Regeln wurden gefunden: (0/0) und (4/2).

Die Knoten, die keine Regeln enthalten, aber trotzdem vorhanden sind, werden in dieser Arbeit als Dummy-Knoten bezeichnet. Um den Speicherverbrauch, der durch die Dummy-Knoten verursacht wird, zu verringern, können zunächst alle Dummy-Knoten gelöscht werden, die keinen Nachfolger haben. Anschließend können die Dummy-Knoten, die nur einen Nachfolger haben, gelöscht werden, indem sie durch dessen Nachfolger ausgetauscht werden. Diese Schritte werden so oft wiederholt bis kein Knoten mehr entfernt werden kann. Hieraus entsteht der optimierte binäre Suchbaum:




**Legende**

| | |
|---|---|
| Grün | Regel |
| Grau | Dummy-Knoten |
| Fett | gesuchte Regeln |

Abbildung 5.3: Optimierter binärer Suchbaum

Wie anhand des Beispiels in Abbildung 5.3 zu sehen, lassen sich nicht alle Dummy-Knoten entfernen. Da (6/2) zwei Nachfolger besitzt, muss dieser erhalten bleiben. Da (4/1) zusätzlich (4/2) als Nachfolger hat, bleibt auch dieser Dummy-Knoten erhalten. Durch diesen Umstand wird der Speicherbedarf für den optimierten- und den nicht-optimierten-Suchbaum höher sein als bei der binären Suche. In der Auswertung wird anhand der Code-Zeilen der Speicherbedarf ermittelt.

# 6 Umsetzung

Es folgt die Umsetzung von Code Generator und Gateway-2017.

## 6.1 Code Generator

In diesem Abschnitt wird beschrieben, wie der Code Generator implementiert wurde, wie dieser verwendet und konfiguriert werden kann.

### Implementierung des Code Generators

Der Code Generator besteht aus mehreren Klassen: Klassen für die Erstellung des Source-Codes, für das Einlesen der Konfigurationdateien und Klassen für Hilfsfunktionen. Die Klasse „FileHandling“ stellt Hilfsfunktionen zum Einlesen und Schreiben von Textdateien zur Verfügung. In der Klasse „GatewayConfig“ werden die Parameter aus der Konfigurationsdatei eingelesen und gespeichert. Die „Tree“ Klasse wird verwendet um den binären Suchbaum zu erstellen. Die Klassen „Rule“ und „MergedRule“ speichern die eingelesenen Regeln. Die Klassen, die den Source-Code erzeugen, haben die beiden Funktionen „writeSourceFile“ und „writeHeaderFile“ und sind von der Klasse „Generator“ abgeleitet. Folgende Klassen generieren folgende Source- und Header-Dateien:

| **Klasse** | **Source- & Header-Dateien** | **Anwendung** |
|---|---|---|
| FunctionGenerator | function.c & function.h | speichert die Funktionspointer für die Regeln |
| RulesGenerator | importedRules.c & importedRules.h | lädt die Regeln in den Speicher des Gateways-2017 |
| BufferGenerator | buffer.c & buffer.h | lädt die für die Regeln erforderlichen Ethernet-Buffer |
| TreeGenerator | tree.h | legt fest, ob die binäre Suche oder der binäre Suchbaum angewendet wird |

Tabelle 6.1: Beschreibung der Klassen des Code Generators und Nutzen der generierten Source- und Header-Dateien

### Benutzung des Code Generators

Zunächst sind folgende Abhängigkeiten für den Code Generator erforderlich:

| | |
|---|---|
| Benötigte Software: | Java Compiler (>=7.0 und java/javac/javadoc in $PATH) und make |
| Optionale Software: | GNU Compiler Collection (GCC) (für XML Ausgabe) |

Nun kann die Software kompiliert werden:

```
make  #Compiliert die Java Sources
make install  #erstellt das Start Skript launch.sh
make doc # erstellt die Javadoc Dokumentation
```

Im Ordner befindet sich jetzt ein Skript „launch.sh“, mit dem der Generator verwendet werden kann. Falls eine alternative Konfigurationsdatei verwendet werden soll, kann diese in dem Skript angegeben werden. Standardmäßig befindet sich eine Beispielkonfiguration (config.txt) im Hauptverzeichnis des Code Generators.

### Konfiguration des Code Generators

In der Konfigurationsdatei des Code Generators kann eingestellt werden, an welcher Stelle die verschiedenen Source- und Header-Dateien angelegt werden sollen.

Außerdem kann eingestellt werden, ob eine zufällige Regel-Konfigurationsdatei für das Gateway-2017 erstellt werden und wie viele Einträge diese Regel-Konfigurationsdatei haben soll.

Ob bei der Regelsuche die binäre Suche oder der binäre Suchbaum verwendet wird, kann mit dem Parameter „rulesAsTree“ konfiguriert werden. In welchem Format die Parameter zu setzen sind und welche Parameter es zusätzlich gibt, wird in den Kommentaren der Konfigurationsdatei beschrieben.

## 6.2 Gateway

In diesem Abschnitt wird beschrieben, wie das Gateway-2017 implementiert wurde und wie es verwendet werden kann.

### Implementierung des Gateways

Der Quellcode besteht aus der „config.c“, in der sich die Konfiguration des Schedulers befindet. Diese Datei befindet sich im Ordner „Application“. In diesem Ordner befindet sich ebenfalls der

Ordner „generated“. Hier müssen die Dateien des Code Generators abgelegt werden. Außerdem befindet sich hier der Ordner „written“, in dem sich der geschriebene Code befindet. Der Programm-Code des Gateways-2017 wurde doxygen kompatibel dokumentiert.

#### Benutzen des Gateways

Der Code muss kompiliert werden:

```
make  #Compiliert den Source-Code
make doc # erstellt die doxygen Dokumentation
```

Im Hauptverzeichnis des Gateways-2017 befindet sich nun die Binary Datei „netx.rom“. Diese kann auf eine Secure Digital (SD)-Karte kopiert werden oder mit dem Tool „bootwizard“ permanent auf das Hilscher Board geflasht werden.

### 6.2.1 Scheduler

Das Gateway-2017 besteht aus 4 Tasks, die vom Scheduler periodisch aufgerufen werden. Zusätzlich existiert eine ISR, die für eingehende CAN-Frames zuständig ist. Ein grobes Schedule wird entworfen.

#### Zuordnung und Buffering von CAN-Frames

Der Task „fillBufferToRte()“ liest CAN-Frames aus dem Buffer „CAN_TO_RTE“ aus und sucht über die binäre Suche oder mit dem binärem Suchbaum nach auszuführenden Regeln und führt zutreffende Regeln aus.

#### Sende Buffer via RTE

Der Task „sendRTE()“ prüft, ob der Timeout eines Buffers kleiner gleich der Systemzeit ist. Sollte dies der Fall sein, kopiert der Task den Buffer in einen Ethernet-Frame. Danach wird der Buffer wieder ans Ende der Liste einsortiert und die Systemzeit inkrementiert.

#### Empfangen und Entpacken von RTE-Frames

Wenn ein Ethernet-Frame im Empfangs HW-Buffer des Ethernet-Controllers gespeichert ist, entpackt dieser Task alle CAN-Frames aus dem Ethernet-Frame und speichert sie im Buffer „RTE_TO_CAN“.

### Senden von CAN-Frames

Solange der HW-Buffer für CAN-Frames nicht voll ist und sich noch Nachrichten im Buffer „RTE_TO_CAN“ befinden, kopiert dieser Task alle zwischengespeicherten CAN-Frames in den HW-Buffer.

### Sende Ethernet HW-Buffer

„Sende Ethernet HW-Buffer“ ist die Funktion „event_sendTT()“, die zum Scheduler gehört und einen Ethernet-Frame sendet. Indem sie vom Scheduler aufgerufen wird, ist garantiert, dass der Frame im passenden Zeitslot versendet wird.

### ISR

Die ISR wird beim Eintreffen von CAN-Frames ausgeführt. Die CAN-Frames aus dem HW-Buffer des CAN-Controllers speichert die ISR im Buffer „CAN_TO_RTE“.

### Sync Window

Die Funktion „TTE_SCHED_SYNC“ sorgt für die Synchronisierung der Schedules der Kommunikationsteilnehmer, sodass das Event „Sende Ethernet HW-Buffer“ im passenden Zeitslot ausgeführt wird.

### Timing

Da zunächst nicht bekannt ist, wie lange die Laufzeit dieser Tasks sein wird, wurde ein Schedule mit großem Puffer erstellt. Das Testgerät zum Senden von CAN-Frames kann minimal mit einer Periode von 1 ms senden. Daher wurde die Schedule Periode ebenfalls mit 1 ms konfiguriert und die Tasks gleichverteilt jeweils 1 mal in das Schedule eingetragen.

# 7 Evaluierung und Validierung

In diesem Kapitel werden die Tests erläutert, die mit dem System durchgeführt wurden, die Übereinstimmung der Wunsch-, Muss-, und Abgrenzungskriterien werden besprochen und die Laufzeit der Software Komponenten ermittelt und ausgewertet.

## 7.1 Verfikation und Tests

Für die Verifikation der Funktionsweise des Gateways-2017 wurde folgender Testaufbau verwendet:

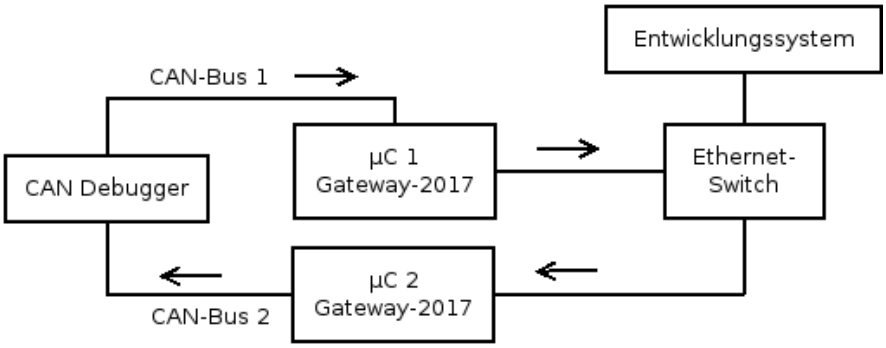



Abbildung 7.1: Schematische Darstellung des Testaufbaus

Der CAN-Debugger sendet eine vorab erstellte Liste mit CAN-Frames an den CAN-Bus 1. Der µC 1 verpackt diese CAN-Frames und verschickt sie als Ethernet-Frames über den Ethernet-Switch an den µC 2. Der µC 2 entpackt die CAN-Frames aus den Ethernet-Frames und versendet diese CAN-Frames über den CAN-Bus 2 zurück an den CAN-Debugger. Um zu prüfen, ob die Ethernet-Frames korrekt sind, können diese auf dem Entwicklungssystem mithilfe von Wireshark mitgeschnitten werden. Der CAN-Debugger ist die CANcardXL von der Firma Vector mit einer Messgenauigkeit von 10 µs. Hiermit können beliebige CAN-Frames verschickt sowie ein Trace aufgezeichnet werden. Ein Trace ist eine Aufzeichnung des Netzwerkverkehrs, in dem sich die Informationen zu den aufgezeichneten Frames befinden, darunter deren relative Ankunftszeit.

### 7.1.1 Buffering

Um das Buffering zu testen, müssen verschiedene Kriterien untersucht werden. Um zu prüfen, ob alle Nachrichten eingehen und in der richtigen Reihenfolge bleiben, wird eine Liste an CAN-Nachrichten erzeugt. Diese zählt die CAN-ID hoch und erstellt einen zufälligen Payload. Da die empfangenen Daten (DLC, Daten und ID-Feld) gleich den gesendeten und auch deren Reihenfolge gleich geblieben ist, erwies sich dieser Test als erfolgreich.

Um zu prüfen, ob der Timeout (aus der Konfigurationsdatei) korrekt arbeitet, wurden nun Regeln mit unterschiedlichen Timeouts angelegt. Die CAN-Frames, die einen kürzeren Timeout hatten, überholten nun die CAN-Frames, die zeitlich früher am Gateway-2017 eintrafen, aber einen längeren Timeout hatten. Auch das Einsortieren eines Frames mit einem kleineren Timeout als der Timeout des Buffers, in den dieser Frame einsortiert wurde, sorgte dafür, dass der Buffer den kleineren Timeout übernahm.

Damit verhält sich das Buffering wie im Konzept unter Regeln (4.2) definiert.

### 7.1.2 Regeln

Zunächst wurde das Gateway-2017 mit nur einer Default Regel (Regel (0/0)) konfiguriert. Diese umfasst alle CAN-IDs. Es wurde überprüft, ob alle CAN-Frames in der gleichen Reihenfolge bleiben und ob die Anzahl der gesendeten CAN-Frames gleich der Anzahl der empfangenen CAN-Frames ist. Die Anzahl und die Reihenfolge der gesendeten CAN-Frames entsprach exakt den empfangenen CAN-Frames.

Um zu prüfen, ob die Einstellungen beim Timeout korrekt sind, wurde ein CAN-Frame versendet und verifiziert, dass der zeitliche Abstand auf dem CAN-Debugger gleich der Einstellung im Timeout ist. Es wurden mehrere Regeln im Gateway-2017 hinterlegt, die auf einen CAN-Frame zutreffen, um sicherzustellen, dass die Anwendung von mehreren Regeln korrekt funktioniert. Ob alle Regeln angewendet wurden, konnte anhand der Ethernet-Frames, die das Entwicklungssystem mitschneidet, verifiziert werden. Das Verhalten bei keiner Regel wurde verifiziert, indem CAN-Frames an das Gateway-2017 geschickt wurden und das Gateway-2017 daraufhin keine Ethernet-Frames versendete. Auch wenn spezielle Regeln nur einen Teil der CAN-Frames abdecken, wurden nur Ethernet-Frames verschickt, die diese CAN-Frames enthalten. Somit waren alle Tests erfolgreich abgeschlossen.

## 7.2 Wunsch- und Muss-Kriterien

Im welchem Umfang die in Abschnitt 4.1 aufgestellten Kriterien erfüllt wurden, wird nachfolgend tabellarisch dargestellt.

| Typ | Beschreibung | Siehe/Erklärung |
|---|---|---|
| Muss | maximale Bandbreite CAN | Seite 48 Abschnitt 7.4.6 |
| | CAN-Frames bündeln | Seite 37 Abschnitt 7.1.1 |
| | Reihenfolge bleibt unverändert | Seite 37 Abschnitt 7.1.1 |
| | Daten, DLC- und ID-Feld unverändert | Seite 37 Abschnitt 7.1.1 |
| Wunsch | Ack-Slot CAN-Bus übergreifend | Seite 26 Abschnitt 4.6 |
| | Error Frame CAN-Bus übergreifend | Seite 26 Abschnitt 4.6 |
| | Remote Frame CAN-Bus übergreifend | Seite 26 Abschnitt 4.6 |
| | Duplikate filtern | Seite 26 Abschnitt 4.5 |
| Abgrenzung | kein CAN- nach CAN-Gateway | wurde nicht implementiert |
| | kein Ethernet- nach Ethernet-Gateway | wurde nicht implementiert |
| | kein Ethernet- über CAN-Tunneln | wurde nicht implementiert |

Tabelle 7.1: Kriterien, die das Gateway-2017 erfüllt

Grün markierte Kriterien wurden voll erfüllt. Rot markierte Kriterien wurde nicht erfüllt und gelbe Kriterien sind teilweise erfüllt worden. In der dritten Spalte findet sich ein Verweis, wo dieses Kriterium detaillierter erläutert wird.

## 7.3 Simulation

Die CoRE Gruppe hat mithilfe der Abstract Network Description Language (ANDL) verschiedene Simulationen für Omnet++ entwickelt, um die Netzwerk-Kommunikation im Prototypenfahrzeug simulieren zu können (Steinbach u. a., 2016), z.B. die Simulation „majorNetwork“:

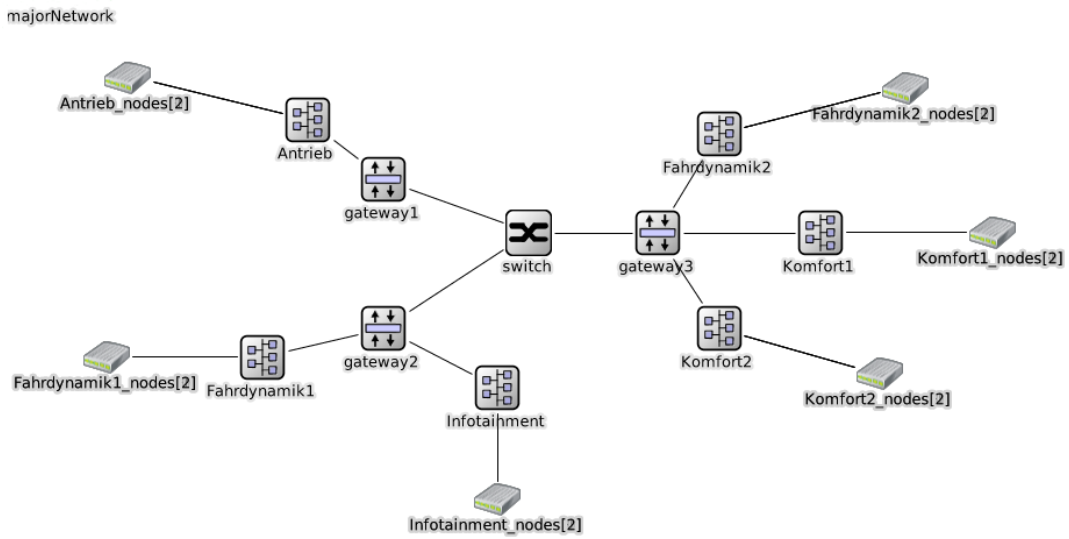



Abbildung 7.2: Simulation „majorNetwork“ (Fahrzeugnetz) in Omnet++

Die folgenden Abschnitte handeln davon, welche Anforderungen das Gateway-2017 im Vergleich zur Simulation „majorNetwork“ erfüllt und wie die Simulation oder das Gateway-2017 angepasst werden müssten, damit gleiche Ergebnisse erzielt werden.

### 7.3.1 Erfüllte Anforderungen

In der Simulation wird eine Nachrichtenbündelung genutzt. Diese Frames heißen in der Simulation „GatewayAggregationMessages“. Dieses Feature wird vom Gateway-2017 unterstützt.

In Abhängigkeit von der CAN-ID wird ein CAN-Frame an ein über die MAC-Adresse spezifiziertes Gateway auf dem Ethernet-Backbone verschickt. Dieser Vorgang wird durch das Regelwerk unterstützt.

### 7.3.2 Nicht erfüllte Anforderungen

„gateway2“ und „gateway3“ können zusätzlich CAN-Frames von einem CAN-Bus auf einen anderen übertragen. Die SW des entwickelten Gateway-2017 unterstützt derzeit nur eine CAN-Schnittstelle.

### 7.3.3 Umbau Simulation

Würde man die Simulation anpassen, könnte ein Gateway ausreichen, das nur eine CAN-Schnittelle unterstützt. Es müsste für jeden CAN-Bus ein eigenes Gateway-2017 vorhanden

sein. Die CAN-Frames zwischen den CAN-Bussen müssten alle über den Ethernet-Backbone übertragen werden. Falls dies aus Sicherheitsgründen nicht gewünscht ist, könnte die Kommunikation zwischen den CAN-Bussen über Virtual Local Area Networks (VLANs) abgesichert werden. Als Beispiel wäre folgende VLAN-Konfiguration denkbar:

| VLAN | Bus 1 | Bus 2 | Bus 3 |
|---|---|---|---|
| VLAN1 | Fahrdynamik2 | Komfort1 | Komfort2 |
| VLAN2 | Fahrdynamik1 | Infotainment | |
| VLAN3 | alle | | |

Tabelle 7.2: Beispielhafte VLAN-Konfiguration

So wäre die Kommunikation, die zurzeit Gateway intern behandelt wird, auf ein VLAN beschränkt und die Kommunikation zwischen den derzeitigen Gateways wäre weiterhin in einem VLAN.

### 7.3.4 Erweiterung des Gateways

Der verwendete µC von Hilscher hat nur zwei CAN-Controller. Daher wäre die Portierung auf einen anderen µC nötig. Eine zweite CAN-Schnittstelle wäre bei der Kommunikation von CAN nach CAN durch eine Erweiterung des Regelwerks möglich. Der Code Generator würde dann zusätzliche Regeln erstellen, die diese CAN-Frames direkt in den Buffer „RTE_TO_CAN“ einsortieren. Da bei der Suche nach Regeln weiterhin nur ein Funktionspointer aufgerufen wird, sollte dies keine negativen Auswirkungen auf die Performance haben. Ein Problem wäre der Empfang von Ethernet-Frames. Hier wäre es am einfachsten, wenn das Gateway-2017 für jeden angeschlossen CAN-Bus eine andere Ethernet-Empfangsadresse verwenden würde. Ein CAN-Frame, der an beide CAN-Busse gehen soll, muss dann zweimal übermittelt werden. Außerdem müssten beide Ethernet-Empfangs-Buffer (für jede Ethernet-Empfangsadresse) nach neuen Frames durchsucht werden. Das würde die Laufzeit beim Verarbeiten von empfangenen Ethernet-Frames erhöhen.

## 7.4 Zeitliches Verhalten des Gateways

Für die zeitlichen Messungen wurde getestet, mit welchem Optimierungsflag die besten Ergebnisse erzielt werden. Mit dem Optimierungsflag „-O2“ kann der Linker den Code nicht linken. Der Code läuft ordnungsgemäß mit „-O1“ und wurde daher mit diesem Flag kompiliert. Im Vergleich zu „-O0“ verbessert sich die Laufzeit der einzelnen Funktionen. Vor allem Funktionen, in denen die String Libaries (memcpy, memset) eingesetzt werden, profitieren von dem

Optimierungsflag. Ein Vergleich zwischen „-O1“ und „-O0“ beim Verpacken von CAN-Frames bei einem Versand ohne Nachrichtenbündelung erbrachte folgende Messergebnisse:

| Funktion | -O1 | -O0 |
|---|---|---|
| ISR | 3,9 µs | 7,7 µs |
| fillBufferToRTE | 13,7 µs | 24,0 µs |
| sendRTE | 18,5 µs | 26,5 µs |

Tabelle 7.3: Vergleich der Laufzeit zwischen -O0 und -O1

Um diese Messungen durchzuführen, wird am Anfang der zu vermessenden Funktionen ein General Purpose Input Output (GPIO) Ausgang auf „HIGH“ gesetzt. Am Ende dieser Funktion wird dieser Ausgang wieder auf „LOW“ zurückgesetzt. Die Flanken lassen sich mit einem Oszilloskop erfassen. Hierzu wurde ein Oszilloskop der Firma Tektronix, vom Typ MS04054B verwendet. Dieses Oszilloskop hat eine maximale Sample Rate von 2,5 Giga Samples und erreicht damit eine Genauigkeit von 0,4 ns, was für die Messungen im µs Bereich ausreicht. Außerdem wurden zwei CAN-Debugger Geräte verwendet. Die „CANcardXL“ der Firma Vector wird für das Senden von CAN-Frames eingesetzt. Ergänzend wurde das „VN5610“ von der Firma Vector eingesetzt, das CAN- und Ethernet-Frames messen kann und im weiteren Verlauf als CAN-Ethernet-Debugger bezeichnet wird. Die Genauigkeit liegt hier bei 1 µs. Für die Latenz-Messungen wurde der CAN-Ethernet-Debugger und für die Laufzeit-Messungen das Oszilloskop verwendet.

### 7.4.1 Laufzeit der ISR

Die Laufzeit der ISR liegt bei O(1) bzw. in Zahlen bei 3,9 µs, da die ISR beim Empfang jedes CAN-Frames ausgeführt und jeder CAN-Frame in den Buffer „CAN_TO_RTE“ gespeichert wird. Da das Gateway-2017 nur eine ISR besitzt, kann es nicht durch andere ISRs zu einer Warteschlange kommen. Die Größe des CAN-Frames hat keine Auswirkungen auf die Laufzeit, da die Datenstrukturen der CAN-API statisch sind. Das Datenfeld ist immer 8 Byte groß. Um diese Behauptung zu belegen, wurde die ISR zusätzlich mit dem Oszilloskop vermessen. Die Messergebnisse beim Empfang von CAN-Frames mit 0 Byte Nutzlast waren identisch mit den Messergebnissen bei CAN-Frames mit einer Nutzlast von 8 Byte.

### 7.4.2 Laufzeit der Bufferzuordnung

Im folgenden Kapitel geht es darum, welche Auswirkungen die Anzahl der Regeln auf die Laufzeit des Gateways-2017 hat. Der Task „fillBufferToRTE“ liest einen CAN-Frame ein, sucht

nach auszuführenden Regeln und führt diese aus. Dieser Task wurde vermessen, um zu prüfen, wie sich die Laufzeit durch die Anzahl der Regeln verändert:

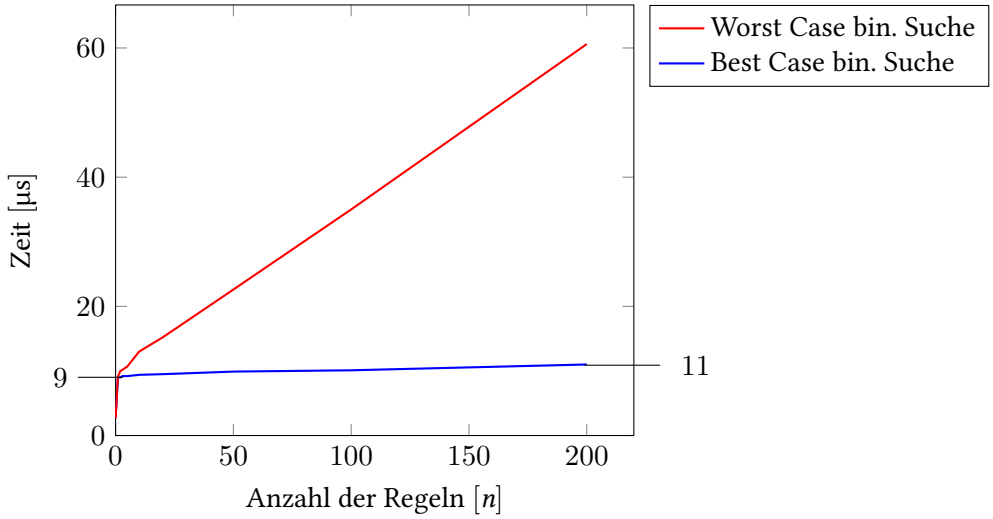



Abbildung 7.3: Dauer der binären Suche und Anwendung einer Regel in Abhängigkeit zu der Anzahl an Regeln

Wie in der Abbildung 7.3 zu sehen ist, benötigt das Suchen und Ausführen einer Regel eine Zeit von etwa 9 µs. Da bei einer einzelnen Regel keine Suchzeit anfällt, entsprechen diese 9 µs der reinen Ausführungszeit einer Regel. Im günstigsten Fall (blaue Linie) verändert sich die Laufzeit mit zunehmender Regelanzahl nur geringfügig. Ändert sich die Anzahl der Regeln von 1 auf 200 Regeln, steigt die Laufzeit von 9 µs auf 11 µs. Die blaue Linie gilt allerdings nur, wenn die untersuchten Regeln ausschließlich einen 29er prefix besitzen. Für diesen Fall hat die binäre Suche kaum eine Auswirkung auf die gesamte Laufzeit. Wenn aber unterschiedliche prefixes in der Konfiguration vorkommen, liegt die reale Laufzeit zwischen dem Worst Case (rote Linie) und dem Best Case (blaue Linie). Je mehr Regeln vorliegen, desto größer ist der Anteil der Suchzeit an der gesamten Laufzeit.

Der Worst Case in der Abbildung 7.3 wird erzeugt, indem ein CAN-Frame empfangen wird, dessen CAN-ID größer ist als alle anderen IDs in der Liste. An der höchsten Stelle bricht die binäre Suche ab und startet von dort eine lineare Suche. Vereinfacht lässt sich sagen, je niedriger die prefixes in der Konfiguration sind, desto größer wird die Laufzeit der binären Suche. Wenn der Abbruch der binären Suche dicht am gewünschten Ziel liegt, z.B. wenn eine Regel (540/28) lautet und nach der CAN-ID 541 gesucht wird, bricht die binäre Suche ein Element weiter ab und muss nur ein Element mit der linearen Suche zurück gehen. Für diesen

Fall könnte man den Code-Generator auch in Zukunft um eine Funktion erweitern, der aus Regeln mit einem hohen prefix (24-28) mehrere einzelne Regeln mit einem 29er prefix macht. So kann der Worst Case für zu erwartende CAN-Frames verringert oder komplett vermieden werden. Falls auf dem CAN-Bus Frames vorkommen, die nicht weitergeleitet werden, können diese ebenfalls zu einer sehr hohen Suchzeit führen, da sie nicht in der Regelliste vorkommen und eine lineare Suche verursachen. Um dies zu vermeiden, sollte für den produktiven Einsatz des Gateways-2017 die lineare Suche komplett deaktiviert werden. Die Messungen haben gezeigt, dass die binäre Suche optimal ist für Regeln, die genau für eine CAN-ID bestimmt sind (29er prefix). Bei Regeln mit einem anderen Prefix kann es durch die lineare Suche im Worst Case zu einem linearen Anstieg der Suchzeit kommen. Das Konzept zum Gateway-2017 wurde daher um den binären Suchbaum erweitert.

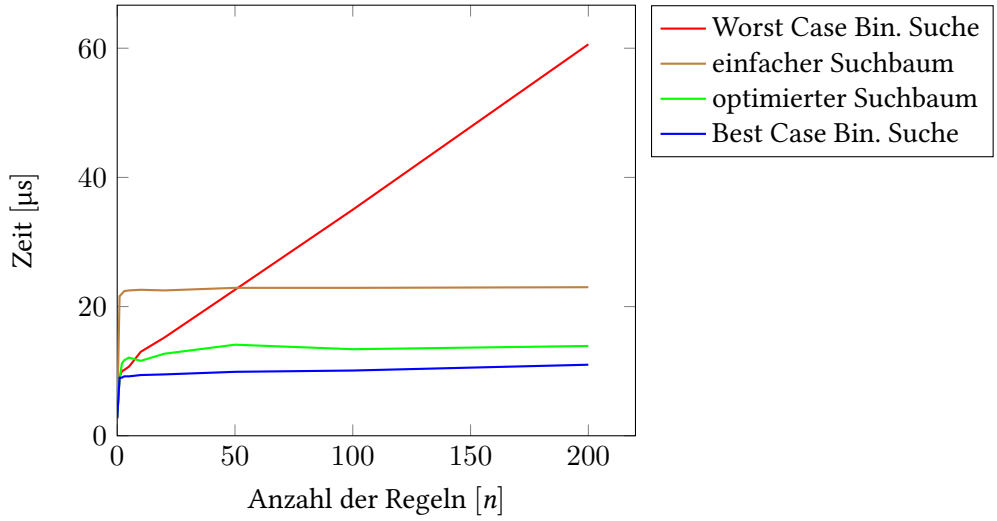



Abbildung 7.4: Dauer der binären Suche oder des binären Suchbaums und Anwendung einer Regel in Abhängigkeit zur Anzahl an Regeln

Wie anhand der Abbildung zu sehen, ist bei einer geringen Anzahl an Regeln (unter 10) die Zeit für alle Suchalgorithmen relativ gleich, etwa 10 µs. Da die binäre Suche hier selbst im Worst Case schneller als der binäre Suchbaum ist, sollte bei einer so geringen Anzahl an Regeln immer die binäre Suche verwendet werden. Da die binäre Suche im Best Case immer unter dem optimierten Baum bleibt, sollte bei Regeln, die nur 29er prefixe verwenden, dieses Suchverfahren eingesetzt werden. Beide Algorithmen zeigen durch ihre gute Laufzeit von O(log*n) im Best Case, dass sich die Laufzeit des Gateways-2017 durch die Anzahl an Regeln nur geringfügig erhöht. Der einfache Suchbaum zeigt den Worst Case des optimierten Suchbaums. Dieser liegt im konkreten Fall bei einer Suchtiefe von 30 (prefix 0-29). Der optimierte Suchbaum

wurde mit einer zufällig generierten Konfiguration getestet. Falls manuell eine Konfiguration erzeugt würde, die 30 Regeln hintereinander kettet (0/0),(0/1),(0/2)..., dann würde der Worst Case auftreten. Diese Konfiguration ist nicht realistisch, da der Empfang eines CAN-Frames mit der ID 0 die Ausführung von 30 Regeln bedeuten würde.

### Ausgeführte Regeln

Da das Gateway-2017 beim Empfang eines CAN-Frames auch die Ausführung von mehreren Regeln unterstützt, musste zusätzlich geprüft werden, wie sich die Anzahl der ausgeführten Regeln auf die Laufzeit des Gateways-2017 auswirkt:

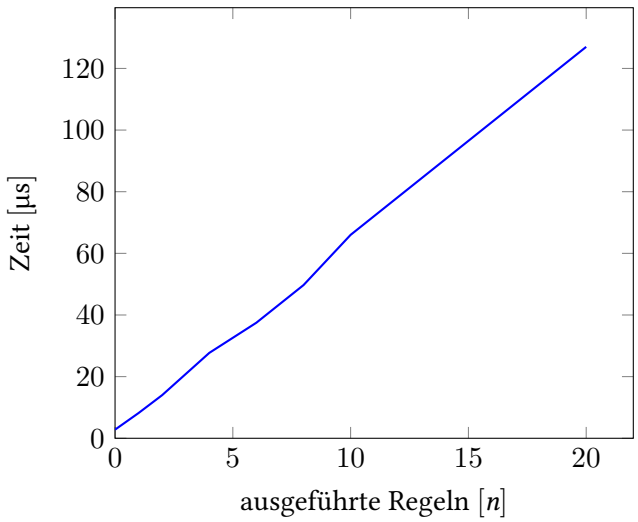



Abbildung 7.5: Auswirkung der Anzahl von ausgeführten Regeln auf die Laufzeit

Die Dauer der Bufferzuordnung steigt linear mit der Anzahl der Regeln. Dies war zu erwarten, da der CAN-Frame mit jeder Regel erneut kopiert wird, wenn es sich bei dieser Regel um eine Regel mit einer anderen Ethernet-Empfangsadresse handelt. Eine hohe Anzahl an Ethernet-Empfängern erhöht auch die Auslastung auf dem Ethernet-Backbone. Daher sollte ein CAN-Frame, der an mehrere Empfangs-Gateways gesendet werden soll, besser über eine Multicast-Adresse verschickt werden. Das würde die Anzahl der auszuführenden Regeln und die Auslastung auf dem Ethernet-Backbone verringern. Der Offset bei 0 ausgeführten Regeln entspricht der Suchzeit. Die weiteren Regeln sind durch die verkettete Liste direkt aufrufbar und müssen nicht erneut gesucht werden. Der zusätzliche Zeitbedarf entsteht durch die Ausführung der Regeln.

### 7.4.3 Laufzeit beim Senden der Ethernet-Frames

Da beim Senden eines Ethernet-Frames ein Buffer in den zu versendenden Ethernet-Frame kopiert werden muss, hängt die Laufzeit dieser Funktion linear von der Anzahl der CAN-Frames ab, die in diesem Ethernet-Frame gebündelt sind:

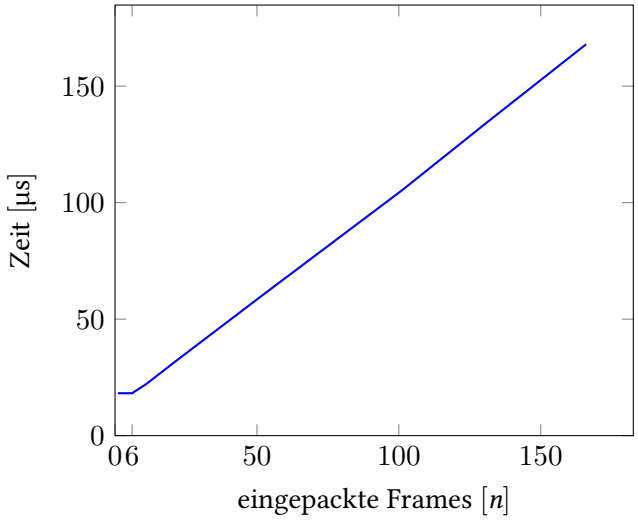



Abbildung 7.6: Zeitliche Auswirkungen in Abhängigkeit zu der Anzahl von CAN-Frames in einem Ethernet-Frame beim Senden eines Ethernet-Frames

Da ein Ethernet-Frame eine minimale Nutzlast von 46 Byte hat und die Ethernet-API teilweise alte Daten im Buffer behält, muss der Ethernet-Frame bis zur minimalen Größe aufgefüllt werden. Dies erklärt, warum die Laufzeit von 1-6 Frames identisch bleibt und erst ab 7 Frames linear ansteigt.

### 7.4.4 Laufzeit beim Empfang von Ethernet-Frames

Durch die Nachrichtenbündelung verändert sich die Laufzeit, die im Gateway-2017 benötigt wird, um alle CAN-Frames aus einem Ethernet-Frame zu entpacken.

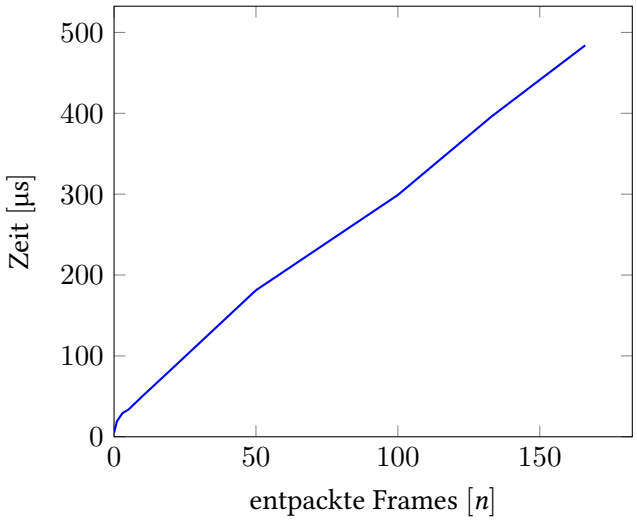



Abbildung 7.7: Zeitliche Auswirkungen in Abhängigkeit zu der Anzahl von CAN-Frames in einem Ethernet-Frame beim Empfang eines Ethernet-Frames

Wie in der Abbildung 7.7 zu sehen, geht die Anzahl der verpackten CAN-Frames annähernd linear in die Verarbeitungszeit ein. Dieses Verhalten hat Auswirkungen auf das Scheduling, da sich in Abhängigkeit von der Nachrichtenbündelung die Laufzeiten des Task verändern.

### 7.4.5 Laufzeit beim Senden von CAN-Frames

Die Funktion „Senden von CAN-Frames“ kopiert CAN-Frames in den HW-Buffer des CAN-Controllers. Diese Funktion verläuft annähernd linear.

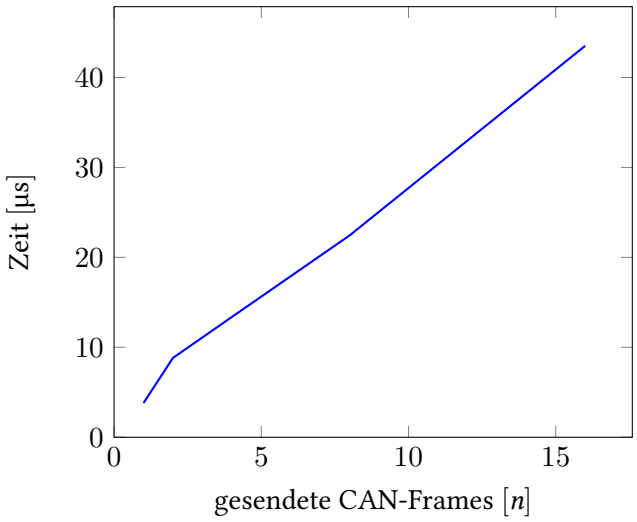



Abbildung 7.8: Messung der Laufzeit für das Senden von CAN-Frames

Bei dieser Funktion ist kein Offset vorhanden, sodass es keine Laufzeit-Vorteile hätte, sie nur aufzurufen, wenn sich mehrere CAN-Frames im Buffer befinden.

### 7.4.6 Laufzeit-Vergleich mit dem Gateway-2015

Der Bachelorarbeit von Patrick Kuncke lässt sich entnehmen, dass das Gateway-2015 beim Empfang von CAN-Frames 2 ISRs durchlaufen muss („CANRecv“ 45-52µs und „Worker ISR CAN“ 258-289µs) um einen CAN-Frame über Ethernet zu verschicken. Das ergibt im Optimalfall in Summe 303µs für die Verarbeitung.

Beim Gateway-2017 setzt sich diese Zeit aus der ISR 3,9 µs, dem Task „Zuordnung und Buffering von CAN-Frames“ 14 µs, dem Task „Sende Buffer via RTE“ 18,2 µs und dem Task „Sende Ethernet HW-Buffer“ 5 µs zusammen. In Summe ergibt das nur 41,1 µs.

Meine Messungen wurden mit dem Optimierungsflag -O1 durchgeführt. Ob auch in vorangegangen Messungen ein Opmtimierungsflag verwendet wurde, konnte den Abschlussarbeiten nicht entnommen werden.

Bei der Kommunikation in entgegengesetzter Richtung von Ethernet nach CAN durchläuft das Gateway-2015 die ISR „EthRecv0“ 5 µs, „CallBackTask“ 30 µs und „Worker ISR Eth“ 180-190 µs.

Das Gateway-2017 durchläuft den Task „Empfangen und Entpacken von RTE-Frames“ 19,1 µs und „Senden von CAN-Frames“ 3,8 µs.

Im direkten Vergleich der Laufzeiten zeigte sich das Gateway-2017 also überlegen.

| Gateway | CAN nach Ethernet | | Ethernet nach CAN | |
|---|---|---|---|---|
| Gateway-2017 (Wendt) | 41.1 | µs | 22.9 | µs |
| Gateway-2015 (Depke) | 303 | µs | 215 | µs |

Tabelle 7.4: Laufzeit-Vergleich Depke und Wendt

Die Laufzeit der Funktionen, die an der Übertragung von CAN nach Ethernet beteiligt sind, ist klein genug um eine volle Auslastung auch bei 1 MBit auf dem CAN-Bus zu verarbeiten, da ein minimaler CAN-Frame 47 Bit groß ist und 21277 mal pro Sekunde übertragen werden kann. Es ergibt sich eine Auslastung der Rechenzeit von 87,5%. Bei der Verbindung Ethernet nach CAN liegt die Auslastung bei 48,7%.

### 7.4.7 Auswertung der Laufzeit

Insgesamt ist zu erkennen, dass die Anzahl der Regeln mit dem jeweils optimalen Suchalgorithmus kaum Auswirkungen auf die Laufzeit hat. Auch die ebenfalls untersuchten Funktionen zum Senden, Empfangen und Verarbeiten von Ethernet- und CAN-Frames haben eine Laufzeit, die auf einen Frame bezogen, relativ konstant ist, was die Berechnung der Auslastung bei maximalem Durchsatz wie im Abschnitt 7.4.6 einfach möglich macht. Wenn auf die Nachrichtenbündelung verzichtet wird, könnte die Ausführungszeit von der Bufferzuordnung und dem Senden von Ethernet-Frames verringert werden, da der CAN-Frame nur noch in den Ethernet HW-Buffer kopiert werden müsste. Außerdem braucht der Ethernet-Frame nicht auf seine minimale Größe aufgefüllt werden, da immer ein CAN-Frame enthalten ist. Hier kann auch mit Nachrichtenbündelung eine weitere Optimierung erfolgen. Steht am Anfang des Ethernet-Frames die Anzahl enthaltener CAN-Frames, muss der Ethernet-Frame nicht aufgefüllt werden. Wenn beim Kompilieren der Optimierungsflag -O2 verwendet wird, kann die Laufzeit vermutlich noch einmal verringert werden.

## 7.5 Speicherbedarf

Der generierte Code besitzt eine variable Zeilenanzahl an Quellcode. Dies hängt von der Anzahl der Regeln ab. Zwischen der binären Suche und dem binären Suchbaum besteht ein großer Unterschied. Im folgenden Diagramm wird daher die Anzahl der Quellcode-Zeilen zwischen den verschiedenen Verfahren verglichen.

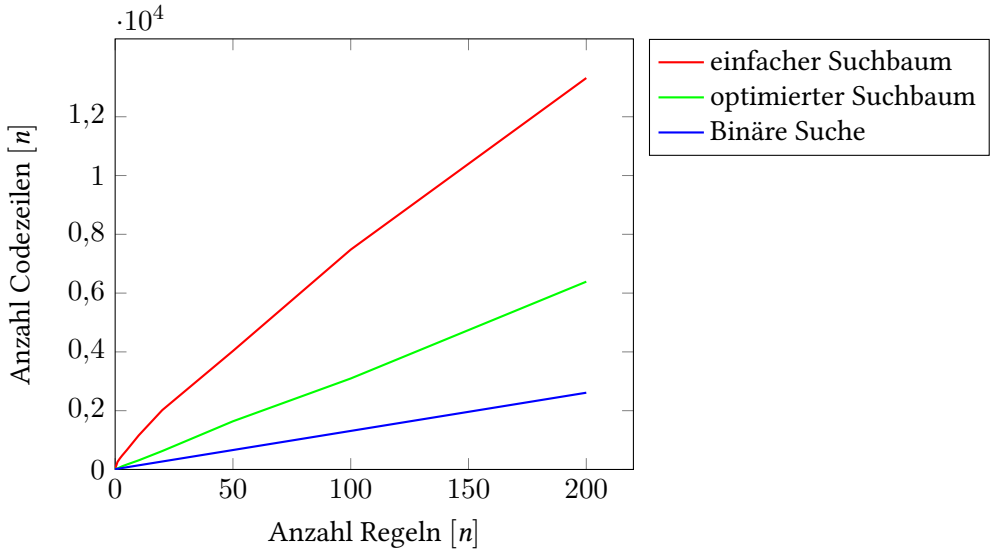



Abbildung 7.9: Die Anzahl der Source-Code-Zeilen in Abhängigkeit zum eingesetzten Such-Verfahren

Im Bild erkennt man, dass die binäre Suche den geringsten Speicherbedarf hat, da sie keine Dummy-Knoten benötigt. Der einfache Suchbaum benötigt deutlich mehr Code-Zeilen, da alle Knoten geladen werden. Der optimierte Baum entfernt nicht benötigte Dummy-Regeln und hat einen deutlich geringeren Speicherbedarf als die nicht optimierte Form.

Bei der Testdurchführung mit 500 Regeln reichte der Speicher des µCs nur noch für den optimierten Baum und die binäre Suche aus.

# 8 Konfiguration Scheduler

Da nach der Auswertungsphase die Laufzeiten der einzelnen Tasks bekannt sind, kann ein passendes Schedule erstellt werden. Um die Latenz der Schedules zu vergleichen, wird für die nachfolgenden Konfigurationen auf den Task „Sync Window“ verzichtet. Dieses „Sync Window“ würde 200 µs benötigen und einmal pro Scheduler-Periode aufgerufen werden müssen. In der Folge würden Ausreißer in den Messungen um eben diese 200 µs auftreten. Im Laboraufbau sind nur zwei Gateways-2017 mit dem Ethernet-Switch verbunden und Ethernet ist Full-Duplex fähig, sodass es trotz abgeschaltetem „Sync Window“ zu keiner zeitlichen Verzögerung durch eine Belegung des physikalischen Mediums kommt. Der Switch im Prototypenfahrzeug ist so konfiguriert, dass nur die Ethernet-Frames, die für die Kommunikation zwischen den Gateways-2017 erforderlich sind, weitergeleitet werden. So wird auch hier verhindert, dass zusätzliche Latenzen durch ein belegtes physikalisches Medium entstehen.

Es folgen drei Varianten, wie der Scheduler für die beschriebenen Tasks konfiguriert werden kann:

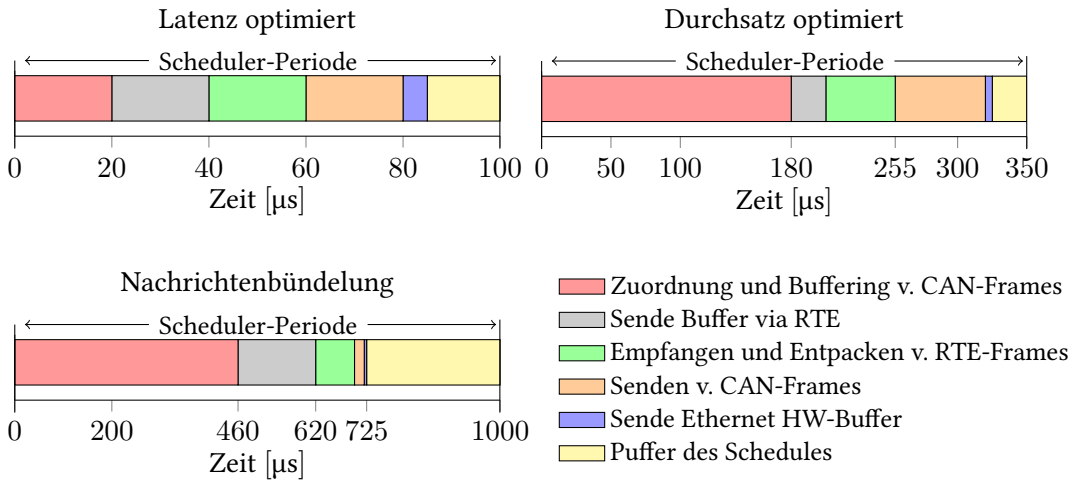



Abbildung 8.1: Drei Schedules im Vergleich graphisch dargestellt

Um diese Schedules zu vergleichen, soll berechnet werden, wie sich das jeweilige Schedule auf die Latenz und die Auslastung auswirkt. Hierzu muss zunächst die Laufzeit der einzelnen

Funktionen addiert werden. Die exakte Latenz wird etwas über der Laufzeit der einzelnen Funktionen liegen, da Kontext-Wechsel und die Laufzeit des Schedulers noch nicht berücksichtigt sind.

### Latenz

Die beiden Latenzen (Ethernet nach CAN und CAN nach Ethernet) wurden folgendermaßen berechnet:

$$\begin{aligned} MinmaleLatenzRTENachCAN &= EmpfangenUndEntpackenVonRTEFrames \\ &\quad + SendenVonCanFrames \\ MinmaleLatenzCANNachRTE &= ZuordnungUndBufferingVonCANFrames \\ &\quad + SendeBufferViaRTE \\ &\quad + SendeEthernetHWBuffer \\ &\quad + ISR \\ maximaleLatenz &= minimaleLatenz + SchedulingPeriode \end{aligned} \tag{8.1}$$

Berechnung der minimalen und maximalen Latenz

### Auslastung

Die Auslastung hängt von der gewünschten Bandbreite und der Größe der einzelnen CAN-Frames ab und wird wie folgt berechnet:

$$Auslastung[\%] = \frac{CanFramesProSekunde * BitsProFrame}{Bandbreite[BitsProSekunde]} \tag{8.2}$$

Berechnung der CAN-Bus Auslastung

Die „CanFramesProSekunde“, die das Gateway-2017 verarbeiten kann, hängen davon ab, wie oft die unterschiedlichen Funktionen aufgerufen werden und wie lange diese ausgeführt werden.

### Vergleich

Bei der Schedule Variante „Latenz optimiert“ wird eine Periode von 100 µs verwendet um eine möglichst gute Latenz zu erreichen. Die maximale Auslastung bei nur minimalen Frames mit einer Bandbreite von 1 MBit/s kann nur erreicht werden, wenn beide Richtungen aktiv (Ethernet nach CAN/CAN nach Ethernet) sind. Bei maximalen CAN Frames ist eine volle

Auslastung aber möglich. Pro Schedule Periode kann 1 CAN-Frame in einen Ethernet-Frame gebündelt werden und 1 CAN-Frame aus einem Ethernet-Frame entpackt werden. Dieser Schedule soll für die Tests im Prototypenfahrzeug verwendet werden, da er die beste Latenz liefert. Die CAN-Busse im Fahrzeug werden nur mit 500 KBit/s Bandbreite betrieben, sodass immer eine volle Auslastung möglich ist.

Die Scheduling Variante „Nachrichtenbündelung“ zeigt Probleme, die durch die Nachrichtenbündelung für das Scheduling entstehen. Die Funktion „Zuordnung und Buffering von CAN-Frames“ sortiert 1 bis n CAN-Frames in die Buffer ein. Da mehrere CAN-Frames zugeordnet werden müssen, variiert die Laufzeit der Funktion um den Faktor der Anzahl der CAN-Frames.

Das gleiche Verhalten tritt bei den Funktionen „Sende Buffer via RTE“ und „Empfangen und Entpacken von RTE-Frames“ auf, da deren Laufzeit von der Größe der Ethernet-Frames abhängt. Die Funktion „Senden von CAN-Frames“ kann maximal 16 CAN-Frames in den HW-Buffer des CAN-Controllers kopieren und müsste daher in einem Scheduler Umlauf mehrfach aufgerufen werden, solange bis alle CAN-Frames, die von der Funktion „Empfangen und Entpacken von CAN-Frames“ zwischengespeichert wurden, verschickt sind. Wenn das Scheduling auf das Empfangen von Ethernet-Frames mit maximal 166 verpackten CAN-Frames konfiguriert ist, wird das Versenden des letzen CAN-Frames in diesem Ethernet-Frame um die Anzahl der Vorgänger verzögert (7,25 ms bei minimalen CAN-Frames und 1 MBit/s).

Um einen Kompromiss zwischen diesen Schedules zu finden, wurde das Schedule „Durchsatz optimiert“ erzeugt. Hierzu wurde zunächst berechnet, wie lange die Übertragung von 8 minimalen CAN-Frames auf dem CAN-Bus dauert. Dies dauert 352 µs. Daher ist es ausreichend, wenn der Schedule mit einer Periode von 350 µ abgearbeitet wird. Als Erstes startet die Funktion „Zuordnung und Buffering von CAN-Frames“. Diese benötigt 20 µs pro CAN-Frame. Danach startet die Funktion „Sende Buffer via RTE“ nach 180 µs (8 * 20 + 20 Sicherheit). Da diese bei 11 Frames 22,2 µs dauert, startet die Funktion „Empfangen und Entpacken von RTE-Frames“ nach 205 µs (22,2+180+Sicherheit). Diese Funktion benötigt 50 µs. Die Funktion „Senden von CAN-Frames“ startet daher bei 255 µs und benötigt 8 µs pro Frame. Bei 320 µs (255+64+Sicherheit) startet die Funktion „Sende Ethernet Buffer“, die für das Senden von einem Frame 4 µs benötigt. Die restlichen 30 µs bleiben für die ISR, die pro Frame 3,9 µs benötigt. Bei diesem Schedule ist die Latenz zwar höher als beim „Latenz optimierten“, aber der CAN-Bus kann in beide Richtungen bei 1 MBit/s Bandbreite voll ausgelastet werden.

Der Scheduler verursacht Probleme bei unterschiedlichen Lasten. Ein Schedule wird so ausgelegt, dass auch bei einer maximalen Ausführungszeit das Schedule einzuhalten ist. Da sich die Laufzeit der Funktionen in Abhängigkeit zur Frames Anzahl verändert, lässt sich nur ein

Schedule für eine feste Anzahl an zu verarbeitenden Frames festlegen. Dadurch bedingt ist das Schedule auch Full-Duplex ausgelegt, obwohl der CAN-Bus nicht Full-Duplex fähig ist.

Abschließend lässt sich daher zusammengefasst sagen, dass das „Latenz optimierte“ Schedule bei 500 KBit/s Bandbreite alle Frames verarbeiten kann und die bessere Latenz liefern wird. Der „Durchsatz optimierte“ Schedule kann auch einen CAN-Bus, der 1 MBit/s Bandbreite verwendet, verarbeiten. Er bietet aber eine höhere Latenz.

# 9 Auswertung Latenz

Im folgenden Kapitel wird die komplette Übertragungs-Latenz des Gateways-2017 und die Laufzeit des Schedulers ermittelt. Es wird das Gateway-2017 unter realen Bedingungen im Prototypenfahrzeug getestet und abschließend wird ein Versuch ohne Scheduling durchgeführt.

## 9.1 Latenz Gateway im Labor

In diesem Abschnitt wird die Latenz des Gateway-2017 mit 3 Messungen ermittelt.

### CAN nach Ethernet

Zunächst wurde die minimale Latenz berechnet. Sie ergibt sich aus der Laufzeit der ISR 3,9 µs, der Funktion „Zuordnung und Buffering von CAN-Frames“ 8 µs, und der Funktion „Sende Buffer via RTE“ 18,2 µs. Summiert liegt die optimale Latenz bei 30,1 µs, addiert mit der Scheduling Periode ergibt sich die maximale Latenz von 130,1 µs. Um die Latenz zu ermitteln, die ein CAN-Frame im Gateway-2017 beim Schedule mit 100 µs und ohne Nachrichtenbündelung vom Empfang bis zum Versenden als Ethernet-Frame benötigt, wird ein GPIO am Anfang der ISR, die den CAN-Frame einsortiert, auf „HIGH“ gesetzt und am Ende der „sendRTE“ Funktion, die den Ethernet-Frame verschickt, auf „LOW“ gesetzt. Die im Diagramm dargestellte Zeit ist die Zeit, die von der „HIGH“ bis zur „LOW“ Flanke vergeht.

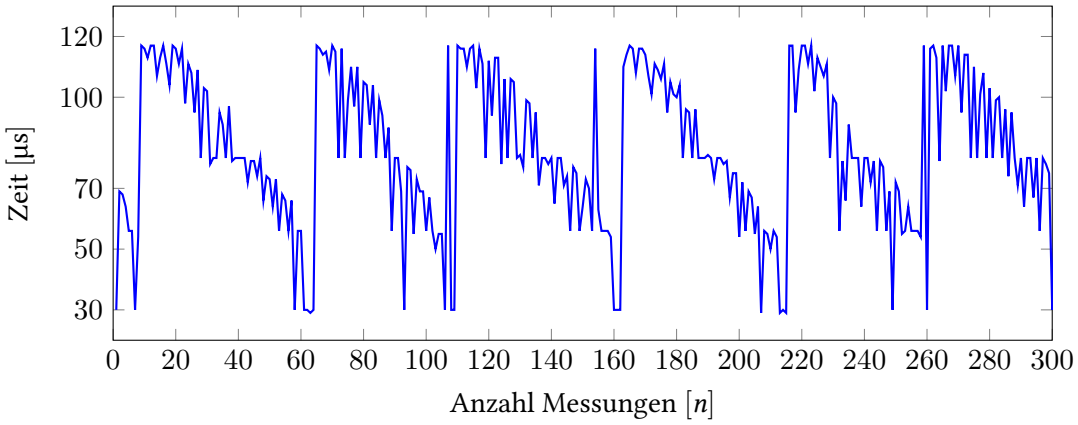



Abbildung 9.1: Messung der Latenz von CAN nach Ethernet mit Scheduler

Die CAN-Frames wurden mit einer Periode von einer ms an das Gateway-2017 geschickt. Der Schedule wird in dieser Zeit 10 mal durchlaufen. Im Optimalfall, wenn beide Uhren (CAN-Ethernet-Debugger und µC) synchron laufen, müsste immer eine identische Ausführungszeit gemessen werden. Um einen Drift der µC Uhr zu bestimmen, wurde in einer Funktion, die im Schedule ausgeführt wurde, ein GPIO ein- und wieder ausgeschaltet. Da diese Funktion einmal im Schedule aufgerufen wird, müsste der Abstand zwischen diesen Spitzen einer Scheduler-Periode entsprechen. Dieser Abstand liegt bei 100 µs, gemessen wurden aber 100,8 -101,2 µs. Diese leichte Abweichung nach oben entsteht wahrscheinlich durch einen überladenen Schedule. Daraus erklärt sich das Sägezahnmuster in der Messung. Zunächst nähert sich der Versendezeitpunkt dem Optimalfall an. Ist dieser im Schedule überschritten, schlägt die Latenz auf den Worst Case um. Ein Vorteil, der sich durch dieses Abdriften der Uhren ergibt, ist, dass die zuvor berechneten Werte für den Worst Case und den Best Case mit nur einer Messung verifiziert werden konnten.

### Ethernet nach CAN

Zusätzlich sollte auch die Latenz vermessen werden, die das Gateway-2017 benötigt, um einen CAN-Frame zu entpacken und zu versenden. Hierzu wurde ein GPIO am Anfang der „Empfangen und Entpacken von RTE-Frames“ 19,1 µs, auf HIGH gesetzt. Am Ende der „Sende CAN-Frames“ 3,8 µs, wurde der GPIO wieder zurückgesetzt. Die Best Case Laufzeit dieser beiden Funktionen ergibt 22,9 µs.

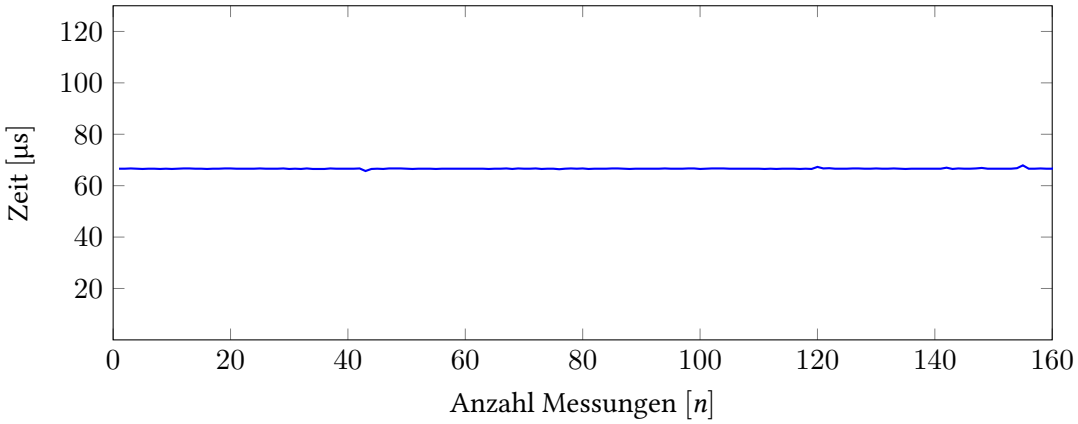



Abbildung 9.2: Messung der Latenz von Ethernet nach CAN mit Scheduler

Weil es auf der Ethernet-Empfangsseite keine ISR gibt, zeigt sich in der Messung kein Sägezahnmuster. Das Ergebnis liegt konstant bei 66,5 µs. Da bei der Messung zwischen den beiden Funktionen im Schedule ein weiterer Task liegt, verzögert sich die Messung um diesen Task. Hier zeigt sich ein Problem, das durch das Scheduling entsteht. Selbst wenn kein Frame von CAN nach Ethernet geht, wird im Schedule die volle Zeit für die Funktion reserviert.

### Latenz CAN nach CAN

Der folgende Abschnitt zeigt alle Latenzen, die bei der Übertragung von CAN nach CAN über den Ethernet-Backbone auftreten. Hierbei handelt es sich um folgende 5 Komponenten:

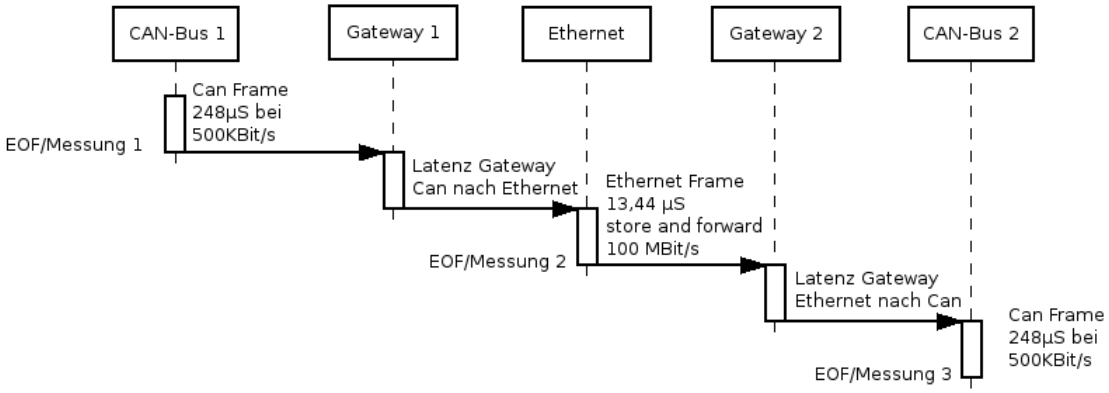



Abbildung 9.3: Zusammensetzung der Latenz bei der Übertragung eines CAN-Frames über den Ethernet-Backbone zu einem anderen CAN Bus

Die Dauer der CAN-Arbitrierung hängt von der Größe und dem Inhalt eines CAN-Frames ab. Ein maximaler CAN-Frame mit 153 Bit (inklusive bit stuffing) belegt den CAN-Bus bei einer Bandbreite von 500Kbit/s für die Dauer von 306 µs. Diese Zeitdauer errechnet sich mit folgender Formel:

$$Uebertragungsdauer[s] = \frac{1}{Bandbreite/AnzahlBits} \tag{9.1}$$

Übertragungsdauer eines Frames

Für die Übertragungsdauer des Ethernet-Frames gilt sinngemäß die gleiche Formel. Da bei den Latenz-Messungen keine Nachrichtenbündelung stattfindet und nur minimale Ethernet-Frames mit 84 Bytes übertragen werden, muss nur eine Übertragungsdauer berechnet werden. Es ergibt sich eine Übertragungszeit von 6,72µs. Da der Ethernet-Switch store and forward verwendet, verdoppelt sich die Verzögerungszeit auf 13,44 µs. Mit dem CAN-Ethernet-Debugger lässt sich die Dauer der Übertragungszeit festellen. Es werden drei Messungen am Ende der EoF-Felder der CAN-Frames und dem Ethernet-Frame durchgeführt und mit einem Zeitstempel versehen. Die Latenz des Gateway-2017 CAN nach Ethernet ergibt sich aus der Differenz zwischen der ersten und der zweiten Messung, abzüglich der Übertragungsdauer des Ethernet-Frames. Die Differenz zwischen der zweiten und dritten Messung, abzüglich der CAN-Arbitrierungszeit, ist

die Latenz des Gateways-2017 von Ethernet nach CAN. Diese Arbitrierungszeit liegt statisch bei 248 µs, weil der CAN-Bus frei von anderen CAN-Frames ist.

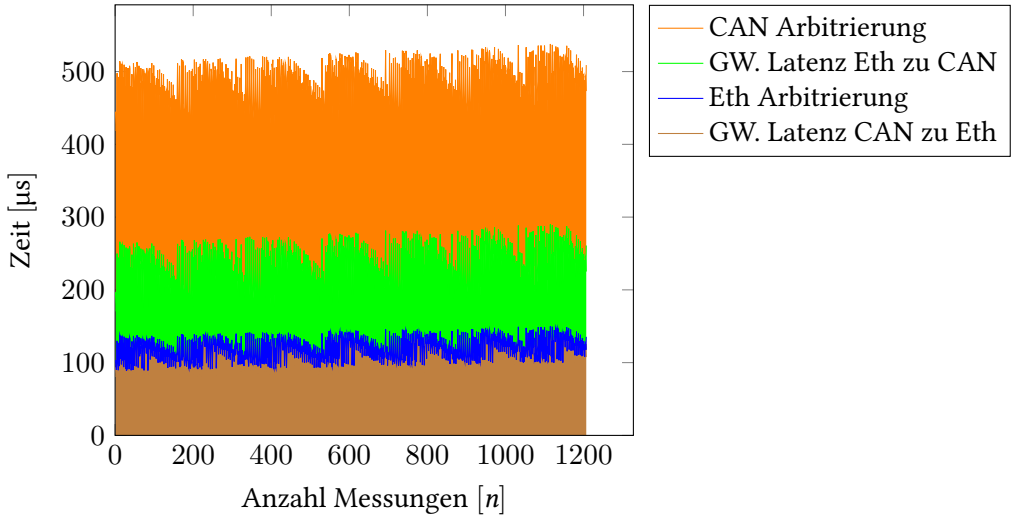



Abbildung 9.4: Messung der Latenz von CAN nach CAN im Labor

In der Messung ist nach wie vor ein leichtes Sägezahnmuster von CAN nach Ethernet zu erkennen. Dieses Muster zeigt sich auch in der gesamten Messung nach der Übertragung über den Ethernet-Backbone. Die Latenz (grün, blau und braun) der Gateways-2017 gleicht der Arbitrierungszeit (orange) eines CAN-Frames. Durch die Verwendung des Schedulers liegt auch die gesamte Latenz deutlich über den berechneten Laufzeiten im Gateway-2017 aus der Tabelle 7.4. Beim Gateway-2015 ist die Latenz des Gateways-2015 gleich der Summe der Laufzeiten der einzelnen Funktionen, da alles in verschiedenen ISRs ausgeführt wird. Wie die Messungen von Patrick Kuncke belegen, kann es durch lange Ausführungszeiten zu Warteschlangen im Gateway-2015 kommen und dementsprechend auch zu Ausreißern in der Latenz (bis zu 10 ms).

Beim Gateway-2017 wird nur das Buffern der CAN-Frames von einer ISR ausgeführt. Alle anderen Tasks werden vom Scheduler aufgerufen. Daher ist bei einer richtigen Konfiguration des Schedulers garantiert, dass es zu keiner Warteschlange im Gateway-2017 kommt. Das Schedule „Latenz optimiert“, das für diese Messungen ausgewählt wurde, ist so ausgelegt, dass bei 500 KBit/s Bandbreite eine volle Auslastung möglich ist und keine Warteschlagen entstehen.

## 9.2 Laufzeit Scheduler im Labor

Da der Aufruf und der Rücksprung in eine Funktion bei dem µC etwa 1 µs beträgt, sollte zusätzlich ermittelt werden, wie lange der Scheduler braucht, um nach Beendigung einer Task den nächsten Task zu starten. Hierzu wurde zunächst ein Task, der ca. 5 µs benötigt, in den Schedule eingetragen. 6 µs nach Beginn der ersten Task wurde der nächste Task im Schedule eingetragen. Um zu prüfen, ob der zweite Task wirklich 6 µs nach der ersten startet, wurden in beiden Tasks GPIOs am Anfang auf HIGH und am Ende auf LOW gesetzt. Zusätzlich wurde auch die Funktion im Scheduler vermessen, die die Tasks aufruft:

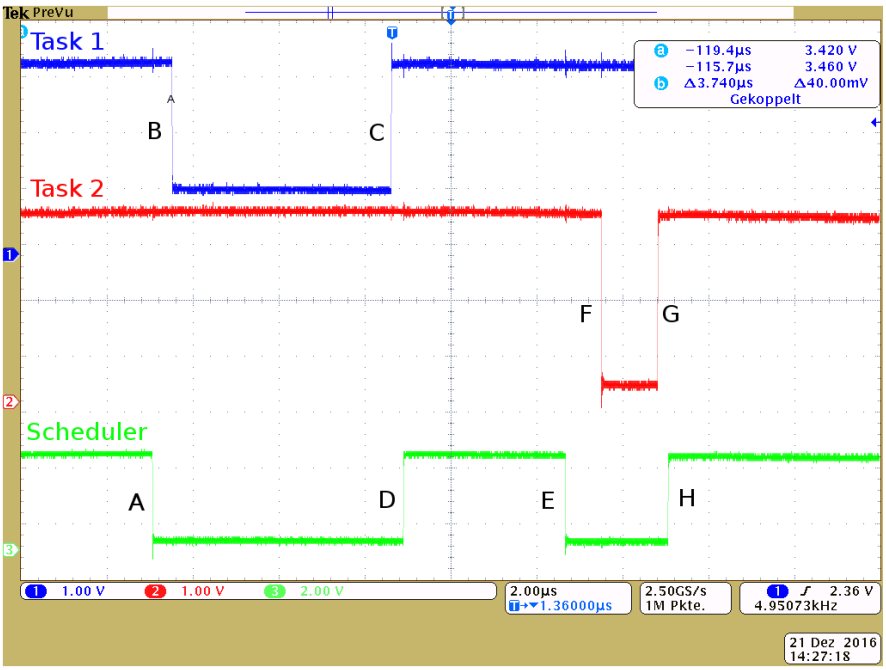



Abbildung 9.5: Messung der Scheduler-Laufzeit

Es startet zunächst der Scheduler (Flanke A) den Task 1 (Flanke B).

Nachdem der Task 1 beendet ist (Flanke C), geht der Programmfluss zurück in die Scheduler-Funktion. Es dauert 300 ns bis (Flanke D).

Nach weiteren 3,7 µs wird der Scheduler wieder aktiv (Flanke E) und startet nach 880 ns den 2. Task (Flanke F).

Die unterschiedlichen Zeitabstände zwischen dem Starten des 1. Tasks (Flanke A bis Flanke B) und des 2. Tasks (Flanke E bis Flanke F) erklären sich durch Cache Misses des Prozessors oder eine unterschiedliche Anzahl an verwendeten Registern, die vor dem Kontext-Wechsel auf den Stack gesichert werden müssen.

Zusätzlich gibt es noch die ISR „FIQ_handler“ und die Funktion „scheduler_clearEventTrigger“, die zwischen zwei Tasks ausgeführt werden. Diese wurden zusätzlich vermessen und ergaben die 3,7 µs zwischen Flanke D und Flanke E.

So lässt sich abschließend sagen, dass der Scheduler bei überwiegend kleinen Tasks einen großen Anteil der Rechenzeit benötigt. Der Abstand zwischen Task 1 (Flanke C) und Task 2 (Flanke F) liegt bei insgesamt 4,83 µs. Beim „Latenz optimierten“ Schedule werden bereits 20% der Laufzeit für den Scheduler benötigt.

## 9.3 Validierung im Prototypenfahrzeug

Nachdem die Tests und die Laufzeitmessungen erfolgreich waren, wurde die Software im Prototypenfahrzeug der CoRE Gruppe eingesetzt und vermessen.

Dieses Fahrzeug hat ein zentrales CAN-Gateway (VW-Gateway), das die CAN-Busse I, E und K miteinander verbindet. Das VW-Gateway prüft, ob erforderliche Frames eingehen und meldet Fehler, falls dies nicht der Fall sein sollte. Daher muss das VW-Gateway weiterhin alle Frames erhalten. Frames, die vom VW-Gateway kommen, dürfen nicht auf den CAN-Bus gehen, da die Nachrichten sonst doppelt auf dem CAN-Bus vorhanden wären. Um dies zu verhindern, werden bidirektionale Filter eingesetzt, die bestimmte Frames anhand ihrer CAN-IDs ausfiltern (Kuncke, 2016).

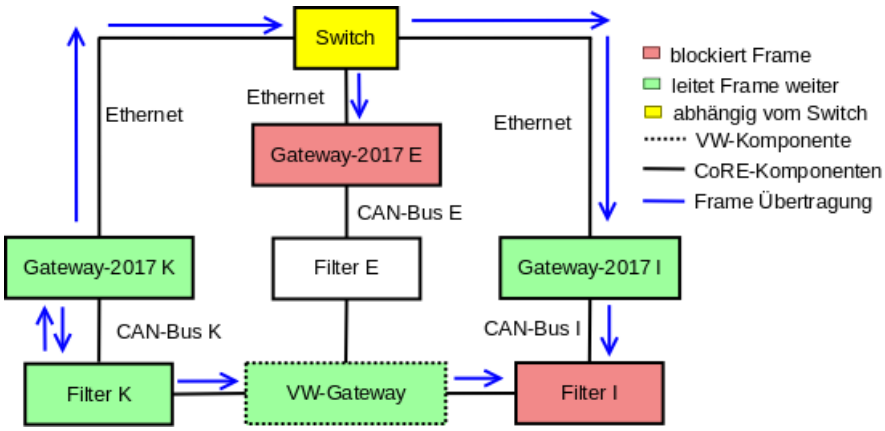



Abbildung 9.6: Testaufbau im Prototypenfahrzeug und schematische Darstellung einer Kommunikation von CAN Bus K nach CAN Bus I

In der Abbildung 9.6 wird die Übertragung eines CAN-Frames dargestellt, der vom CAN-Bus K zum CAN-Bus I übertragen werden soll. Der Frame geht zwei Wege: einerseits über den Filter K, der den Frame zum VW-Gateway durchlässt. Das VW-Gateway sendet den Frame

zum Filter I, der den Frame, entsprechend der CAN-ID, nicht weiterleitet. Andererseits nimmt das Gateway-2017 K den Frame entgegen und verpackt diesen in einen Ethernet-Frame, da sich die CAN-ID des Frames im Regelwerk des Gateways-2017 K befindet. Der Ethernet-Frame wird vom Gateway-2017 K verschickt. Der Switch sendet ihn weiter an das Gateway-2017 I, falls sich dessen MAC-Adresse in der MAC-Tabelle des Switches befindet. Sollte dies nicht der Fall sein, wird die Nachricht an Gateway-2017 I und Gateway-2017 E verschickt. Da das Gateway-2017 E keine Frames an diese MAC-Adresse erwartet, wird der Frame dort verworfen. Gateway-2017 I empfängt den Ethernet-Frame, entpackt den CAN-Frame und verschickt diesen auf dem CAN-Bus I. Ebenso verhält es sich mit den CAN-Frames von E nach I, E nach K , I nach E, I nach K und K nach E.

### 9.3.1 Testverfahren

Das Gateway-2017 wird über die Fahrzeug-Diagnose und durch die Auswertung der Traces getestet.

#### Fahrzeug-Diagnose

Das Auto führt beim Starten der Zündung eine Diagnose der Steuergeräte an den CAN-Bussen durch und zeigt auf der Anzeige des Infotainment-Systems an, welche Fahrerassistenzsysteme nicht korrekt erkannt werden.

Um sicherzustellen, dass der Testaufbau ordnungsgemäß funktioniert, wurden zunächst nur die Filter programmiert und angeschlossen. Auf dem Infotainment-System und im Tacho des Prototypenfahrzeugs erfolgten Fehlermeldungen, z.B. Line Assistant, Adaptive Cruise Control (ACC) und Kurvenlicht. Das war zu erwarten und zeigte zunächst nur an, dass die Filter einen Teil der Frames blockieren. Wenn das Gateway-2017 anschließend im Fahrzeug installiert wird, sollten diese Frames übertragen werden und keine Fehlermeldungen mehr auftreten.

Es trat jedoch weiterhin ein Fehler beim Assistenzsystem ACC auf. Um die CAN-IDs zu finden, die diesen Fehler verursacht haben, wurden Teilmengen an IDs von einer minimalen Konfiguration bis zur gewünschten Konfiguration gebildet und einzeln getestet.

Nachdem die CAN-IDs 0x120b und 0x17331910 aus der Konfiguration entfernt wurden, erfolgte keinerlei Fehleranzeige mehr. Diese aufwändige Fehlersuche war nötig, da beim Prototypenfahrzeug die Zuordnung von CAN-Botschaften zu den Funktionen (Steuergeräte) nicht bekannt ist.

#### Auswertungen der Traces

Um zu prüfen, ob das Gateway-2017 alle CAN-Frames ordnungsgemäß überträgt, wurde ein Skript einwickelt, das einen Trace automatisch überprüft, der die Kommunikation zwischen zwei unterschielichen CAN-Bussen enthält. Als Parameter benötigt das Skript alle CAN-IDs, die übertragen werden sollen. Ein Skript für die Auswertung ist erforderlich, da ein Trace von 30 Sekunden bereits beim Starten der Zündung und des Motor zwischen 2-10 MB groß ist.

Beim Überprüfen der Traces wird vom Skript festgestellt, ob alle gesendeten CAN-Frames mit den zuvor angegebenen IDs auf beiden Bussen identisch sind. Hier trat ein Fehler auf: Sobald ein zweiter Teilnehmer zeitgleich einen CAN-Frame mit einer höheren Priorität versenden wollte, scheiterte die Übermittlung und wurde nicht erneut initiiert.

Die CAN-API des µCs bietet die Möglichkeit, dass bei einem Fehlschlag des Sendevorgangs eine erneute Sendung des CAN-Frames initiiert wird. Nachdem diese Funktion aktiviert wurde, prüfte das Skript alle Traces und meldete keine Fehler mehr.

### 9.3.2 Latenzen unter realen Bedingungen

Zur Vermessung der Latenzen wurden CAN-Frames mit der CAN-ID 0x324 und einem Datenfeld vermessen, das immer 8 Byte Daten groß war. Diese CAN-Frames sendet das Prototypenfahrzeug periodisch. Das Gateway-2017 übermittelt sie von CAN-Bus E nach CAN-Bus K.

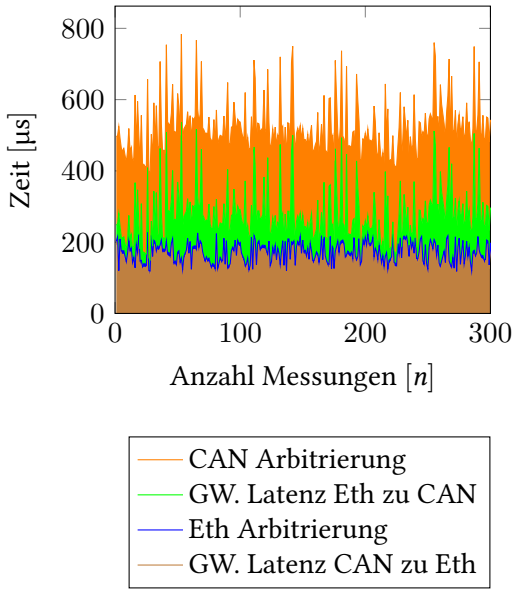



Abbildung 9.7: Messung Gateway-2017 im Prototypenfahrzeug

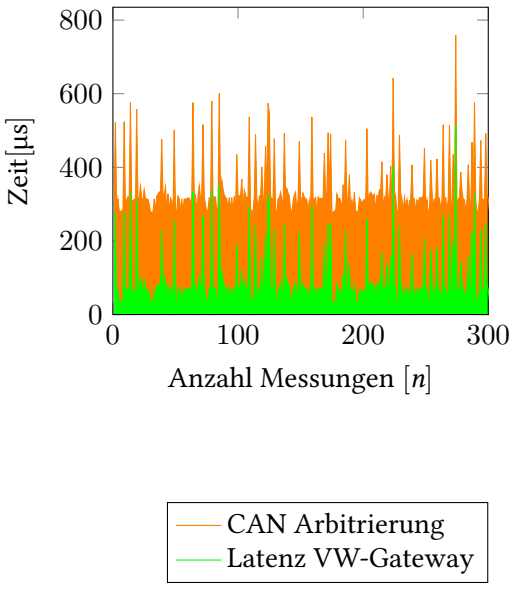



Abbildung 9.8: Messung VW-Gateway im Prototypenfahrzeug

In der Abbildung 9.7 sind die Messungen des Gateway-2017 erfasst, wie sie bereits im Laboraufbau durchgeführt und beschrieben wurden (Abbildung 9.4). Das zweite Diagramm (Abbildung 9.8) zeigt die Messergebnisse des VW-Gateways. Beim VW-Gateway gibt es nur zwei Messpunkte, jeweils am Ende des CAN-Frames. Wenn die Arbitrierung vom zweiten Messpunkt abgezogen wird, ergibt sich die Latenz des VW-Gateways. Die Arbitrierungszeit ist nicht mehr statisch, da sich bei der Messung des VW-Gateways und des Gateway-2017 unter realen Bedingung auch andere Teilnehmer auf dem CAN-Bus befinden. In der Messung des VW-Gateway und in der Messung „GW. Latenz Eth. zu CAN“ ist zusätzlich die Zeit enthalten, die die Gateways warten müssen bis der CAN-Bus frei wird.
Im Vergleich zu den Messungen im Labor (Abbildung 9.4) sind unter realen Bedingungen (Abbildung 9.7) die Ausreißer in den Messungen deutlich sichtbar. Das lässt sich dadurch erklären, dass hier mehrere CAN-Teilnehmer gleichzeitig auf den CAN-Bus schreiben wollen, und sich CAN-Frames so erst verzögert auf dem CAN-Bus zeigen. Das wird auch durch die Messungen des VW-Gateways bestätigt. Auch diese zeigen einzelne Ausreißer.

Dort, wo keine Ausreißer auftreten, liegt die Latenz des Gateways-2017 unter realen Bedingungen auf dem gleichen Niveau (300 µs) wie unter Laborbedingungen. Es ist zu erkennen, dass die grüne Linie beim VW-Gateway - Ausreißer ausgenommen - konstant unter 100 µs

liegt, während das Gateway-2017 (grüne Linie) etwa 300 µs benötigt und damit eine signifikant höhere Latenz liefert.

Es folgen die zusammengefassten Messergebnisse der drei untersuchten Gateways. Die Messungen zum Gateway-2015 sind der Bachelorarbeit von Patrick Kuncke entnommen:

| Gateway | min | max | avg |
|---|---|---|---|
| Gateway-2015 (Depke) | 820 µs | 32966 µs | 1439 µs |
| VW-Gateway | 275 µs | 643 µs | 338 µs |
| Gateway-2017 (Wendt) | 395 µs | 784 µs | 521 µs |

Tabelle 9.1: Vergleich der Min/Max/Avg Werte zwischen den drei Gateways

Hier zeigt sich, dass der Durchschnittswert (AVG) vom Gateway-2017 um 918 µs kleiner ist als der Wert vom Gateway-2015 (1439 µs). Das Maximum (32966 µs) des Gateway-2015 erklärt sich durch Warteschlangen, die dort bei hoher Last entstehen. Das VW-Gateway ist beim minimalen Wert 31% schneller und beim Durchschnittswert (avg) 36% schneller als das Gateway-2017. Auch bei den maximalen Werten ist das VW-Gateway am schnellsten. Hierbei ist zu berücksichtigen, dass bei dem Gateway-2015 und dem Gateway-2017 die Übertragung über den Ethernet-Backbone erfolgt und daher von zwei Gateways verarbeitet werden muss. Außerdem erkennt man anhand dieser Messergebnisse, dass die Arbitrierung bei einem belegten CAN-Bus unter realen Bedingungen dazu führt, dass sich die maximalen Messergebnisse zwischen dem VW-Gateway und dem Gateway-2017 nur um 18% unterscheiden. Die minimalen Messergebnisse differieren um 31%. Daraus lässt sich folgern, dass bei einem CAN-Bus mit einer hohen Auslastung die Arbitrierungszeit steigt und sich so der relative Abstand der Latenz zwischen dem Gateway-2017 und dem VW-Gateway verringert.

## 9.4 Scheduling, ISR und Super Loop

Die Umsetzung des Gateways-2017 mit einem Scheduler zeigte extreme Schwankungen in der Latenz (Abbildung 9.1), da sich die maximale Latenz um eine Scheduling Periode erhöht. Um die Latenz des VW-Gateways zu erreichen, könnten statt des Schedulers nur ISRs oder eine Super Loop verwendet werden. Die Portierung zu einem Gateway, das nur ISRs enthält, könnte Vorteile bei der maximalen Latenz haben, da die ISR direkt bei Eingang eines Frames ausgeführt werden würde. Die Nachrichtenbündelung benötigt aber weiterhin einen Timer, der derzeit von der Task „Sende Buffer via RTE“ hochgezählt wird. Diese Operation müsste weiterhin im Scheduler durchgeführt werden oder es müsste eine weitere ISR auf einen HW Timer registriert werden. Eine weitere Möglichkeit wäre es, alle Tasks in einer Super Loop

auszuführen. Das hätte den zusätzlichen Vorteil, dass weniger Kontextwechsel zwischen ISRs, Scheduler und Super Loop nötig wären, da es in diesem Fall nur eine Funktion im Scheduler, eine ISR und eine Super Loop gäbe.

### 9.4.1 Nachrichtenbündelung

Bei der Verwendung von ISRs oder einer Super Loop muss das Schedule nicht mehr auf die Anzahl gebündelter Frames angepasst werden. Die Laufzeit der Funktionen variiert abhängig von der Anzahl der zu versendenden, zu empfangenden und zu verarbeitenden Frames, da die Priorisierung nur beim Empfang von CAN-Frames und beim Senden von Ethernet-Frames eine Auswirkung hat. Es kann möglicherweise folgendes Szenario eintreten, dass ein Ethernet-Frame mit 166 CAN-Frames empfangen wird und alle CAN-Frames im Buffer RTE_TO_CAN gespeichert werden. Wird direkt danach ein weiterer Ethernet-Frame mit nur einem CAN-Frame empfangen, muss der CAN-Frame hinten einsortiert werden. Von daher sollte die Nachrichtenbündelung nicht verwendet werden, wenn die Latenz für mindestens einen CAN-Frame wichtig ist.

### 9.4.2 Versuch mit Super Loop

Um zu zeigen, dass sich die gesamte Latenz des Gateways-2017 durch den Einsatz einer Super Loop verbessern lässt, wurde der Aufruf der 5 Tasks aus dem Scheduler in die „Background (BG) Loop“ des Schedulers kopiert. Diese wird ausgeführt, wenn gerade kein Task im Schedule aktiv ist. Da keine Tasks aktiv sind, verhält sich diese Konfiguration wie eine Super Loop. In einer Super Loop stehen alle abzuarbeitenden Task hintereinander in einer Endlosschleife und werden so zyklisch ausgeführt. Für einen produktiven Einsatz ist diese Lösung allerdings nicht geeignet, da der Task „Sende Ethernet HW-Buffer“ eine Scheduler interne Funktion ist und nicht aus der „BG Task“ aufgerufen werden sollte. Für die folgenden Latenz-Messungen wurde diese Funktion im Messzeitraum jedoch manuell aus der „BG Task“ aufgerufen. So lässt sich abschätzen, ob sich der Aufwand für die Portierung lohnt.

### CAN nach Ethernet

Zunächst wird wie auf Seite 54 die Latenz des Gateway-2017 von CAN nach Ethernet vermessen.

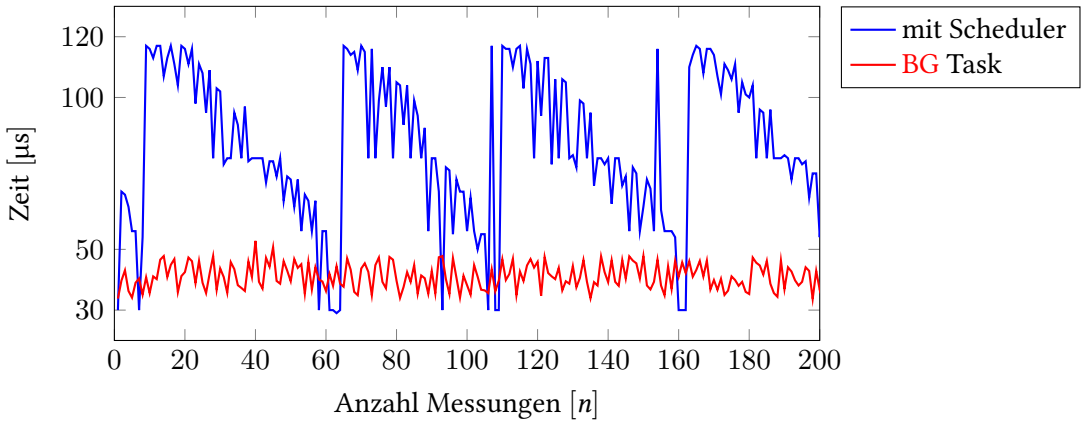



Abbildung 9.9: Messung der Latenz von CAN nach Ethernet mit Super Loop und Scheduler

Beim Einsatz der „BG Task“ ist kein Sägezahnmuster mehr vorhanden. Die Messergebnisse der „BG Task“ liegen in etwa bei den berechneten Werten in der Tabelle (siehe Tabelle 7.4 auf Seite 48). Die Schwankungen in der Messung ergeben sich durch die unterschiedlichen Zeitpunkte, an denen ein CAN-Frame beim Gateway-2017 eintrifft, und durch verschiedene Positionen, in denen sich das Gateway-2017 in der „BG Task“ befindet.

### Ethernet nach CAN

Um festzustellen, ob sich eine weitere Verbesserung erreichen ließ, wurde ebenfalls die Latenz von Ethernet nach CAN vermessen.

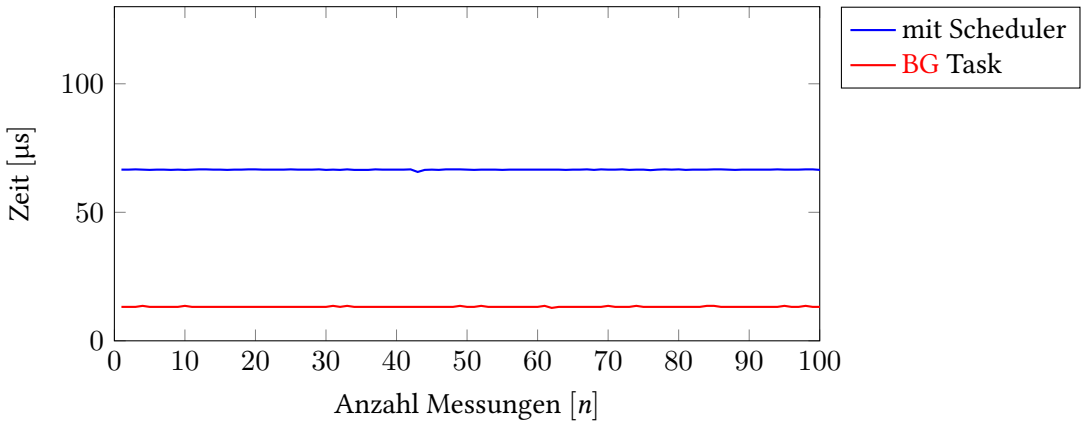



Abbildung 9.10: Messungen der Latenz von Ethernet nach CAN mit Super Loop und Scheduler

Da ein Schedule immer auf die maximale Ausführungsdauer aller Tasks ausgelegt ist und der Task zwischen den beiden Funktionen bei keinem eingegangen CAN-Frame nur 2µs benötigt, während für den Worst Case beim „Latenz optimierten“ Schedule 20 µs eingeplant sind, ist auch hier das Ergebnis mit der „BG Task“ deutlich besser. Die Latenz liegt konstant zwischen 13 und 14 µs. Damit bleibt die Latenz sogar unter den berechneten Laufzeiten der beiden Funktionen aus der Tabelle 7.4.

### CAN nach CAN

Bei dieser Messung handelt es sich um die gleiche Messung wie sie in Abschnitt 9.1 auf Seite 57 durchgeführt wurde. Das Gateway-2017 verwendet hier eine Super Loop:

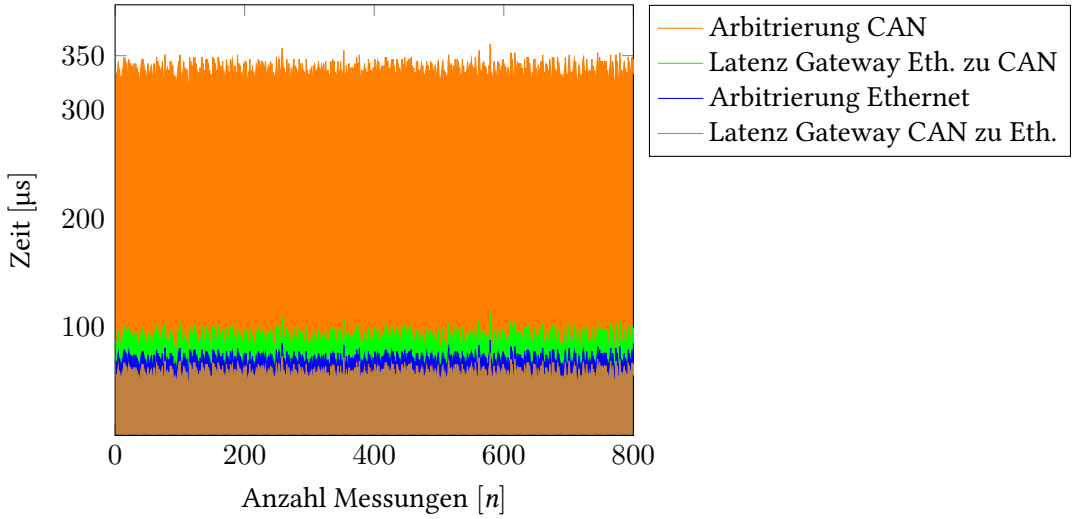



Abbildung 9.11: Messung der Latenz von CAN nach CAN mit Super Loop

Im Vergleich zu den Messungen mit dem Scheduler liegt die gesamte Latenz (grün + blau + braun) nun bei etwa 100 µs und somit niedriger als beim Verfahren mit dem Scheduler, wo die Werte etwa bei 300 µs (Abbildung 9.4) liegen. Die Übertragungsdauer von Ethernet-Frames benötigt bereits über 10% der gesamten Latenz. Da das Gateway-2017 von Ethernet nach CAN kein Regelwerk besitzt, ist hier die Latenz deutlich geringer als die Latenz von CAN nach Ethernet. Um die Messungen aus dem Labor mit den Messungen vom VW-Gateway aus der realen Umgebung vergleichen zu können, wurden bei der Messung aus dem Prototypenfahrzeug alle Werte herausgefiltert, die vermutlich durch einen belegten CAN-Bus entstanden sind. Hierzu wurde ein Histogramm erstellt um die Schwelle zu ermitteln, ab der es nur noch eine einstellige Anzahl an Messergebnissen gibt.

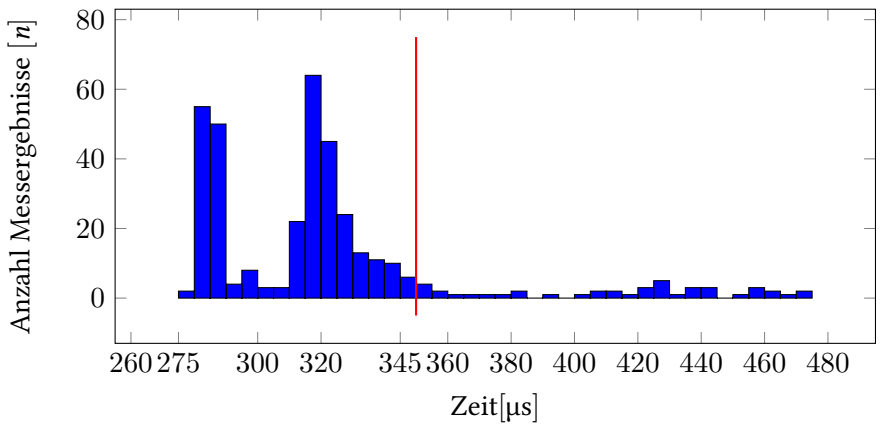



Abbildung 9.12: Häufigkeitsverteilung der Messergebnisse vom VW-Gateway

Diese Schwelle liegt bei 345 µs mit 6 Messergebnissen. Alle höheren Werte (rechts von der roten Linie) werden bei der Berechnung für diese Tabelle nicht berücksichtigt.

| **Gateway** | **min** | **max** | **avg** |
|---|---|---|---|
| VW-Gateway | 275 µs | 345 µs | 337,64 µs |
| Gateway-2017 | 317 µs | 361 µs | 335,97 µs |

Tabelle 9.2: Histogramm bereinigte Messergebnisse

Bei dieser Auswertung (avg) liegen beide Gateways etwa gleich auf. Wenn die „Super Loop“ noch verbessert wird, könnte weitere Zeit eingespart werden. Beispielsweise könnte die Nachrichtenbündelung entfernt werden.

Wären die Tasks inline (Kompileroption), könnte weitere Zeit (1 µs pro Task) eingespart werden.

Der SW-Buffer RTE_TO_CAN könnte entfernt und CAN-Frames direkt in den HW-Buffer kopiert werden.

Die minimale Latenz (Tabelle 9.4.2) 275 µs abzüglich der Arbitrierungsdauer (Formel 9.1) 244 µs ergibt rechnerisch 31 µs beim VW-Gateway gegenüber 69 µs beim Gateway-2017. Der Wert des VW-Gateways wäre schwierig zu erreichen, da alleine die Ethernet-Übertragungsdauer bei 13,44 µs liegt. Durch die Verwendung von einem 1 GBit/s Ethernet-Backbone oder einen Ethernet-Switch, der fast forward verwendet, wäre diese Zeitdauer zu verringern. Der µC besitzt nur einen 100 Mbit/s Ethernet-Controller, sodass er durch ein Modell mit einem 1 GBit/s Ethernet-Controller ersetzt werden müsste.

# 10 Fazit & Aussicht

In diesem Kapitel wird aus den vorangegangen Erkenntnissen ein Fazit gezogen. Darüber hinaus wird beschrieben, ob das Gateway-2017 auf einen anderen µC portierbar wäre und welche Fragestellungen sich an diese Arbeit anschließen.

## 10.1 Fazit

Durch den Einsatz eines CAN-Ethernet Gateways können vorhandene CAN basierte ECUs an einem Ethernet-Backbone weiter betrieben werden ohne dass zusätzliche Entwicklungskosten durch eine Migration der vorhandenen ECUs entstehen.

Das Ethernet ist ein etablierter Standard bei der Vernetzung von Computern. Im Gegensatz zu Feldbussystemen wie CAN sind höhere Bandbreiten möglich. Die Übertragungsdauer eines Ethernet-Frames über den Ethernet-Backbone, inklusive eines gebündelten CAN-Frames, ist deutlich geringer als die Übertragungsdauer eines CAN-Frames auf dem CAN-Bus.

Wie in dieser Bachelorarbeit gezeigt wurde, kann ein CAN-Ethernet Gateway die gleichen Latenzen erreichen wie ein zentrales CAN-Gateway (VW-Golf 7). Das Vergleichen und Erfassen von Messwerten im Prototypenfahrzeug gestaltete sich schwierig, da die Zuordnung der CAN-Frames zu den entsprechenden Funktionen nicht veröffentlicht ist. Dies führte zu teilweise aufwändigem Reverse Engineering.

Da in zukünftigen Fahrzeugen ein Ethernet-Netzwerk vorhanden sein wird, kann auch die Kommunikation zwischen den CAN-Bussen über Ethernet erfolgen. Eine separate Verkabelung der verschiedenen CAN-Busse zu einem zentralen Gateway ist nicht mehr erforderlich.

Für eine Übergangsphase hin zu einem vollständigen Ethernet-Netzwerk könnten zwei CAN-Ethernet Gateways (Super Loop) fast die Latenz eines zentralen CAN Gateways (VW-Gateway) erreichen, wenn auf RTE verzichtet wird. Auch einen voll ausgelasteten CAN-Bus kann das CAN-Ethernet Gateway verarbeiten. CAN-Frames werden erst verworfen, wenn der CAN-Bus überlastet ( >100%) wird. Kurze Störungen auf dem CAN-Bus kann das CAN-Ethernet Gateway (CAN-Ausgang) überbrücken, da bis zu 512 CAN-Frames zwischengespeichert werden.

Damit eine Nachrichtenbündelung effektiv funktioniert, muss nach dem Empfang eines CAN-Frames auf den Ablauf eines Timers gewartet werden. Hierdurch verzögert sich der entsprechende Ethernet-Frame-Versand. Wenn also die Latenz einer CAN-ID gering sein soll, muss auf die Nachrichtenbündelung für diese verzichtet werden. Eine Priorisierung ist zwar beim Empfang eines CAN-Frames möglich, beim Empfang eines Ethernet-Frames ist jedoch keine Priorisierung implementiert. Daraus folgt, dass ein hoch priorisierter CAN-Frame, der in einem Ethernet-Frame gebündelt ist, beim Empfang in eine Warteschlange geraten kann. Um diese Warteschlangen zu verhindern, sollte die Nachrichtenbündelung abgeschaltet werden oder die Implementierung um eine Priorisierung beim Empfang einen Ethernet-Frames erweitert werden.

Da die einzelnen Tasks des Gateways eine kurze Laufzeit besitzen und der verwendete Scheduler eine hohe Laufzeit zwischen den Tasks, ist die Verwendung dieses Schedulers für den Einsatz in einem CAN-Ethernet Gateway bei hohen Latenz-Anforderungen nicht geeignet.

Es lässt sich außerdem festhalten, dass der Kompilationsansatz im Gegensatz zum Interpretationsansatz zu einer besseren Laufzeit führt, z.B. um Faktor 7 bei CAN nach Ethernet. Der Quellcode des Gateways kann daher schlanker gehalten und viele Berechnungen können vom Code Generator im Voraus getätigt werden. Um die Suchzeit nach einer auszuführenden Regel im Regelwerk möglichst gering zu halten, ist die Verwendung eines binären Suchbaums vorteilhaft. So liefert beispielsweise die lineare Suche mit 200 Regeln eine um Faktor 6 längere Laufzeit der gesamten Funktion. Bei der Verwendung von Regeln mit einem 29er prefix erwies sich die binäre Suche als die schnellste Lösung.

## 10.2 Aussicht

Im nachfolgenden Kapitel wird der Aufwand abgeschätzt, der für eine Migration auf einen anderen µC nötig wäre. So bräuchte man in Zukunft kein weiteres Gateway mehr komplett neu zu entwickeln. Erweiterungs- und Verbesserungsmöglichkeiten schließen die Arbeit ab.

### 10.2.1 Portierbarkeit

Für zukünftige Erweiterungen, bei denen mehr als eine CAN- oder Ethernet-Schnittstelle verwendet werden sollen, ist der µC vermutlich nicht ausreichend dimensioniert, da seine Auslastung bereits jetzt mit 1 MBit/s Bandbreite bei nahezu 90 % liegt (Seite 48 Abschnitt 7.4.6). Wenn nur eine Bandbreite von 500 KBit/s gewünscht ist, sollte die HW auch für eine zweite CAN- oder Ethernet-Schnittstelle ausreichen. Der limitierende Faktor ist hier die Verarbeitungszeit pro CAN-Frame.

#### Code Generator

Der Source-Code, der vom Code Generator generiert wird, umfasst die Dateien „function.c“, „function.h“, „importedRules.c“, „importedRules.h“, „buffer.c“, „buffer.h“ und „tree.h“. In diesen Dateien werden die Regeln und Buffer geladen und die Datenstrukturen definiert. Der Source-Code verwendet die Libaries „stdint.h“ und „string.h“ und ist „std=c89“ kompatibel.

Der vom Code Generator generierte Code lässt sich direkt für einen anderen µC kompilieren, da davon auszugehen ist, dass für jeden modernen µC ein Compiler zur Verfügung steht, der mindestens „std=c89“ und beide oben genannten Libaries unterstützt. Für die Portierung auf einen anderen µC muss keine Anpassung am Code Generator vorgenommen werden.

#### Gateway

Die Datei „can.c“ enthält die ISR sowie die Funktionen zum Buffern und Senden von CAN-Frames. Sie muss angepasst werden, da sie die CAN-API des µC nutzt. Im „bufferManagement.c“ muss die Funktion „extractBuffer“ verändert werden, da sie als Parameter „CAN_FRAME_t *“ von der CAN-API erfordert. Die rules- und die timer-Dateien müssen nicht angepasst werden. Der Scheduler von Kai Müller müsste ebenfalls für eine neue Plattform portiert werden. Da die Laufzeit des Schedulers zwischen 2 Tasks zu groß ist, wäre es sinnvoller, direkt auf eine ISR oder Super Loop basierte Lösung zu portieren.

### 10.2.2 Zukünftige Arbeiten

Eine Portierung zu einer Super Loop oder ISR gestützten Implementierung des Gateways-2017 wäre interessant, um zu vergleichen, wie sich dies auf Laufzeiten und Latenz auswirkt. In diesem Zusammenhang wäre außerdem eine Lösung zu entwickeln, die alle Berechnungen in einer Super Loop durchführt und nur das Versenden von RTE-Frames im Schedule vorsieht. So könnte eine niedrigere Latenz trotz RTE erreicht werden.

Die Performance des Gateways-2017 könnte weiter gesteigert werden, da auch bei abgeschalteter Nachrichtenbündelung ein CAN-Frame in den zugehörigen Buffer gespeichert wird und der Buffer an den Anfang der Buffer-Liste (Insertion Sort) umsortiert wird. Auf diese Sortierung könnte verzichtet werden, der Buffer direkt verschickt und so die Performance auf dem Weg CAN nach Ethernet erhöht werden.

Damit einzelne CAN-Frames trotz Nachrichtenbündelung hoch priorisiert behandelt werden, könnte ein zusätzliches Regelwerk auf der Ethernet-Empfangsseite angelegt werden.

Außerdem könnte der Code Generator um eine Option erweitert werden, die bei der binären Suche zusammengefasste Regeln in mehrere kleine Regeln umwandelt, um eine übersichtliche

Konfigurationsdatei zu behalten und trotzdem die Performance-Vorteile der binären Suche nutzen zu können.

Das Gateway-2017 kann auf einen anderen µC portiert werden und für andere Feldbussysteme wie Flexray, MOST oder LIN angepasst werden. Zum Testen der Konfiguration wäre eine Portierung des Regelwerks des Gateways-2017 in ein ANDL kompatibles Format hilfreich. So könnte die Konfiguration im Labor getestet werden und erst danach müsste das Ergebnis im Prototypenfahrzeug validiert werden. Da die CoRE Gruppe dieses Jahr (2017) ein neues Prototypenfahrzeug erhält, wäre es sinnvoll, das neue Fahrzeug entsprechend vorzubereiten.

Um den Ethernet-Backbone weiter zu entlasten und die Anzahl der auszuführenden Regeln zu verringern, könnten Multicast-Adressen eingesetzt werden. Zurzeit muss ein CAN-Frame, der an zwei Gateways-2017 gesendet werden soll, an zwei Ethernet-Empfänger geschickt werden. Wenn dieser nun an eine Multicast-Adresse versendet würde, auf der mehrere Gateways-2017 lauschen, bräuchte nur noch ein Ethernet-Frame verschickt zu werden.

Beim Einsatz von RTE könnte die Latenz verbessert werden, indem das „Sync Window“ verkleinert wird. Um den „Sync Window“ Task möglichst selten auszuführen, müsste zunächst festgestellt werden, wie synchron die Uhren der verschiedenen Teilnehmer am RTE laufen. Zu testen wäre, ob weitere Latenz-Verbesserungen zu erreichen sind, wenn dem „Sync Window“ eine geringere Priorität zugewiesen oder das „Sync Window“ in mehrere kleine Tasks aufgeteilt wird.

# Abkürzungsverzeichnis

**µC** Microcontroller 12, 28, 36, 40, 49, 55, 59, 62, 69–73

**ACC** Adaptive Cruise Control 61
**ACK** Acknowledge 6, 16
**ANDL** Abstract Network Description Language 38, 73
**API** application programming interface 28, 41, 45, 62, 72
**ARM** Advanced RISC Machines 12

**BG** Background 65–67

**CAN** Controller Area Network 1–8, 11–28, 30, 34–48, 51–58, 60–73, 76
**CoRE** Communication over Realtime Ethernet 1, 12, 38, 60, 73
**CRC** cyclic redundancy check 6, 9, 26, 76
**CSMA/CD** Carrier Sense Multiple Access/Collision Detection 10
**CSV** Comma-separated values 18

**E** Extended 12, 60, 62
**ECU** Electronic Control Unit 1, 26, 70
**EoF** End of Frame 6, 7, 57

**FCS** Frame Check Sequence 9

**GCC** GNU Compiler Collection 33
**GPIO** General Purpose Input Output 41, 54–56, 59

**HAW** Hochschule für Angewandte Wissenschaften 1, 2, 13
**HW** Hardware 21, 34, 35, 47, 48, 52, 64, 69, 71

**I** Infotainment 12, 60, 61
**ID** Identifier 2, 5, 15–20, 22, 25, 26, 30, 37, 39, 42–44, 60–62, 71
**IDE** Identifier Extension 5, 6
**Ifs** Interframe spacing 6
**IP** Internet Protocol 11, 18
**IPG** Inter-Packet Gap 9
**ISR** interrupt service routine 20, 34, 35, 41, 47, 54, 56, 58, 60, 64, 65, 72

**K** Komfort 12, 60, 62

**LIN** Local Interconnect Network 4, 73

**MAC** Media Access Control 9, 15, 17, 20, 23, 39
**MOST** Media Oriented Systems Transport 4, 13, 73

**OSI** Open Systems Interconnection 11, 76

**RISC** Reduced Instruction Set Computer 12, 74
**RS** Radio Sector 12
**RTE** Realtime Ethernet 1, 3, 70, 72, 73
**RTR** Remote Transmission Request 6

**SD** Secure Digital 34
**SFD** Start Frame Delimiter 9
**SoF** Start of Frame 5
**SRRB** Substitute Remote Request Bit 6
**SW** Software 28, 39, 69

**TCP** Transmission Control Protocol 14
**TDMA** Time Division Multiple Access 10

**UDP** User Datagram Protocol 14

**VLAN** Virtual Local Area Network 40

**VW** Volkswagen 2, 3, 12–14, 60, 63, 64, 68–70

# Glossar

### doxygen

Mithilfe des Tools „doxygen“ kann eine einheitliche Dokumentation von Programmen verschiedener Programmiersprachen erstellt werden. 34

### fast forward

Beim fast forward-Verfahren speichert der Switch den Frame bis zur Ziel-Adresse und leitet den Frame dann direkt weiter. Daher erhöht sich die Latenz nur um die Übertragungsdauer von Preamble und Ziel-Adresse. Anders als beim store and forward-Verfahren ist eine Prüfung der CRC Summe nicht möglich. 69

### Gateway

Ein Gateway verbindet zwei Netzwerke, die verschiedene Protokolle im OSI Schichten-Modell von Layer 1-7 verwenden. Da diese Protokolle unterschiedliche Informationen enthalten, darf ein Gateway Daten verändern oder entfernen, wenn es für die Übertragung erforderlich ist. 1–4, 10, 12–21, 23–28, 32–34, 36–41, 43, 44, 46–48, 50, 51, 54–58, 60–66, 68–73, 76

### Insertion Sort

„Insertion Sort“ ist ein Sortierverfahren, bei dem alle Elemente miteinander verkettet sind. Ist die Liste bereits sortiert, kann ein Element, bei dem sich der Wert verändert hat, mit einer Laufzeit von O(n) an die neue Position verschoben werden. Hierzu wird das Element zunächst aus der Kette entfernt, die neue Position linear gesucht und dort eingefügt. 21, 72

### javadoc

Mithilfe des Tools „javadoc“ kann eine einheitliche Dokumentation eines in Java programmierten Programms erstellt werden. 33

### Latenz

Der Begriff „Latenz“ wird verwendet für Messergebnisse, die von außerhalb durchgeführt wurden (Black-Box-Test), die Übertragung von CAN nach Ethernet, Ethernet nach CAN und von CAN nach CAN über Ethernet. Alle Laufzeiten und Verzögerungen durch den Scheduler oder durch Kontext-Wechsel, vom Ein- bis zum Ausgang gemessen, ergeben summiert die Latenzen des Gateways. 1–3, 13, 14, 41, 50–58, 62–73

**Laufzeit**

Der Begriff „Laufzeit“ wird verwendet, um die Verarbeitungsdauer einer einzelnen Funktionen zu beschreiben. Hierzu wurden meist Änderungen am Software-Code vorgenommen (White-Box-Test). Außerdem wird der Begriff „zur Laufzeit“ für die Beschreibung von Berechnungen genutzt, die zur Ausführungszeit eines Programms durchgeführt werden. 2, 3, 14, 19–21, 26–29, 35, 36, 40–48, 50–52, 54, 56, 58, 60, 65, 67, 71, 72, 76

**NetzID**

Die NetzID ist das Ergebnis aus der „Und“ Verknüpfung von einer Adresse mit dem dazugehörigen prefix. 18–20

**prefix**

Der Prefix ist eine Kurzform der Subnetzmaske. Der Prefix gibt die Anzahl der Einsen in der Subnetzmaske an. So entspricht z.B. die Subnetzmaske 255.255.255.0 einem Prefix von 24. 18–20, 29, 30, 42, 43, 71, 77

**store and forward**

Bei der Übermittlung eines Ethernet-Frames speichert ein Switch den Frame zunächst komplett. Nur wenn dieser korrekt ist, leitet der Switch den Frame weiter. Daher verdoppelt sich die Latenz im Vergleich zur direkten Übertragung. 57, 76

**Subnetzmaske**

Die Subnetzmaske definiert, wie groß ein Netz ist. Alle auf 1 gesetzten Bits bilden den Netzanteil und alle auf 0 gesetzten Bits den Hostanteil. Ein Beispiel ist die 255.255.255.0, die eine IPv4 Subnetzmaske ist. Werden zwei Adressen mit einer Subnetzmaske „Und“ verknüpft und ist das Ergebnis identisch, befinden sich beide Adressen im gleichen Netz. 18, 77

**Unshielded-Twisted-Pair-Kabel**

Bei diesem Kabeltyp handelt es sich um ein ungeschirmtes Kabel. Alle Drähte sind paarweise verdrillt. 5

# Literaturverzeichnis


CAN in Automation (2014). *Growing car sales increase CAN business*. URL: https://can-newsletter.org/engineering/engineering-miscellaneous/nr_growing-car-sales-increase-the-can-business_140107/ (besucht am 19. 12. 2016).

CoRE-Arbeitsgruppe (2014). *Communication over Real-time Ethernet*. Hamburg. URL: http://core.informatik.haw-hamburg.de (besucht am 18. 08. 2014).

D'Ambrosia, John (2010). "100 gigabit Ethernet and beyond [Commentary]". In: *Communications Magazine, IEEE* 48.3, S6–S13. ISSN: 0163-6804. DOI: 10.1109/MCOM.2010.5434372.

Depke, Jan (2015). "Inkrementelle Konsolidierung automobiler Bussysteme auf Basis eines Realtime Ethernet Backbones". mastersthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.

FlexRay Consortium (2010). *FlexRay Communications System Electrical Physical Layer Specification*. Specification 3.0.1. Stuttgart: FlexRay Consortium.

Hilscher GmbH (2013). *NXHX 51-ETM - Development Board*. Hilscher GmbH. URL: http://www.hilscher.com/fileadmin/cms_upload/de/Resources/pdf/NXHX_51-ETM_Development_Board_HW_05_EN.pdf (besucht am 31. 10. 2016).

IEEE Ethernet Group (1985). *IEEE Standard for Ethernet*. URL: https://standards.ieee.org/about/get/802/802.3.html.

International Organization for Standardization (2003). *Road vehicles – Controller Area Network (CAN) Part 2: High-speed medium access-unit*. ISO 11898. Genf: ISO.

– (2015). *Road vehicles – Controller Area Network (CAN) Part 1: Data link layer and pyhsical signaling*. ISO 11898. ISO.

Kuncke, Patrick (2016). "Erprobung von Echtzeit Ethernet basierten Automobil-Gateways in einem Prototypfahrzeug". bachelorsthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.

LIN Consortium (2010). *Lin Specification Package Rev. 2.2A*. URL: https://www.cs-group.de/fileadmin/media/Documents/LIN_Specification_Package_2.2A.pdf (besucht am 07. 10. 2016).

Medhat, Hassan Maha (2015). *Bit stuffing techniques Analysis and a Novel bit stuffing algorithm for Controller Area Network (CAN).* URL: http://www.ijcsonline.com/IJCS/IJCS_2015_0203007.pdf (besucht am 21. 12. 2016).

MOST Cooperation (2011). *Media Oriented Systems Transport.* URL: http://www.mostcooperation.com/ (besucht am 06. 01. 2011).

Müller, Kai (2011). "Time-Triggered Ethernet für eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfähigen Netzwerkstack-Architektur". bachelorsthesis. Hamburg: Hochschule für Angewandte Wissenschaften Hamburg.

Stasch, Eckhard (1997). Chemnitz. URL: https://www.tu-chemnitz.de/informatik/RA/news/stack/kompendium/vortraege_97/ethernet/technologie.html (besucht am 21. 12. 2016).

Steinbach, Till u. a. (2016). "Extending OMNeT++ Towards a Platform for the Design of Future In-Vehicle Network Architectures". In: *Proceedings of the 3rd OMNeT++ Community Summit, Brno, Czech Republic, September 15, 2016.* Hrsg. von Anna Foerster u. a. ArXiv e-prints. URL: https://core.informatik.haw-hamburg.de/bib/eigene/smbk-eotpd-16.pdf (besucht am 06. 01. 2017).

Tanenbaum, Andrew S. und David J. Wetherall (2012). *Computernetzwerke* -. 5. Aufl. Muenchen: Pearson. ISBN: 978-3-868-94137-1.

Thomas, G., D. Davids und O. Holme (2015). *CAN over Ethernet Gateway - A Convenient and flexible Solution to Access Low Level Control Devices.* "Melbourne, Australia". URL: http://icalepcs.synchrotron.org.au/papers/mopgf120.pdf (besucht am 19. 12. 2016).

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

Hamburg, 18. April 2017 Lukas Wendt