HAW HAMBURG

BACHELORTHESIS
Kai Steffen Wienberg

Implementierung und Evaluation einer Time-Sensitive Software-Defined Networking Architektur für den Automobilbereich

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Kai Steffen Wienberg

Implementierung und Evaluation einer Time-Sensitive Software-Defined Networking Architektur für den Automobilbereich

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. Thomas Schmidt
Zweitgutachter: Prof. Dr. Franz Korf

Eingereicht am: 25. Februar 2020

Kai Steffen Wienberg

Thema der Arbeit

Implementierung und Evaluation einer Time-Sensitive Software-Defined Networking Architektur für den Automobilbereich

Stichworte

TSSDN, OpenFlow, NETCONF, SDN, TSN, ONOS

Kurzzusammenfassung

In dieser Bachelorarbeit wird eine Applikation zur Konfiguration eines Time-Sensitive Software-Defined Networking (TSSDN)-Switches konzeptioniert und implementiert. Daraufhin wird anhand verschiedener Testarchitekturen und Testfälle evaluiert, ob TSSDN und der Switch die Anforderungen im Automobilbereich erfüllen können. Durch genaue Messungen der Latenz und Berechnung des Jitters wird ermittelt, wie sich Cross-Traffic auf priorisierte Nachrichten auswirkt und ob durch Scheduling der Nachrichten maximale Laufzeiten garantiert werden können.

Kai Steffen Wienberg

Title of Thesis

Implementation and Evaluation of a Time-Sensitive Software-Defined Networking Architecture for the Automotive Sector

Keywords

TSSDN, OpenFlow, NETCONF, SDN, TSN, ONOS

Abstract

In this bachelor thesis, an application to configure a Time-Sensitive Software-Defined Networking (TSSDN) switch is designed and implemented. An evaluation is carried out based on different test architectures and test cases to answer the question whether TSSDN and the network switch can satisfy the requirements of the automotive sector. By performing precise latency measurements and calculating the resulting jitter it will be determined how cross-traffic impacts prioritized messages and if a maximum latency can be guaranteed using scheduling.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Im Automobilbereich befinden sich die Kommunikationsnetzwerke derzeit im Wandel, da neue Anwendungen aus den Bereichen der Fahrerassistenzsysteme und des Infotainments sowie insbesondere des automatisierten und autonomen Fahrens einen weit höheren Bedarf an leistungsfähigen Kommunikationsverbindungen haben, als bisherige im Automobil eingesetzte Systeme garantieren können [18]. Dies liegt vor allem an der wachsenden Anzahl neuer Sensorik, wie zum Beispiel HD-Kameras, LIDAR und Radar, welche die Umwelt mit einem deutlich höheren Detailgrad aufzeichnen und daher weit höhere Bandbreiten als bisherige Systeme übertragen müssen, sowie der vermehrten Kommunikation zwischen Systemen, die zunehmend über Anwendungsdomänen hinaus stattfindet [18].

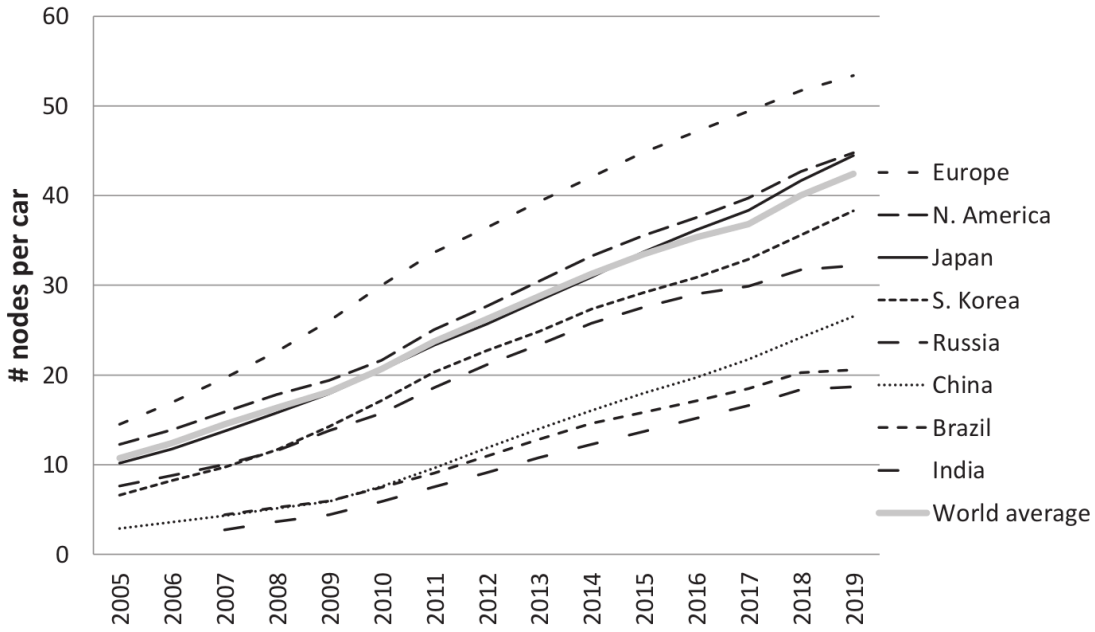


Abbildung 1.1: Durchschnittliche Anzahl von Netzwerkknoten im Auto, abhängig von der Region (Quelle: [14] Figure 2.2)

In Abbildung 1.1 ist dargestellt, wie sich die Anzahl an Kommunikationsknoten im Au-

tomobilnetzwerk über die letzten Jahre stetig erhöht hat.
Bereits 2016 wurde das Kommunikationsnetzwerk eines Autos inklusive der dort geltenden Echtzeitanforderungen mit einer Netzwerkarchitektur auf Basis von dem Echtzeit-Ethernet "TTEthernet" (SAE AS6802) umgesetzt [18]. Eine weitere Echtzeit-Ethernet Lösung ist "Time-Sensitive Networking (TSN)", hinter welcher unter anderem die Flugzeug- und Automobilindustrie als treibende Kraft stehen ([18] Kapitel 2.1.3). Seit 2018 ist TSN zudem in den Standard IEEE 802.1Q [9] integriert.
Eine weitere vielversprechende Technologie ist Software-Defined Networking (SDN), welche die Kontrollebene des Netzwerks von den Switches auf einen zentralen Host verlagert, um somit zentralisierte Entscheidungen über den Netzwerkfluss treffen zu können. Dies hat eine Reihe von Vorteilen in Automobilnetzwerken. So kann durch SDN im Fall einer beschädigten Netzwerkleitung der Netzwerkfluss dynamisch über andere Leitungen umgeleitet werden, was die Sicherheit im Auto erhöht [6]. Es können zudem frei konfigurierbare Netzwerktopologien geschaffen und eine dynamische Rekonfiguration des Netzwerkflusses durch den Host durchgeführt werden [6]. Durch dieses hohe Maß an Flexibilität können neue Funktionen im Auto leichter und schneller umgesetzt werden [6]. SDN erhöht zudem die Sicherheit, weil die Weiterleitung von Nachrichten im Netzwerk sehr spezifisch erlaubt und blockiert werden kann, wie z. B. auf IP-, Protokoll- und Port-Basis. Die Vorteile der Latenzgarantien aus TSN als Echtzeit-Ethernet Lösung und die Vorteile von SDN zur flexiblen Flusskonfiguration können laut Simulationsmodellen in Kombination ohne Leistungseinbußen genutzt werden [6]. Die Kombination beider Technologien wird als **Time-Sensitive Software-Defined Networking (TSSDN)** bezeichnet und ist aufgrund der soeben genannten Vorteile interessant für den Automobilbereich [6].
Diese Arbeit ist in der CoRE RG (Communication-over-Realtime-Ethernet Research Group) der HAW Hamburg entstanden, welche sich mit der Implementierung von Echtzeit-Ethernet-Kommunikation in Fahrzeugen befasst.
In dieser Arbeit soll eine Lösung zur Konfiguration einer TSSDN-Netzwerkarchitektur mit den herstellerunabhängigen Protokollen "OpenFlow" und "NETCONF" konzeptioniert und implementiert werden. Dies umfasst eine Applikation, welche zwischen den Fahrzeugmodi "Fahren" und "Update / Diagnose" umschalten kann und dabei abhängig vom Modus den Netzwerkverkehr korrekt weiterleitet. Des Weiteren soll die entwickelte Lösungsarchitektur anhand echter Anforderungen aus dem Automobilbereich auf einem TSSDN-Switch (TrustNode IRTN16R als Testgerät) evaluiert werden.
Struktur der Arbeit: Kapitel 2 bringt die Grundlagen von TSSDN näher, die für das bessere Verstehen der Arbeit erforderlich sind. Dazu gehören diverse Ethernet-Standards sowie die Protokolle OpenFlow und NETCONF. Das Kapitel 3 enthält die Anforderun-

gen an die Lösung und leitet daraus die erforderlichen Testfälle ab. In Kapitel 4 wird das Konzept zur Umsetzung der Anforderungen vorgestellt und eine Methode zur Berechnung gültiger Netzwerk-Zeitpläne (Schedules) wird erarbeitet. Die verwendete Hardware und tatsächliche Implementierung des Konzeptes sowie das Vorgehen bei der Durchführung von den Testfällen werden in Kapitel 5 erläutert. Kapitel 6 stellt die Testergebnisse vor und es wird auf Basis der Anforderungen evaluiert, ob sich die erarbeitete Lösung für den Einsatz im Automobilbereich eignet. Abschließend wird in Kapitel 7 ein Fazit sowie ein Ausblick gegeben.

2 Grundlagen

In diesem Kapitel wird kurz der aktuelle Stand von Ethernet im Automobilbereich angeschnitten. Daraufhin werden die grundlegenden Funktionsweisen von TSN und SDN beschrieben. Diese Oberbegriffe bestehen für sich gesehen jeweils aus Unterprotokollen, Spezifikation und Komponenten, auf welche eingegangen wird.

2.1 Ethernet im Automobilbereich

Im Automobilbereich werden heutzutage viele unterschiedliche Kommunikationstechnologien genutzt, welche alle für einen speziellen Anwendungsfall entwickelt wurden: CAN für robuste Kommunikation zwischen Steuergeräten (Electronic Control Units, ECUs), LIN für niedrige Kosten, MOST für hochauflösende Audioübertragung, FlexRay für X-by-Wire, Pixel Links für unkomprimierte Videoübertragung und Consumer Links für die Anbindung von Endgeräten der Kunden. Diese Technologien unterscheiden sich nicht nur in den unterstützten Datenübertragungsraten, sondern auch in den Kommunikationsmechanismen (siehe Tabelle Abbildung 2.1).
Jeder neue Anwendungsfall führt zu neuen Anforderungen, Standardisierungen, Kommunikationsprinzipien und Zertifizierungen. Dies bindet sehr viele Ressourcen bezüglich der Entwicklungs- und Test-Teams, vor allem weil die Technologien immer komplexer werden. Neue Technologien verlangen auch nach dem Training neuer Spezialisten, die Inkonsistenzen und Probleme lösen können. Daher ist ein leistungsstarkes Netzwerk im Automobil zwar eine fundamentale Anforderung, aber die Anzahl an Netzwerktechnologien muss so klein wie möglich gehalten werden. Ethernet im Automobil bringt die Möglichkeit, eine Konsolidierung voranzutreiben ([14] Kapitel 2.2.8).

Technologie	Mehrfaches Zugriffsschema	Datenübertragungsrate	Anwendungsfall
CAN	Prioritätsbasierte Nachrichten	500 Kbit/s	Robuste ECU Steuerung
LIN	Master-Slave und Schedule-Tabellen	19,2 Kbit/s	Niedrige Kosten Steuerung
MOST	Prioritätsbasiert, TDMA, Token	$<$ 25, 50, 150 Mbit/s	Komplex hochqualitatives Audio
FlexRay	(Flexibles) TDMA	$<$ 10 Mbit/s	Echtzeitsteuerung, X-by-Wire
Pixel Links	Keine, nur 2 Kommunikationspartner	Bis zu 3 Gbit/s unidirektional	Unkomprimiertes Video
Consumer Links	Keine, nur 2 Kommunikationspartner	Bis zu 5 Gbit/s	Integration Endgeräte der Kunden
Automotive Ethernet	Switched (geschaltet) für jede Verbindung, Warteschlangen	100, 1000 Mbit/s pro Verbindung und Richtung	Hohe Datenraten, Anwendungsfall unabhängige Pakete

Abbildung 2.1: Vergleich zwischen Ethernet und existierenden Netzwerktechnologien im Automobil (nach Quelle: [14] Tabelle 2.7)

BMW verbaut seit 2008 in jedem Modell Ethernet-Gateways, um die Dauer der Installation von Softwareupdates zu reduzieren ([14] Kapitel 3). Das SEIS-Forschungsprojekt hat 2009 bis 2012 mit einem Budget von 18 Millionen Euro die Einführung IP-basierter Kommunikationslösungen im Automobil erforscht. Unter den vielen Firmen im Projektkonsortium waren unter anderem Audi, BMW, Continental, Daimler, Bosch, VW und Alcatel-Lucent ([18] Kapitel 2.2). Durch den Ethernet-Standard 100BASE-T1 [12] können mit nur einem verdrillten Kupfer-Adernpaar 100 Mbit/s übertragen werden, wodurch sich Ethernet im Automobil als kostengünstige Alternative zu MOST immer mehr durchsetzt und die Hersteller durch das geringere Gewicht der Verkabelung bei der Einhaltung von Emissionsgrenzen unterstützt ([14] Kapitel 3.3). Diesen Standard hat BMW erstmals 2013 in seinen Autos eingesetzt und sich bis 2015 vorgenommen, die komplette Infotainment-Domäne sowie weite Teile der Fahrerassistenzsysteme auf diese Technologie zu migrieren ([14] Kapitel 3.4.3). Die Automobilindustrie ist derzeit neben der Flugzeugindustrie eine treibende Kraft in der IEEE TSN-Gruppe, was das Interesse der Hersteller am zukünftigen Einsatz von TSN im Automobil weiter bestärkt ([18] Kapitel 2.1.3). Eine noch offene Problemstellung für den Einsatz von TSN ist die Konfiguration, an welcher zur Zeit aktiv im Standard IEEE 802.1Qcc [10] gearbeitet wird (siehe Kapitel 2.2.4). Zusätzlich ist es wünschenswert die Vorteile von SDN in Automobilnetzen zu nutzen, um durch die Möglichkeit Netzwerkflüsse dynamisch umzuleiten und sehr spezifisch zu

erlauben und zu blockieren die Sicherheit zu erhöhen und durch die Flexibilität kostengünstiger neue Funktionen zu implementieren. Das Problem der Zusammenführung beider Technologien als TSSDN mit einer zentralen Konfiguration eines solchen Netzwerkes wird in dieser Arbeit gelöst.

2.2 Time-Sensitive Networking

TSN bezeichnet eine Reihe von Ethernet-Substandards, welche ihre Ursprünge im Audio- und Videobereich haben. Das Einsatzgebiet hat sich immer mehr auch in den Automobilbereich und andere Industriebereiche erweitert. Ziel dieser Netzwerktechnik ist es, verlässliche Aussagen bezüglich des Zeitverhaltens des Netzwerkes geben zu können. Dabei sind die Begriffe Latenz und Jitter ausschlaggebend und wie folgt definiert:

- **Latenz:** Die Ende-zu-Ende-Latenz beschreibt die Laufzeit einer Nachricht vom Sender zum Empfänger. Im Automobilbereich werden oftmals unidirektionale oder asymmetrische Verbindungen eingesetzt, weshalb die Ende-zu-Ende-Latenz dort die wichtigste Art der Latenz ist. Es gibt auch die Roundtrip-Latenz, welche allerdings bei Client-Server-Anwendungen die relevante Größe ist [18] und im Folgenden nicht näher betrachtet wird. Die Latenz wird nachfolgend als Synonym für die Ende-zu-Ende-Latenz verwendet.
- **Jitter:** Der absolute Jitter ist die Varianz zwischen der minimalen und maximalen Latenz über alle Nachrichten hinweg. Der relative Jitter ist die Differenz der Latenz über zwei Nachrichten. In Echtzeitsystemen ist der absolute Jitter relevant [18] und wird im Folgenden lediglich Jitter genannt.

Dieser Abschnitt gibt einen Überblick über IEEE 802.1Q-2018, welcher die bisher einzeln gepflegten TSN Substandards enthält, und vergleicht diesen zu dem herkömmlichen IEEE 802.3 Ethernet. Außerdem wird auf das zeitliche Planen (Scheduling) von Ethernet-Paketen in TSN-Netzen eingegangen.

2.2.1 Prioritäten

Standard Ethernet-Switches nach dem Standard IEEE 802.3 [11] leiten die Ethernet-Pakete gleichberechtigt weiter. Dies bedeutet, dass ausgehende Pakete in einer FIFO-Warteschlange gepuffert werden, bevor sie auf dem Port ausgesandt werden (siehe Abbil-

dung 2.2). In dieser FIFO-Warteschlange können bei einem Stau auf dem ausgehenden Port eine geringe Anzahl an Paketen gepuffert werden, ehe neu hinzukommende Pakete verworfen werden. Es gibt also keinen Schutz davor, dass durch ein hohes Datenaufkommen unwichtiger Kommunikationspartner die Warteschlange gefüllt ist und somit andere Kommunikation mit harten Echtzeitanforderungen verzögert oder ganz verworfen wird.

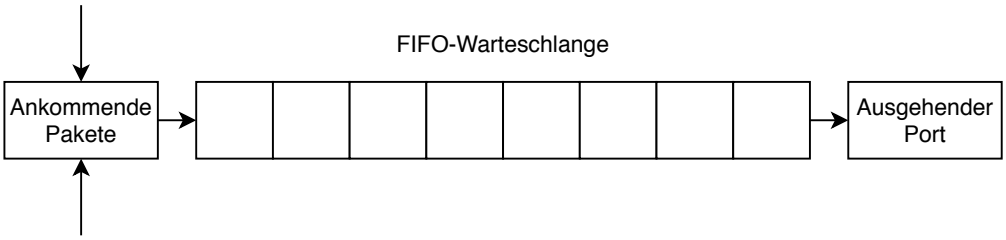


Abbildung 2.2: Aufbau eines 802.3-Switches

In dem Standard IEEE 802.1Q [9] wurde daher bereits 1998 die Struktur eines Standard-Ethernet-Paketes mit einem zusätzlichen 802.1Q Header um 4 Byte erweitert, damit Prioritäten unterstützt werden können (siehe Abbildung 2.3).

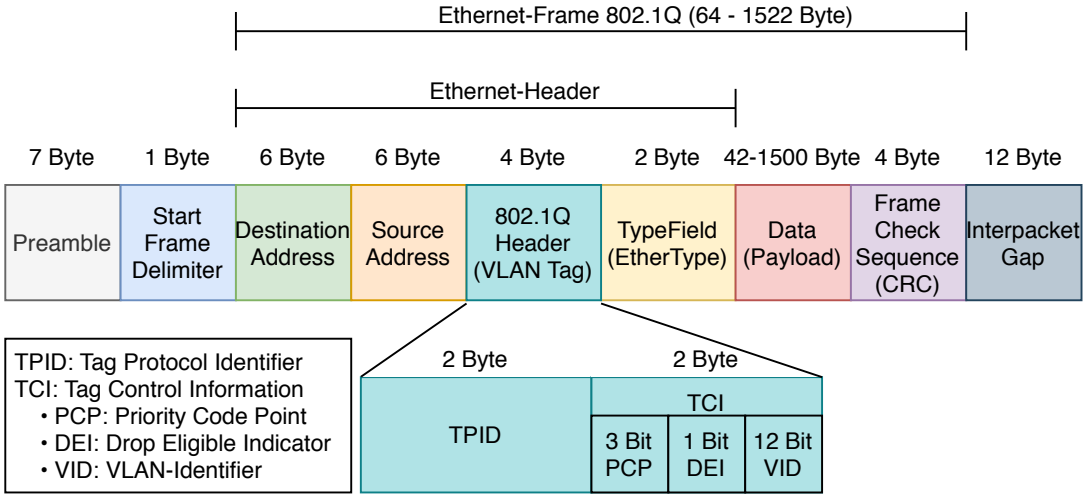


Abbildung 2.3: Struktur eines Ethernet-Paketes nach IEEE 802.1Q

Durch diese Erweiterung ist es neben Priorisierung der Pakete außerdem möglich, VLANs zu erstellen. Der 802.1Q Header setzt sich aus dem Tag Protocol Identifier (TPID) und

der Tag Control Information (TCI) zusammen. Für die Priorität der Netzwerkkommunikation sind nur folgende Subfelder der TCI zuständig:

- PCP: Das Priority Code Point Feld ist 3 Bit groß und erlaubt die Zuweisung des Ethernet-Paketes zu einer Prioritätsklasse von 0 bis 7. Je höher die Ziffer, desto stärker ist die Priorität.
- DEI: Der Drop Eligible Indicator ist 1 Bit groß und gibt an, ob das Ethernet-Paket im Fall eines Staus verworfen werden darf. Dieser Indikator kann entweder allein oder zusammen mit dem PCP-Feld verwendet werden.

Netzwerkkommunikation mit harten Echtzeitanforderungen kann nun eine höhere Priorität signalisieren und sollte somit eine gewisse Dienstgüte vom Switch zugesichert bekommen. Die Ausprägung und Logik dieses Quality of Service genannten Verfahrens kann je nach Implementierung oder Konfiguration des jeweils eingesetzten Switches unterschiedlich sein und unter anderem strikt prioritätsbasiert, gewichtet, oder bandbreitenbasiert arbeiten. Um die Prioritätsfunktion von 802.1Q umsetzen zu können, muss der 802.3-Switch (siehe Abbildung 2.2) erweitert werden. Der Aufbau eines 802.1Q-Switches ist in Abbildung 2.4 dargestellt.

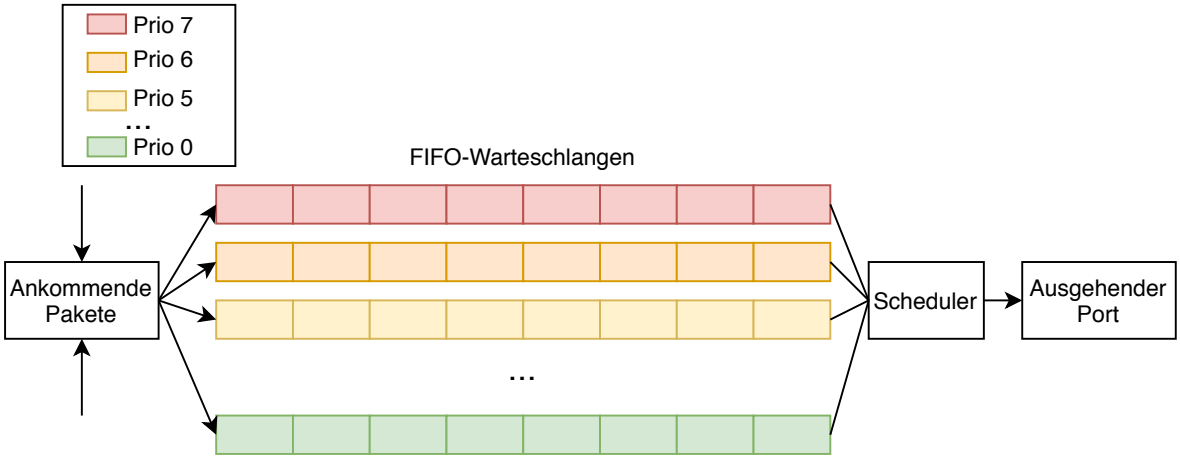


Abbildung 2.4: Aufbau eines 802.1Q-Switches

Ein Planer (Scheduler) entscheidet anhand der Prioritätsklasse und der zuvor genannten Logik, welche Pakete auf dem ausgehenden Port versandt werden. Pakete derselben Prioritätsklasse werden weiterhin gleichberechtigt behandelt. Neben dem Vorteil dieses

Prioritätskonzeptes gibt es allerdings den Nachteil, dass Kommunikation einer niedrigen Priorität verzögert oder komplett verdrängt werden kann, wenn die höhere Priorität die Verbindung auslastet. Da es nicht erlaubt ist einen Ethernet-Frame während der Übertragung zu unterbrechen, kann es auch vorkommen, dass große niedriger priorisierte Pakete die höher priorisierten Pakete blockieren. Durch ein eigenes Scheduling kann dieses Problem gelöst werden, welches im nächsten Unterkapitel 2.2.2 erklärt wird.

2.2.2 Scheduling gemäß IEEE 802.1Qbv-2015

Das Grundprinzip von TSN liegt in der zeitgesteuerten (engl.: time-triggered, TT) Kommunikation, welche ursprünglich in IEEE 802.1Qbv-2015 [8] standardisiert und mittlerweile in IEEE 802.1Q-2018 integriert wurde. Dadurch kann eine minimale Latenz von nur 1 µs für zeitkritische Applikationen bei entsprechender Konfiguration garantiert werden [14]. Für den Datenverkehr im Netzwerk werden wieder Prioritäten vergeben und innerhalb einer Priorität erfolgt eine Gleichberechtigung. Allerdings wurde jede Prioritätsklasse in einem 802.1Qbv-Switch um ein sogenanntes “Gate“ und eine Transmission Selection erweitert (siehe Abbildung 2.5).

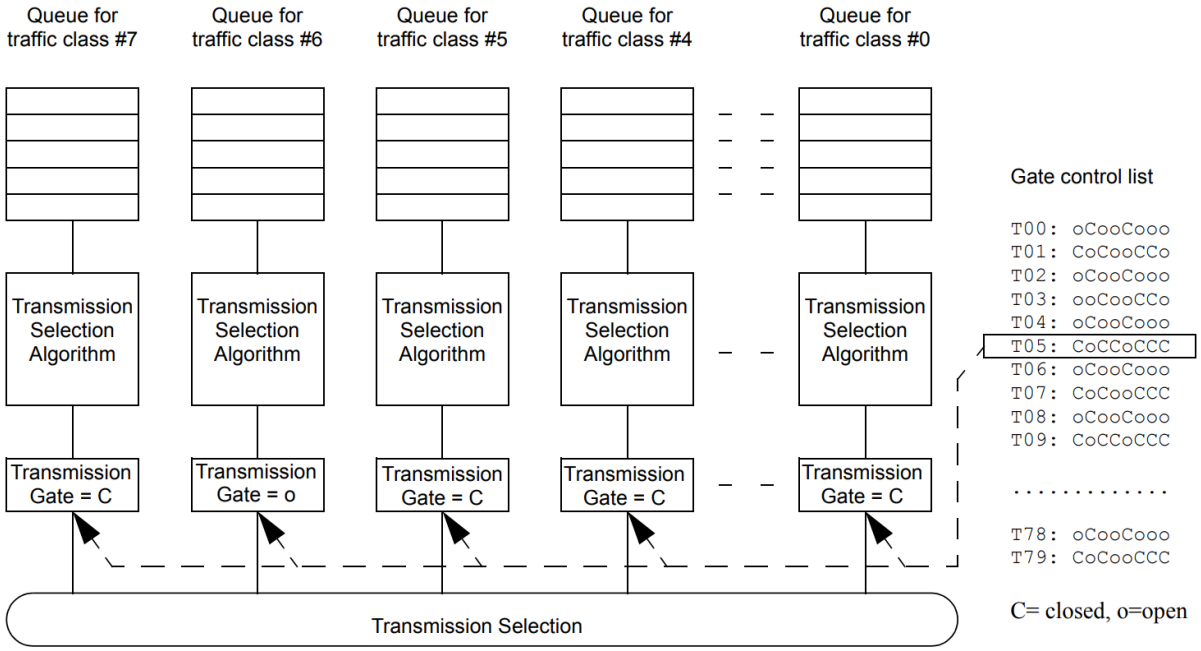


Abbildung 2.5: Aufbau eines 802.1Qbv-Switches (Quelle: [9] Abbildung 8-14)

In jeder FIFO-Warteschlange einer Prioritätsklasse wendet jeweils ein Shaper einen Transmission Selection Algorithmus ähnlich zur Logik des Schedulers in 802.1Q an, um ein weiterzuleitendes Paket auszuwählen. Folgende Algorithmen stehen gemäß 802.1Qbv [9] zur Verfügung:

- Strikt prioritätsbasiert
- Credit-based shaper (CBS)
- Enhanced Transmission Selection (ETS)
- Eigene Herstellerimplementierung

Der in dieser Arbeit eingesetzte Switch unterstützt nur die strikt prioritätsbasierte Auswahl, sodass hohe Prioritäten die niedrigeren Prioritäten vollständig verdrängen. Durch den Einsatz von CBS kann dieses Verhungern vermieden werden.
Jeder FIFO-Warteschlange ist daraufhin ein eigenes Gate angehängt, welches für eine bestimmte Dauer entweder den Zustand open oder closed annimmt. Somit kann eine genaue Zeitsteuerung des Datenverkehrs erfolgen. Der Zustand der Gates wird über eine Gate Control List (GCL) gesteuert. Nur wenn ein Gate offen ist können ausgehende Pakete weiter zum Time Aware Shaper (TAS) versandt werden, welcher die Transmission Selection durchführt. Sind mehrere Gates zur gleichen Zeit geöffnet, wird durch den TAS entschieden, welches der möglichen Pakete tatsächlich weitergeleitet und somit letztlich über den Switch-Port ausgesandt wird. Ist das Gate einer Prioritätsklasse geschlossen, über welche neue Pakete versendet werden sollen, füllt sich die FIFO-Warteschlange dieser Prioritätsklasse bis sie ihren maximalen Füllstand erreicht hat und weitere Pakete verwirft. Durch das Öffnen des Gates wird die FIFO-Warteschlange der Prioritätsklasse wieder zur Übertragung freigegeben. Die GCL enthält pro ausgehendem Port eine Liste von Einträgen, welche den Netzwerk-Zeitplan (Schedule) repräsentieren. Ein Eintrag besteht aus einer binär interpretierten dezimalen Zustandszahl, um die Gatezustände anzugeben, sowie der Dauer dieses Zustandes in Nanosekunden. Die Zustandszahl hat den Wertebereich $0 \leq n \leq 255$. Der Zustand closed (C) eines Gates ist als 0 codiert und der Zustand open (○) ist als 1 codiert. Da 802.1Q genau 8 Prioritätsklassen von 0 bis 7 definiert, sind pro Port die Gates 0 bis 7 vorhanden.

Die Gatenummer entspricht der Bitposition von der Zustandszahl. So wird zum Beispiel Gate 0 über das Least-Significant-Bit 2^0 gesteuert. Sollen etwa die Gates 5, 3 und 0 geöffnet werden, muss folgende Zustandszahl angegeben werden:

$$\begin{aligned}
\text{Gate} &= 7 \quad 6 \quad 5 \quad 4 \quad 3 \quad 2 \quad 1 \quad 0 \\
\text{Zustand} &= \text{C} \quad \text{C} \quad \text{o} \quad \text{C} \quad \text{o} \quad \text{C} \quad \text{C} \quad \text{o} \\
&= 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \\
&= 0 + 0 + 2^5 + 0 + 2^3 + 0 + 0 + 2^0 \\
&= \underline{41}
\end{aligned} \tag{2.1}$$

Der nächste Eintrag besteht wieder aus den neuen anzunehmenden Gatezuständen sowie der Dauer. Sind keine weiteren Einträge vorhanden, beginnt ein neuer Zyklus, indem der Schedule erneut von vorn mit dem ersten Eintrag beginnt.
Durch diesen Mechanismus können Garantien für Latenzen und Jitter im µs-Bereich gegeben werden, sofern der Zeitschlitz getroffen wird. Der TT-Traffic hat harte Anforderungen bezüglich seiner Bandbreite und Latenz. Daraus können statische Ablaufpläne mit Dauer und Frequenz berechnet werden, um das bzw. die Gates dieses zeitkritischen Datenverkehrs zu öffnen. Die übrige freie Zeit wird für den restlichen Datenverkehr reserviert. Durch die Aufteilung der Restkapazität für niedrigere Prioritäten werden diese nun nicht komplett von den höheren Prioritäten verdrängt und durch die garantierten und wohldefinierten Übertragungszeiten des hoch priorisierten Datenverkehrs sind Dienste mit harten Echtzeitanforderungen gesichert.
Allerdings besteht immer noch das in Kapitel 2.2.1 vorgestellte Problem der Verzögerung von höher priorisiertem Datenverkehr durch nicht unterbrechbaren, niedriger priorisierten Datenverkehr. Abbildung 2.6 stellt dieses Problem dar:

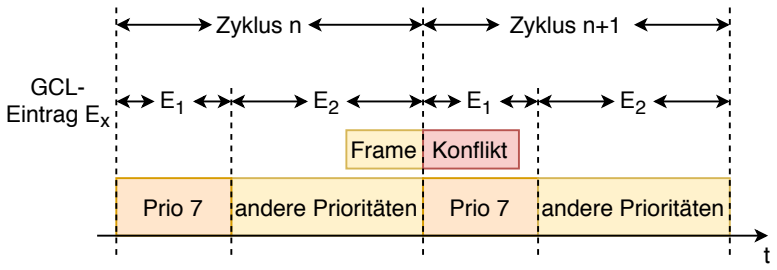


Abbildung 2.6: Niedrig priorisiertes Paket kann nicht unterbrochen werden und blockiert den hoch priorisierten Zeitschlitz des nächsten Zyklus (Quelle: [21])

Um das Problem zu lösen, wird vor dem TT-Traffic in der GCL pro Port ein Schutzband (Guard Band, GB) [21] eingeführt. Dieser Eintrag schließt alle Gates für die Dauer, die benötigt wird, um ein maximales 802.1Q Ethernet-Paket zu übertragen, damit ein letztes noch angefangenes Ethernet-Paket des niedrig priorisierten Zeitschlitzes vollständig übertragen werden kann und somit der TT-Traffic nicht verzögert wird. Dadurch wird die Datenrate zwar geringfügig gemindert, dafür wird aber ein deterministisches Verhalten des Netzwerkes gewonnen. Dies zeigt auch ein Grundproblem beim Einsatz von Scheduling, denn durch die harte Aufteilung und Reservierung der verfügbaren Zeit auf die Prioritätsklassen kommt es bei einer Fehlplanung des Schedules oder verpassen der Zeitschlitze zu einer geringeren Bandbreitenausnutzung. Das Verfahren des Schutzbandes ist in Abbildung 2.7 dargestellt.

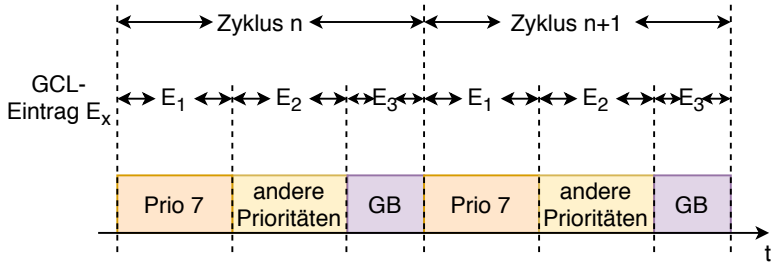


Abbildung 2.7: Konflikt wurde durch den Einsatz des GB gelöst (Quelle: [21])

Einen kompletten Schedule zu berechnen und in die GCL einzutragen wird GCL-Synthese genannt. Es gibt mehrere Ansätze und Algorithmen, um einen optimierten und gültigen Schedule zu berechnen. Diese reichen vom Einsatz ganzzahliger linearer Optimierung [16] über "Satisfiability Modulo Theories" [16] bis hin zu Heuristiken [16] oder "Array Theory Encoding" [17]. Allerdings ist die Berechnung eines vollständigen Schedules über das gesamte Netzwerk ein mathematisch komplexes und aufwendiges (NP-vollständiges [18] Kapitel 3.4.1) Problem [16]. Daher wird die GCL-Synthese üblicherweise im Vorfeld und nicht im laufenden Betrieb durchgeführt. Diese Thematik der Optimierung hat den Umfang einer eigenen Arbeit, daher wurde im Rahmen dieser Arbeit nur ein rudimentärer Algorithmus in Kapitel 4.1 erarbeitet, welcher gültige Schedules berechnet, aber nicht das optimale Ergebnis liefert.

2.2.3 Frame Preemption durch IEEE 802.1Qbu-2016

Das Problem von der Verzögerung hoch priorisierter Nachrichten durch nicht unterbrechbare, niedriger priorisierte Nachrichten kann zusätzlich durch IEEE 802.1Qbu-2016 [7] Frame Preemption gelöst werden. Dadurch kann ein Ethernet-Frame bei der Übertragung unterbrochen und später weiter übertragen werden. Infolgedessen muss das in Kapitel 2.2.2 eingeführte Schutzband nicht mehr die Dauer der Übertragung einer größtmöglichen Nachricht betragen, sondern kann auf die Zeit reduziert werden, die benötigt wird, einen Ethernet-Frame zu unterbrechen ([18] Kapitel 2.1.3).

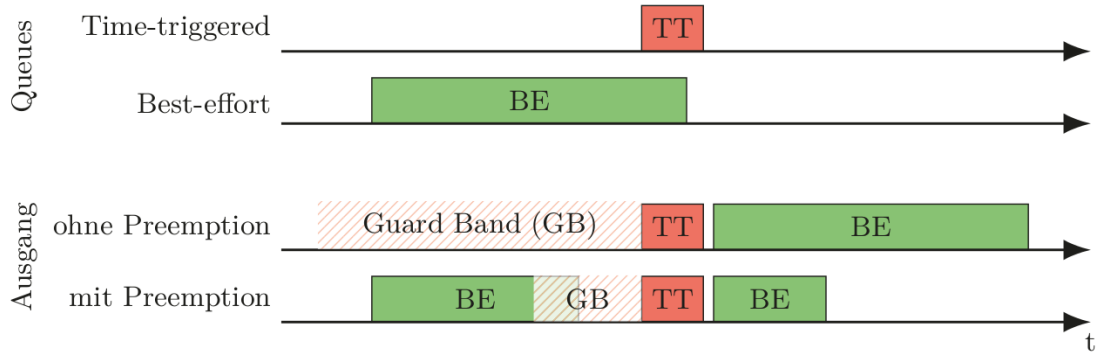


Abbildung 2.8: Verkürzung des Guard Band durch Preemption (Quelle: [18] Abb.: 2.12)

In Abbildung 2.8 ist dargestellt, wie durch den Einsatz von Frame Preemption die gleiche Datenmenge in kürzerer Zeit übertragen werden kann und somit eine bessere Ausnutzung der Bandbreite erreicht wird.

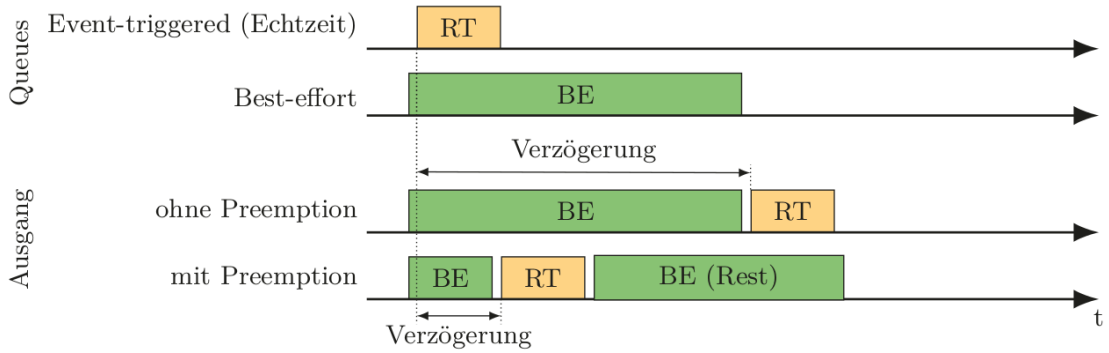


Abbildung 2.9: Verkürzung Latenz und Jitter durch Preemption (Quelle: [18] Abb.: 2.13)

Es wird zudem die Latenz und der Jitter von hoch priorisierten asynchronen Nachrichten reduziert (siehe Abbildung 2.9 RT), da niedriger priorisierter Datenverkehr (BE) während

der Übertragung nach 64 Byte unterbrochen werden darf. Die Beschränkung eine Fragmentierung der Frames auf mindestens 64 Byte große Fragmente ist vorhanden, um die Kompatibilität mit Standard Ethernet 802.3 [11] aufrechtzuerhalten. Durch das benötigte Interpacket Gap (IPG) nach jedem Ethernet-Paket beziehungsweise Fragment dauert die Übertragung mit jeder Unterbrechung allerdings länger und die Datenübertragungsrate wird somit wieder gemindert, da während des IPG keine nutzbaren Informationen übertragen werden können. Der in dieser Arbeit verwendete Switch unterstützt Frame Preemption nicht, sodass diese Funktion nicht evaluiert werden kann.

2.2.4 Zentrale Konfiguration nach IEEE 802.1Qcc-2018

Ein Standard zur Konfiguration von TSN, der zurzeit entwickelt wird, ist IEEE 802.1Qcc [10]. Sogenannte "Talker"- und "Listener"-Hosts teilen dem Switch über ein User/Network Interface (UNI) mit, dass sie einen Kommunikationsstrom aufbauen möchten, und übergeben auch die Informationen welche Anforderungen ihre Daten bezüglich der Übertragung haben. Daraufhin sollen sich alle TSN-Switche in der Topologie passend umkonfigurieren oder ein Scheitern signalisieren ([10] Kapitel 46.1.1). Es werden drei Modelle für die Verteilung der Konfiguration unterstützt.

- **Zentralisiertes Modell**

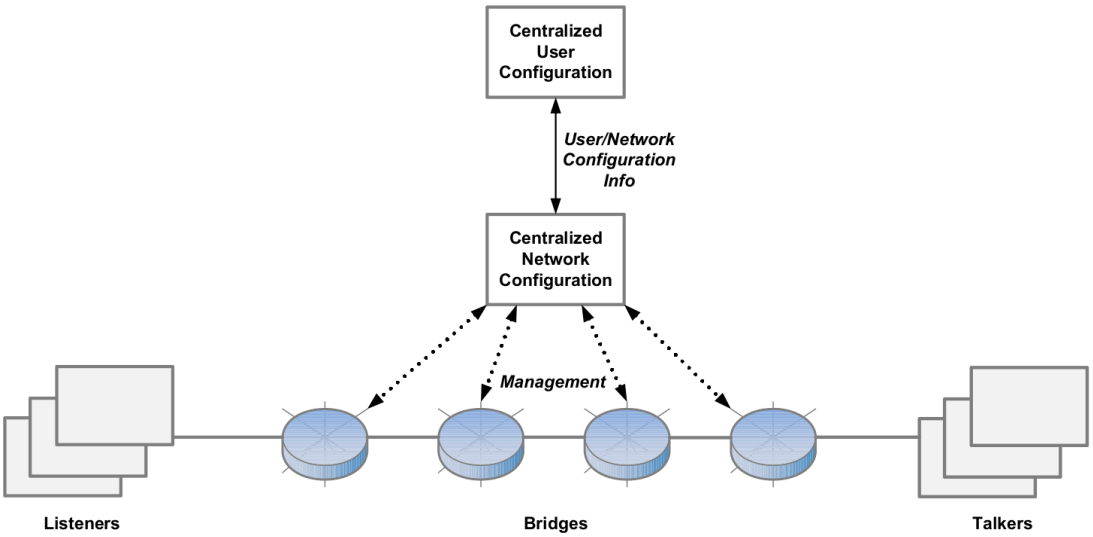


Abbildung 2.10: Zentralisiertes Konfigurationsmodell (Quelle: [10] Abbildung 46-3)

- **Zentralisiert-verteiltes Modell**

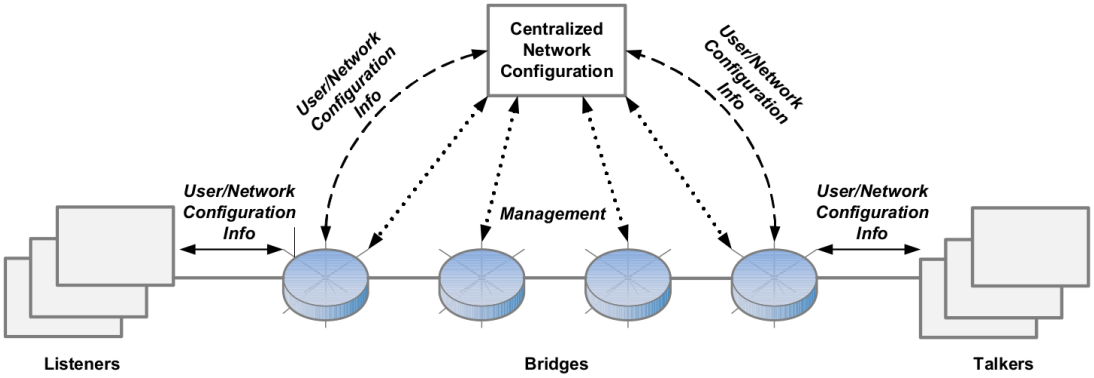


Abbildung 2.11: Zentralisiert-verteiltes Konfigurationsmodell (Quelle: [10] Abbildung 46-2)

- **Verteiltes Modell**

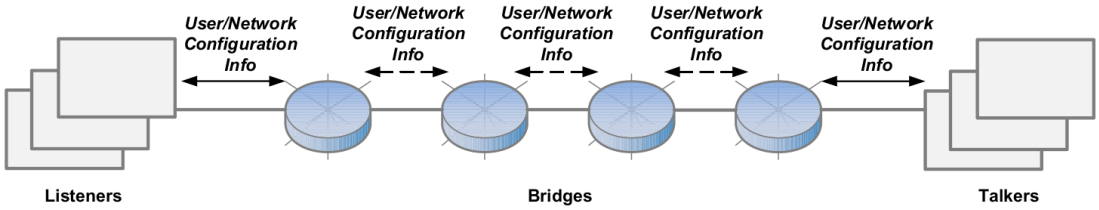


Abbildung 2.12: Verteiltes Konfigurationsmodell (Quelle: [10] Abbildung 46-1)

Da die Entwicklungsphase des Standards noch nicht abgeschlossen, kann auf ihn in dieser Bachelorarbeit noch nicht zurückgegriffen werden. Allerdings wird in dieser Arbeit das zentralisierte Modell zur Verteilung von Konfigurationsänderungen mithilfe von OpenFlow und NETCONF umgesetzt.

2.3 Software-Defined Networking

In klassischen Switches sind die Kontrollebene und Datenebene im gleichen Gerät verortet und nicht programmierbar. Dies bedeutet, dass ein Switch selbst entscheidet über welche(n) seiner Ports ein Paket weitergeleitet wird. SDN trennt diese Verankerung und überlässt dem Switch nur noch das Weiterleiten der Pakete über seine Datenebene. Ein SDN-Controller auf einem gesonderten Host übernimmt die Logik der Kontrollebene

von jedem Switch des gesamten Netzwerkes (siehe Abbildung 2.13). Der SDN-Controller besitzt eine zentralisierte Sicht über das Netzwerk und kann dadurch erweiterte Entscheidungen treffen, über welche Switches und deren Ports die Pakete weitergeleitet werden sollen.

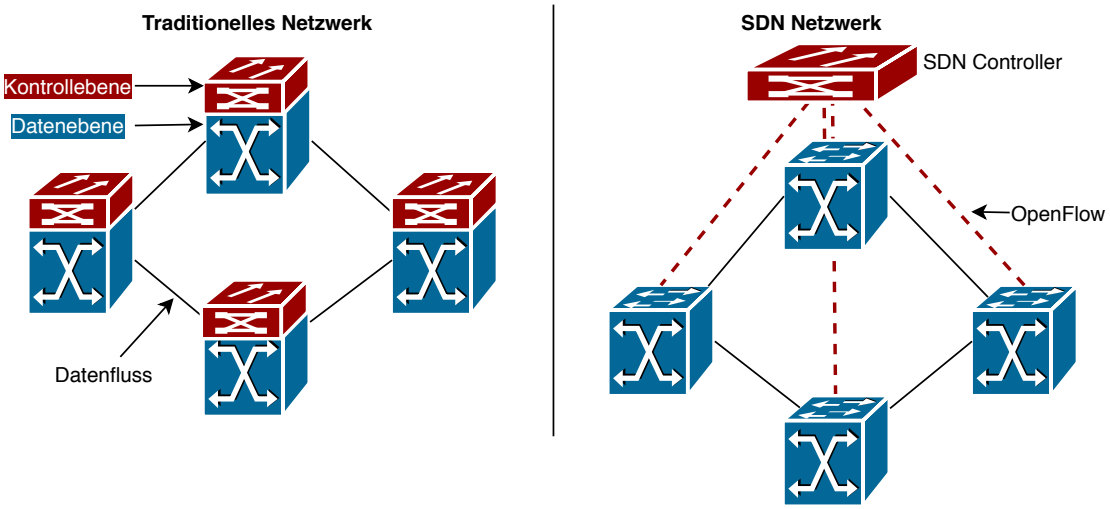


Abbildung 2.13: Vergleich traditionelles Netzwerk gegen SDN (Quelle: [13])

Durch das OpenFlow-Protokoll [5] kommunizieren der SDN-Controller und der SDN-Switch miteinander. Ein SDN-Switch schaut in seiner Weiterleitungstabelle (Flow Table) bei einem ankommenden Paket nach, welche Aktionen er bei dieser Art von Paket ausführen soll. Die Flow Table entspricht somit der Kontrollebene von klassischen Switches. Findet der Switch keinen Eintrag in der initial leeren Flow Table sendet er alle Informationen dieses Paketes an den SDN-Controller. Der SDN-Controller teilt dem Switch daraufhin eine Weiterleitungsregel (Flow Rule) mit, sofern dieses Paket erlaubt sein soll und weitergeleitet werden darf. Die Flow Rule gibt an, welche Eigenschaften und Werte vom Paket durch den Switch auf Übereinstimmung geprüft werden müssen (Match) und welche Aktionen er daraufhin für diese Art von Paket ausführen soll. Ein Match ist bei SDN im Vergleich zu einem Standard-Ethernet-Switch nicht auf die Ziel-MAC-Adresse beschränkt. Stattdessen kann auf vielen Informationen der OSI-Schichten 2 bis 4 ein Match erfolgen.
Vom Switch werden zudem periodisch Statistiken an den SDN-Controller geschickt. Sie enthalten Informationen wie oft eine Flow Rule zutrat oder ob es Topologieänderungen gab. Dadurch kann der SDN Controller mit der Veränderung einer Flow Rule reagieren.

Es kann somit z. B. eine Flow Rule erstellt werden, welche nur eine bestimmte Anzahl an Pings erlaubt, ehe die Kommunikation weiterführend blockiert wird. Bei 802.1Qcc werden dagegen keine Statistiken vom Switch zum Centralized Network Configuration Host übermittelt. Die mit 802.1Qcc eingeführte zentrale Netzwerkkonfiguration ist also nicht reaktionär ausgelegt wie OpenFlow. Stattdessen sind bei 802.1Qcc wieder große Teile der Kontrollebene im Switch selber verortet und das Protokoll ist somit eher für statisches Konfigurations-Management geeignet. Dies ist auch ein Grund, warum in dieser Bachelorarbeit die Netzwerkfluss-Konfiguration mit OpenFlow statt 802.1Qcc verfolgt wird.
Im Vergleich zu einem Standard-Ethernet-Switch sind die verfügbaren Aktionen ebenfalls erweitert und nicht nur auf die Angabe eines ausgehenden Ports beschränkt. So kann neben der Output Action, die das Paket über den angegebenen Port versendet, mit der Set-Queue Action das Paket einer Prioritätsklasse nach 802.1Q zugewiesen werden. Somit unterliegt dieses Paket dem konfigurierten Schedule des Switches nach 802.1Qbv. Es gibt noch weitere Aktionen, die in dieser Arbeit aber nicht verwendet wurden und somit nicht betrachtet werden.
Die Flow Rule bestehend aus Match und Aktionen speichert der Switch in seiner Flow Table zwischen, um zukünftig nicht mehr den SDN-Controller für diese Art von Datenverkehr anfragen zu müssen. Liefert der SDN-Controller keine passende Regel an den Switch, wird das Paket verworfen.
Mit einer selbst programmierten Logik in Form von “SDN Controller Applications“ werden die Flow Rules mit Inhalt befüllt und somit die Datenflüsse im Netzwerk festgelegt.

2.4 NETCONF

Das Network Configuration Protocol (NETCONF) [2] wurde von der IETF als Management-Protokoll für die Konfiguration von Netzwerkgeräten entwickelt. NETCONF löst das Simple Network Management Protocol (SNMP) oder die Kommandozeile als bisherige Lösungen zur Konfiguration ab. Es besteht aus vier Protokollebenen (siehe Tabelle 2.1).

Ebene	Beispiel
Inhalt	Konfigurationsdaten, Nachrichtendaten
Operationen	<get-config>, <edit-config>, <delete-config>, ...
Nachrichten	<rpc>, <rpc-reply>, <notification>
Transport	SSH, TLS, HTTP/TLS, ...

Tabelle 2.1: NETCONF Protokollebenen

Über Remote Procedure Calls (RPCs) werden Operationen von einem NETCONF-Client (Host) auf einem NETCONF-Server (dem Switch) angewiesen. Dadurch können Konfigurationsdaten installiert, geändert und gelöscht werden sowie andere Operationen ausgeführt werden. Nach der Ausführung wird das Ergebnis des RPCs dem Client als Reply mitgeteilt. Alle Protokollnachrichten und Daten verwenden ein XML-Format und können über verschiedene Transportprotokolle übermittelt werden. Die aktuelle Konfiguration des Switches befindet sich "Datastore: running" und Änderungen an diesem werden sofort umgesetzt. Es kann optional weitere Datastores wie "startup" oder "candidate" zu Management-Zwecken geben. Eine Konfigurationsänderung wird als Transaktion behandelt, sodass bei einem Fehler in der Konfiguration oder dem Transport die Transaktion fehlschlägt und der vorher konsistente Zustand erhalten bleibt. Das Schema des XML-Datenmodells ist durch YANG-Modelle im YANG-Namensraum [1] definiert. Abbildung 2.14 zeigt das YANG-Modell für die TAS-Komponente und dessen GCL von dem in dieser Arbeit verwendeten Switch.

Schema	Type	Flags	Opts	Status	Path
TNsysrepo	module				
▶ TNtas	container		config	current	/tn:TNtas
▶ ports[id]	list		config	current	/tn:TNtas/tn:ports
id	leaf	uint32	config	current	/tn:TNtas/tn:ports/tn:id
▶ GCL[id]	list		config	current	/tn:TNtas/tn:ports/tn:GCL
id	leaf	uint32	config	current	/tn:TNtas/tn:ports/tn:GCL/tn:id
timeperiod	leaf	uint32	config ?	current	/tn:TNtas/tn:ports/tn:GCL/tn:timeperiod
gatestates	leaf	uint8	config ?	current	/tn:TNtas/tn:ports/tn:GCL/tn:gatestates
admin_base_time	leaf	uint64	config ?	current	/tn:TNtas/tn:ports/tn:admin_base_time
admin_cycle_time_ext	leaf	uint32	config ?	current	/tn:TNtas/tn:ports/tn:admin_cycle_time_ext
gate_enable	leaf	boolean	config ?	current	/tn:TNtas/tn:ports/tn:gate_enable

Abbildung 2.14: YANG-Modell vom TAS und dessen GCL (Quelle: InnoRoute [20])

Durch NETCONF lassen sich auch Flow Rules nach einem definierten YANG-Modell [20] auf dem Switch installieren, sodass nicht zwingend OpenFlow für die Konfiguration der Netzwerkarchitektur oder Topologie verwendet werden muss.

3 Anforderungen und Testfälle

Im Automobilbereich gibt es feste Anforderungen an die Infrastrukturkomponenten. Um bisher etablierte Übertragungstechnologien wie den CAN-Bus ablösen zu können und auch Reserven für zukünftige Entwicklungen, wie etwa dem autonomen Fahren zu schaffen, müssen Richtwerte bezüglich der Bandbreite, Latenz und Jitter eingehalten werden sowie Prioritäten und Domänenbildung unterstützt werden. In diesem Kapitel werden die Anforderungen aus dem Automobilbereich ([18] Kapitel 3.1) vorgestellt und Testfälle erstellt, um zu ermitteln, ob die TSSDN-Lösung diese Anforderungen erfüllt.

3.1 Bandbreitenverteilung, Latenz und Jitter

Je nach Art der Kommunikation im Fahrzeugnetzwerk werden unterschiedliche Bandbreiten benötigt und es herrschen unterschiedlich starke Anforderungen an die Latenz. Im Hinblick auf zukünftige Entwicklungen sind 3 Mbit/s für Steuerdaten und 68 Mbit/s für Radar- sowie LIDAR-Sensoren und Kamerabilder realistische Anforderungswerte ([18] Kapitel 3.1.1). Die restliche Bandbreite steht somit anderen Anwendungsfällen zur Verfügung. Steuerdaten dürfen eine maximale Latenz von 600 µs nicht überschreiten ([18] Kapitel 3.1.2). Sensoren und Kamerabilder dürfen eine maximale Latenz von 33 ms nicht überschreiten. Für die restliche Bandbreite müssen keine Latenzen eingehalten werden, da z. B. Softwareupdates oder Videostreaming als Anwendungsfälle hohe Latenzen durch Puffern ausgleichen können.
Weil die stärkste Anforderung an die Latenz von den Steuerdaten ausgeht, wird nur die Einhaltung dieser Latenz überprüft. Der Jitter sollte den Richtwert von 10 % der Latenz nicht überschreiten ([18] Kapitel 3.1.3). In konkreten Zahlen darf der Jitter bei den Steuerdaten also nicht höher als 60 µs sein. Um zu prüfen, ob diese Bandbreiten, Latenzen und der Jitter trotz Vollauslastung der jeweiligen Kommunikationsarten garantiert werden können, wird eine Testarchitektur erstellt (siehe Abbildung 3.1).

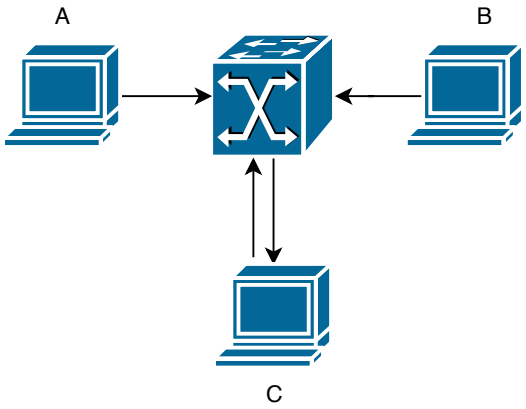


Abbildung 3.1: Testarchitektur für Bandbreitenverteilung, Latenz und Jitter

Durch die Vollauslastung jedes Links wird das Verhalten im schlechtesten Fall mit parallelem Datenverkehr (Cross-Traffic) getestet.

- Host A sendet an Host C UDP-Pakete, welche exemplarisch für die Sensoren und Kamerabilder stehen und 68 Mbit/s erreichen sollen.
- Host C sendet von seinem ersten Netzwerk-Interface UDP-Pakete an sein zweites Netzwerk-Interface, welche für die Steuerdaten stehen und somit eine Latenz von 600 µs nicht überschreiten dürfen. Es wird der gleiche Host zum Senden und Empfangen verwendet, damit eine einheitliche Zeitquelle vorhanden ist und somit die Latenz genau bestimmen zu können. Des Weiteren darf der Jitter maximal 60 µs betragen und es soll eine Datenübertragungsrate von 3 Mbit/s erreicht werden.
- Die restliche Bandbreite wird Host B zur Verfügung gestellt, welcher ebenfalls UDP-Pakete an Host C sendet. Bei dieser Datenübertragung wird nur betrachtet, ob die restliche Bandbreite ausgenutzt wird.

Auf dem Host C, welcher alle drei Kommunikationsdaten empfängt, wird auf beiden Netzwerk-Interfaces der Datenverkehr mitgelesen und gespeichert, um später analysieren zu können, ob die Anforderungen eingehalten wurden. Es werden geeignete Flow Rules und ein geeigneter Schedule erstellt und auf den Switch übertragen.

3.2 Domänenbildung und Kommunikationsflüsse

In einem Auto dürfen nur bestimmte Geräte untereinander kommunizieren. Der CAN-Bus wird daher auf die Domänen: Diagnose, Antrieb, Komfort und Infotainment unterteilt ([18] Kapitel 3.2.1). Je nach der aktuellen Situation eines Autos sollen nur bestimmte Domänen aktiv sein und somit deren Kommunikation erlaubt sein. Die Kommunikation kann über Gateways aber auch über Domänengrenzen hinweg erlaubt sein, z. B. kann das Infotainment-System viele Fahrzeuginformationen der anderen Domänen auf dem Bildschirm anzeigen. Es wird eine neue Testarchitektur erstellt (siehe Abbildung 3.2).

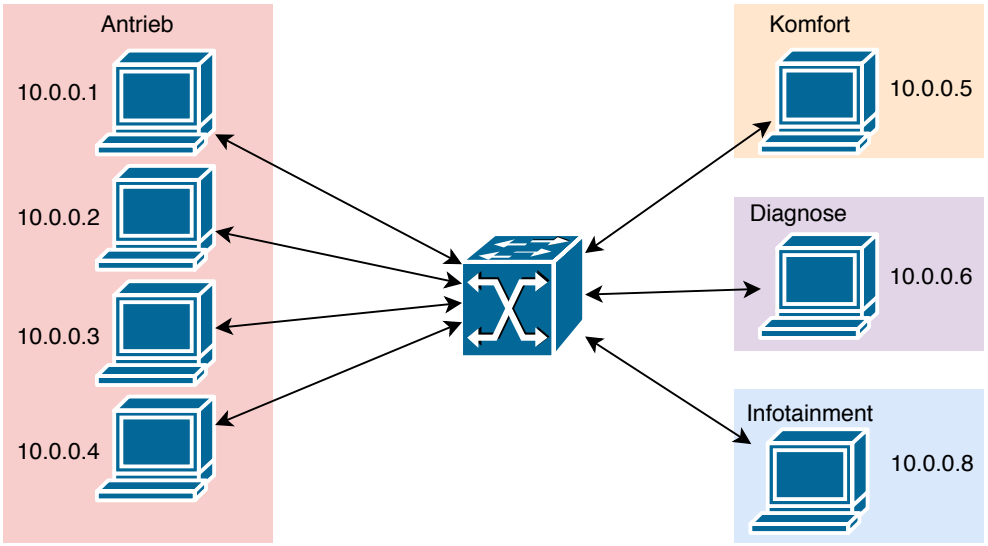


Abbildung 3.2: Testarchitektur für Kommunikationsflüsse mit dem Infotainment-Host

In dieser Topologie dürfen Domänenmitglieder derselben Domäne miteinander kommunizieren, sofern die Domäne aktiv ist. Das Infotainment-System darf je nach einem der zur Verfügung stehenden Modi “Fahren“ und “Update / Diagnose“ mit bestimmten Domänen kommunizieren. Weil die Komfort-Domäne gewöhnlich nur aus Sensor-Aktuator Komponenten auf einem 1-Wire-Bus ohne komplexe Steuergeräte mit Firmwareupdates oder Diagnosefunktionen besteht ([14] Kapitel 2.2.2), ist diese Domäne im Modus “Update / Diagnose“ deaktiviert. Die jeweils erlaubten und verbotenen Kommunikationsflüsse zwischen den Domänen und dem Infotainment-System sind in Tabelle 3.1 dargestellt.

Domäne \ Modus	Fahren	Update / Diagnose
Antrieb	✓	✓
Komfort	✓	✗
Diagnose	✗	✓
Infotainment	✓	✓

Tabelle 3.1: Modi und deren Kommunikationsflüsse mit dem Infotainment-Host

Die Testfälle, welche sich bei jedem Moduswechsel ergeben, lauten:

- Funktioniert die Kommunikation zwischen erlaubten Kommunikationspartnern?
- Ist die nicht erlaubte Kommunikation tatsächlich unterbunden?
- Treten bei einem Wechsel des Modus Paketverluste auf, falls die Kommunikation in beiden Modi erlaubt ist? Ist ein Moduswechsel somit unterbrechungsfrei?
- Wie lange hat es gedauert, bis die Konfigurationsänderungen umgesetzt wurden?

3.3 Prioritäten

Auf dem CAN-Bus werden Prioritäten verwendet, damit sich wichtige Nachrichten in der Warteschlange vor weniger wichtigen Nachrichten einreihen können ([18] Kapitel 3.1.2) und somit eine Latenz von maximal 600 µs erzielt werden kann. Dies muss vor allem auch in Situationen funktionieren, in denen die gesamte Bandbreite von niedriger priorisiertem Datenverkehr ausgenutzt wird. Um die Auswirkungen von Cross-Traffic (CT) auf die Latenz und den Jitter ohne Priorität, mit Priorität und letztlich mit Priorität in Kombination mit einem Schedule zu prüfen, wird wieder die bisherige Testarchitektur aus Kapitel 3.1 genutzt (siehe Abbildung 3.1). Es wird die Latenz von Host C zu Host C gemessen während Cross-Traffic von Host A an Host C verschickt wird. Es wird in folgenden Testfällen immer die Latenz **eines** maximalen Ethernet-Paketes gemessen, welches von Host C an Host C geschickt wird:

- **Ohne CT:** Nur Host C ist angeschlossen. Es wird die Latenz gemessen, welche durch die Datenübertragungsrate, das Betriebssystem-Scheduling, den Netzwerk-Stack und durch den Switch verursacht wird. Der auftretende Jitter sollte sehr gering sein.

- **CT ohne Prio:** Host A und Host C haben die gleiche Priorität 1. Host A erzeugt CT um den Link voll auszulasten. Es wird ein großer Jitter erwartet.
- **CT Q Prio:** Host C hat nun die höhere Priorität 2 und Host A hat weiterhin die Priorität 1. Host A erzeugt CT. Es wird eine geringere maximale Latenz und somit ein geringerer Jitter erwartet.
- **CT Q Prio & GCL:** Host C hat nach wie vor die Priorität 2 und Host A hat immer noch die Priorität 1. Host A erzeugt CT. Die GCL ist so konfiguriert, dass Priorität 1 124 µs geöffnet ist, um ihren maximalen Frame zu senden, dann das Schutzband mit 124 µs geöffnet ist und schließlich die restliche Zeit des 600 µs-Zyklus für Host C mit der Priorität 2 reserviert ist. Es wird eine erneute Verbesserung der Latenz und des Jitters erwartet.

3.4 Leistungs- und Funktionstests Switch

3.4.1 Bandbreitenverteilung

Um neben der Verbesserung der Latenz durch Prioritäten und der GCL auch das Verhalten auf die Bandbreite zu beleuchten und somit das Verhalten des Switches weiterführend zu prüfen, werden weitere Testfälle auf Basis der bisherigen Testarchitektur 3.1 erstellt. In allen Testfällen senden Host A und Host B an Host C ihre Ethernet-Pakete um die Verbindung voll auszulasten. An Host C werden die empfangenen Ethernet-Pakete mitgeschnitten und es wird ausgewertet, wie viel Bandbreite Host A und wie viel Bandbreite Host B durch den Switch zur Verfügung steht.

- **Gleiche Priorität:** Host A und Host B haben die gleiche Priorität 1. Es kann nicht abgeschätzt werden, wie das Ergebnis sein wird, da keine Gleichverteilung oder Ähnliches für gleiche Prioritäten in IEEE 802.3 [11] oder IEEE 802.1Q [9] spezifiziert ist.
- **Höhere Priorität:** Host A hat die Priorität 2 und Host B hat die Priorität 1. Es ist zu erwarten, dass aufgrund des strikten Prioritätskonzeptes fast nur Ethernet-Pakete von Host A empfangen werden.
- **GCL 50/50:** Host A hat immer noch die höhere Priorität 2 und Host B die Priorität 1. Allerdings ist die GCL so konfiguriert, dass Priorität 2 und Priorität 1

im Wechsel jeweils für 10 ms geöffnet sind. Somit stehen die Prioritäten ab sofort synonym für timeslotted groups. Es ist zu erwarten, dass nun eine exakte Gleichverteilung der Datenübertragungsrate vorliegt.

3.4.2 Latenz und Jitter des Switches

In den bisherigen Testfällen wurde die Latenz und der Jitter immer nur über den Host C und somit über die gesamte Übertragung inklusive des Betriebssystem-Schedulings und dem Netzwerk-Stack gemessen. Um genau zu ermitteln, wie hoch der Anteil des Switches an diesen Ergebnissen ist, werden mit einem Oszilloskop auf der physischen Übertragung unmittelbar vor Eintritt eines Ethernet-Paketes in den Switch und nach Austritt aus dem Switch Zeitstempel genommen und aus der Differenz beider Zeitstempel die Latenz berechnet. Die Ethernet-Pakete, welche Host A an Host C schickt, haben dabei eine stetig wachsende Größe der Nutzdaten, um die Cut-Through Funktionalität des Switches zu testen.

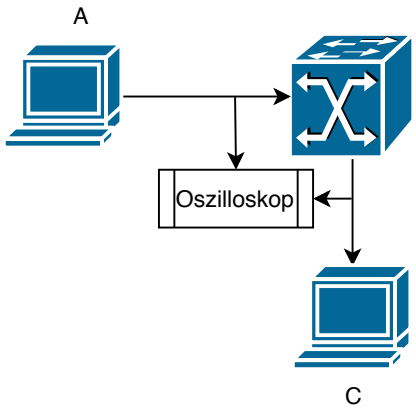


Abbildung 3.3: Testarchitektur für Latenzmessung des Switches mit Oszilloskop

4 Konzept

4.1 Synthese der Gate Control List

In dieser Arbeit wurde ein Algorithmus erarbeitet, um eine GCL-Synthese durchzuführen und somit einen Schedule zu berechnen. Alle Berechnungen sowie der Algorithmus beziehen sich auf eine Datenübertragungsrate von 100 Mbit/s. Für eine erfolgreiche GCL-Synthese ist es erforderlich, die Dauer des Schutzbandes zu kennen. Die Dauer des Schutzbandes ist gleich mit der Übertragungsdauer eines maximalen Ethernet-Paketes, also die Übertragungsdauer von 1542 Byte. Für einen Switch mit 100 Mbit/s Datenübertragungsrate wird die Dauer des Schutzbandes wie folgt berechnet:

$$\frac{1542 \text{ Byte} * 8}{100 \text{ Mbit/s}} = \underline{123360 \text{ ns}} \tag{4.1}$$

In dem erarbeiteten Algorithmus 1 gibt der Anwender für jede zeitkritische Kommunikation die Datenmenge in Bit und Deadline in µs an. Sobald die Eingabe aller zeitkritischen Kommunikationen abgeschlossen ist, beginnt die Berechnung der noch für den nicht zeitkritischen Datenverkehr verbleibenden Restzeit, ohne dass die härteste Deadline des zeitkritischen Datenverkehrs verletzt wird.

Algorithmus 1 GCL-Synthese

```
 1: guardbandNS = 123360    // Dauer des Schutzbandes in ns
 2: repeat
 3:    bits += input()          // Eingabe Datenmenge einer wichtigen Kommunikation
 4:    deadlines.add(input())   // Eingabe dessen Deadline in µs
 5: until input()               // Weitere wichtige Eingaben?
 6: durationNS = bits * 10      // Berechnung Übertragungsdauer in ns
 7: deadline = deadlines.smallest() * 1000          // Härteste Deadline in ns ermitteln
 8: if durationNS+guardbandNS ≥ deadline then
 9:    print(Error: infeasible schedule!)           // Deadline bei 100 Mbit/s unmöglich
10:    exit()                                       // Algorithmus abbrechen
11: end if                          // Nachfolgend jeden Switch-Port mit Schedule befüllen
12: for i = 0 to i < ports do
13:    port[i].GCL[0].timeperiod = durationNS   // Für Dauer wichtige Kommunikation
14:    port[i].GCL[0].gatestates = 2^0 + 2^7    // die Gates 0 und 7 öffnen
15:    port[i].GCL[1].timeperiod = deadline - durationNS - guardbandNS//Restl.Dauer
16:    port[i].GCL[1].gatestates = 2^0+2^1+2^2+2^3+2^4+2^5+2^6   // Restl. Gates öffnen
17:    port[i].GCL[2].timeperiod = guardband                  // Für Dauer Schutzband
18:    port[i].GCL[2].gatestates = 2^0                        // Nur Gate 0 öffnen
19: end for
```

Um die Dauer des übertragenen zeitkritischen Datenverkehrs zu ermitteln, wird dessen Datenmenge in Bits mit 10 multipliziert (Zeile 6), da bei einer Datenübertragungsrate von 100 Mbit/s ein Bit 10 Nanosekunden zur Übertragung benötigt. Daraufhin wird die härteste (früheste) Deadline ermittelt. Diese Deadline wird ebenfalls in Nanosekunden umgerechnet.
Dann wird eine Prüfung durchgeführt, ob die Deadline bei dieser Menge an Daten und deren Übertragungsgeschwindigkeit verletzt werden würde (Zeile 8) oder kein Zeitschlitz mehr für niedriger priorisierten Datenverkehr übrig wäre.
Zuletzt werden die GCL generiert und für jeden Port die gleichen Einträge hinzugefügt (Zeile 12 - 19). In dem ersten Eintrag und somit für die erste Zeitspanne ist nur das Gate des zeitkritischen (hoch priorisierten) Datenverkehrs sowie Gate 0 geöffnet. Laut der Herstellerspezifikation des in dieser Arbeit eingesetzten Switches muss das Gate 0 immer geöffnet sein. Der zweite Eintrag vergibt die Restkapazität für niedriger priorisierten Datenverkehr, indem alle Gates bis auf Gate 7 geöffnet sind. Die Dauer des zweiten Eintrags ergibt sich aus der härtesten Deadline, von welcher aber die verstrichene Zeit der ersten Zeitspanne sowie die Zeit des noch kommenden Schutzbandes abgezogen werden muss (Zeile 15), um die Deadline erfüllen zu können. Der dritte Eintrag ist das Schutzband, währenddessen nur das wie vom Hersteller geforderte Gate 0 geöffnet ist.

4.2 Konfiguration

Die Wahl des SDN-Controllers bestimmt maßgeblich wie der Switch angesprochen wird und somit wie die Lösung zur Konfiguration eines TSSDN-Netzwerkes aufgebaut ist. Ein SDN-Controller ist ONOS (Open Network Operating System), welcher ein Open-Source-Projekt der Linux Foundation ist. Das Ziel dieses Projekts ist ein skalierbares, hochperformantes und hochverfügbares SDN Betriebssystem zu erstellen, welches auch bei Telekommunikationsfirmen produktiv eingesetzt wird. ONOS wurde als SDN-Controller in dieser Arbeit ausgewählt, da er bereits erfolgreich in der CoRe-RG eingesetzt wurde und eine breite Unterstützung von Partnerunternehmen wie unter anderem AT&T, DELL, Telekom, Google, Intel und Samsung hat. Um die SDN Application zur Konfiguration zu schreiben, wird über die "Northbound API" auf Hilfsfunktionen zugegriffen, die z. B. das Erstellen einer Regel (Flow Rule) erlauben. Über die "Southbound API" wird dann das tatsächliche Gerät (Switch) angebunden und per OpenFlow und NETCONF angesprochen. Dabei wird Java als Programmiersprache verwendet, welche ONOS durch eigene bereitgestellte Datentypen und Services erweitert. Ein Architekturüberblick von ONOS ist in Abbildung 4.1 dargestellt.

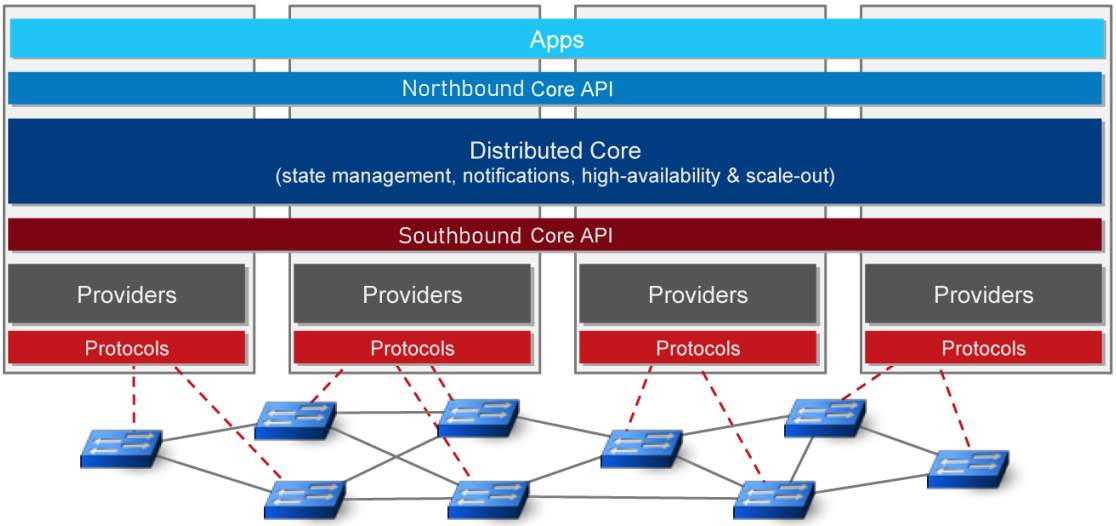


Abbildung 4.1: Architekturüberblick von ONOS (Quelle: [4])

Die Subkomponenten der einzelnen Schichten von ONOS werden in Abbildung 4.2 gezeigt.

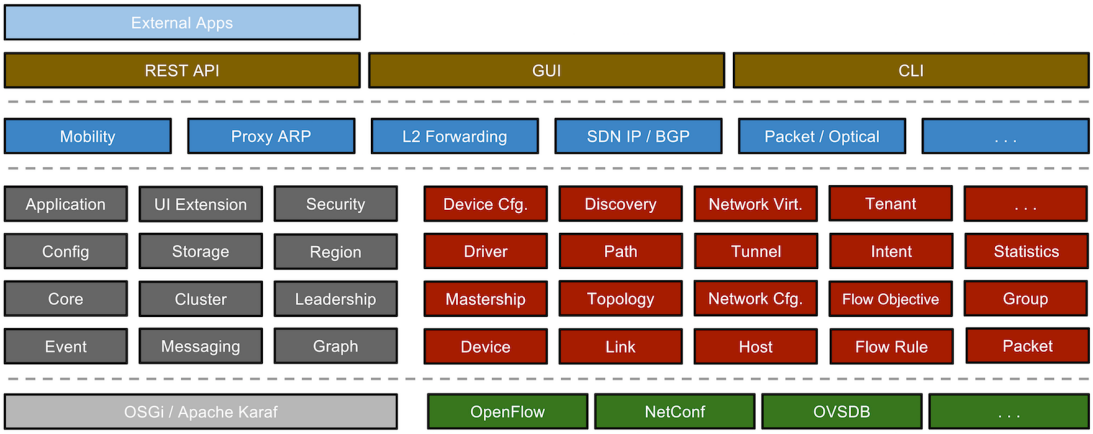


Abbildung 4.2: Subkomponenten der Architektur von ONOS (Quelle: [3])

Sie lassen sich in folgende Subsysteme gruppieren:

- Device Subsystem: Verwaltet die Liste von Infrastrukturgeräten.
- Link Subsystem: Verwaltet die Liste von Infrastrukturverbindungen.
- Host Subsystem: Verwaltet die Liste von Endstationen (Hosts) und deren Standort im Netzwerk.
- Topology Subsystem: Verwaltet zeitgesteuerte Momentaufnahmen von Netzwerkgraphansichten.
- PathService: Berechnet und findet Wege zwischen Infrastrukturgeräten oder zwischen Endstations-Hosts basierend auf der aktuellen Momentaufnahme des Netzwerkgraphen.
- FlowRule Subsystem: Verwaltet die Liste von match/action Flow-Regeln, welche auf einem Infrastrukturgerät installiert sind, und stellt Flow-Metriken bereit.
- Packet Subsystem: Erlaubt es Applikationen auf einkommenden Datenpaketen von Netzwerkgeräten zu horchen und Datenpakete in das Netzwerk an ein oder mehrere Geräte zu senden.

Um die Business-Logik von der Übertragungslogik zu trennen, wird ein Treiber für den Switch geschrieben, welcher somit auch in anderen SDN Applications wiederverwendet werden kann. Als Anforderung muss der Treiber Flow Rules per OpenFlow auf dem Switch installieren können und vorher synthetisierte GCL-Konfigurationsdateien per NETCONF übertragen können. Das UML-Klassendiagramm (siehe Abbildung 4.3) zeigt die API-Methoden des Treibers TNDriver, welche von dem Business-Logik-Teil der SDN Application CarModeSwitcher genutzt werden.

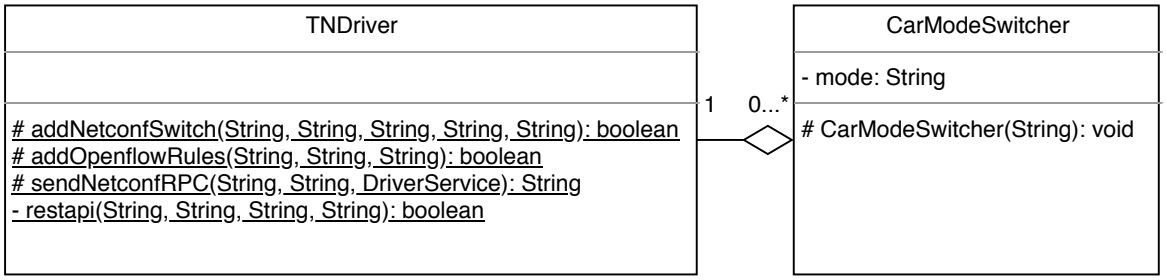


Abbildung 4.3: UML-Klassendiagramm einer SDN Application mit Treibernutzung

Die Parameter der einzelnen Methoden von TNDriver sind:
`addNetconfSwitch(ip, user, passwd, onosUser, onosPasswd)`
`addOpenflowRules(jsonFile, onosUser, onosPasswd)`
`sendNetconfRPC(ip, xmlFile, service)`
`restapi(url, payload, onosUser, onosPasswd)`
Der CarModeSwitcher soll dabei innerhalb der ONOS-Kommandozeile aufrufbar sein mit dem gewünschten Fahrzeugmodus als Argument. Für den Fahrzeugmodus “Fahren“ soll “drive“ und für den Fahrzeugmodus “Update / Diagnose“ soll “updateDiag“ als Argument übergeben werden.

5 Implementierung

Dieses Kapitel stellt die verwendete Hardware zur Implementierung vor und geht auf die Implementierung des Konzeptes in ONOS sowie die Erstellung und Durchführung der Testfälle ein.

5.1 Hardware

In diesem Kapitel wird die Hardware vorgestellt, welche im Rahmen dieser Arbeit verwendet wurde.

5.1.1 Switch

Die Applikation zur Konfiguration des TSSDN-Netzwerkes wird auf dem TrustNode IRTN16R TSSDN-Switch der Firma InnoRoute getestet (siehe Abbildung 5.1).

Abbildung 5.1: TrustNode IRTN16R TSSDN-Switch (Quelle: TrustNode Datenblatt [19])

Dieser Switch ist ein Forschungsgerät für die Netzwerkanforderungen von Industrie 4.0 bezüglich Latenz, Flexibilität und Sicherheit. Er zeichnet sich durch eine niedrige Latenz (2.5 µs bei 1 Gbit/s laut Hersteller [19]) und einen niedrigen Jitter von 1 µs [19] sowie durch die Flexibilität von TSSDN aus. Die Software und Hardware können je nach

Anforderung verändert werden. Freie Ressourcen der CPU werden für Softwareanpassungen genutzt und die Hardware des FPGA ist durch einen eigenen Bitstrom anpassbar. Auf dem Switch läuft die Open vSwitch (OVS) Software, welche Flow Rules vom SDN-Controller erhält. Das Regelwerk vom Open vSwitch wird kontinuierlich abgefragt, um Regeln in die Hardware-Weiterleitungstabelle zu übertragen. Ein Ethernet-Paket durchläuft dabei eine Verarbeitungskette im Switch, die in Abbildung 5.2 dargestellt ist.

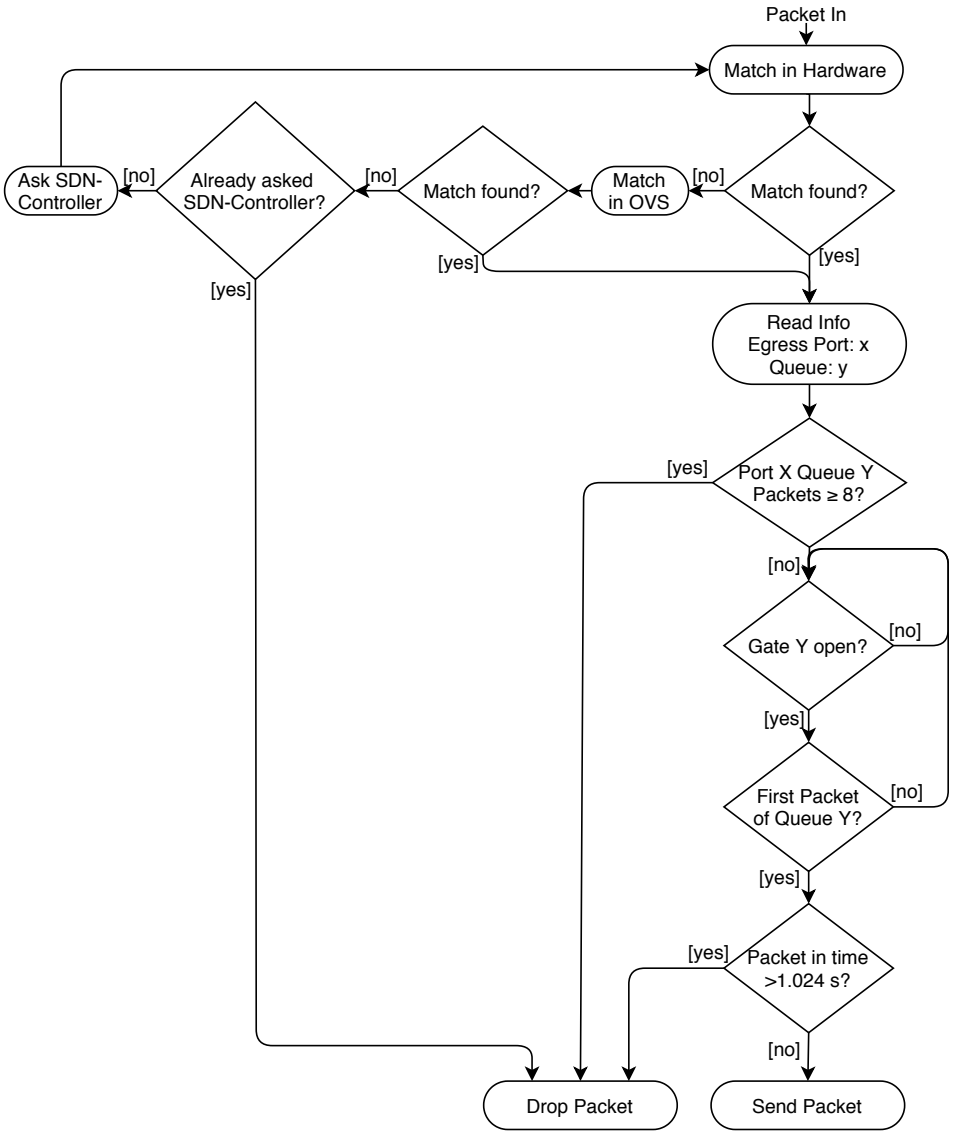


Abbildung 5.2: Verarbeitungskette des Switches für ein Ethernet-Paket

5.1.2 Oszilloskop

Um Pakete bei der physischen Übertragung analysieren zu können und somit die durch den Switch eingeführte Latenz zu ermitteln, wird das Oszilloskop Tektronix MSO4054B verwendet (siehe Abbildung 5.3).

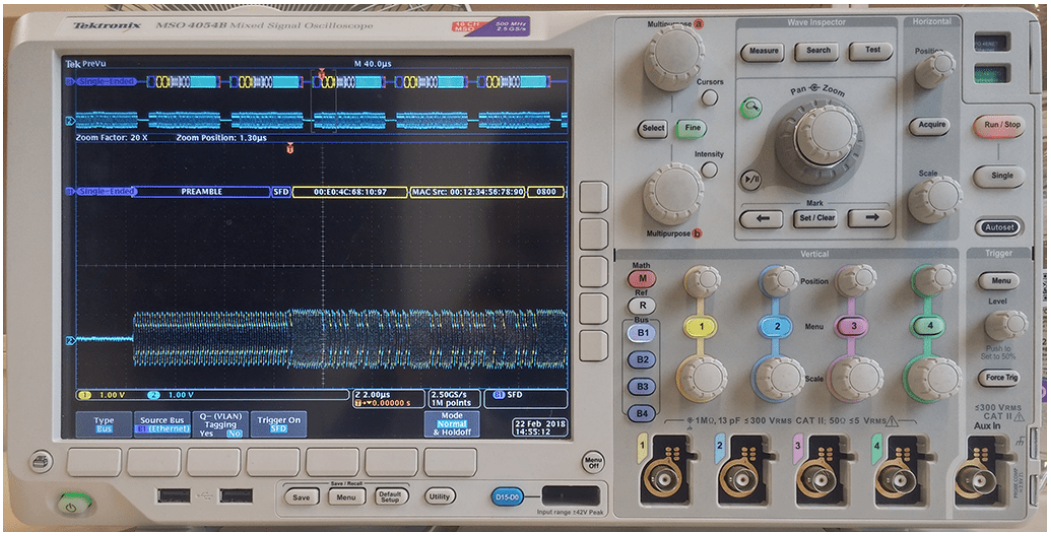


Abbildung 5.3: Decodierter Netzwerkverkehr auf dem Tektronix MSO4054B

Mit der Erweiterung “DPO4ENET“ können die elektrischen Signale bei der physischen Übertragung zu Ethernet-Paketen decodiert werden. Durch den Einsatz von “TDP0500 Differentialtastköpfen“ (siehe Abbildung 5.4) im Gegensatz zu normalen Tastköpfen wird zudem die Störung der Signale gemindert und es können somit Geschwindigkeiten von bis zu 100 Mbit/s decodiert werden.

Abbildung 5.4: TDP0500 Differentialtastkopf

5.2 Umsetzung des Konzeptes

Das in Kapitel 4.2 vorgestellte Konzept besteht aus einem Hilfstreiber und der eigentlichen Applikation "CarModeSwitcher". Ohne den Treiber kann die Applikation den Switch nicht ansprechen, daher wird dieser zuerst beschrieben.

5.2.1 Treiber

Der konzeptionierte Treiber soll von mehreren Applikationen wiederverwendbar sein. Daher enthält er keinen Zustand in Form von Objektvariablen. Alle Funktionen des Treibers sind als `static` definiert, um diesen nicht vorher pro Applikation instanziieren zu müssen. Es wurden zwei weitere Anforderungen an den Treiber gestellt: Flow Rules per OpenFlow auf dem Switch installieren zu können und GCL-Konfigurationsdateien per NETCONF übertragen zu können. ONOS bietet bereits eine REST-API, um OpenFlow Regeln zu installieren und die SSH-Authentifizierung von NETCONF zu übernehmen. Also wurde zuerst die private Hilfsfunktion `restapi` implementiert. Sie kann die verschiedenen REST-API-Endpunkte in Form einer variablen URL ansprechen und überträgt dabei den von der API erwarteten Payload sowie den Benutzernamen und Kennwort für die Authentifizierung. Umgesetzt wurde dies durch eine Java `HttpURLConnection`, auf welcher sich ein HTTP-Post Request mit gewünschter Authorization und Content aufbauen lässt. Nach außen zur Applikation soll der Treiber nun seine weiteren protected Funktionen verfügbar machen. Die Funktionen `addOpenflowRules` und `addNetconfSwitch` nutzen intern die soeben erwähnte restapi-Hilfsfunktion und sprechen unterschiedliche ONOS REST-API-Endpunkte in Form von unterschiedlichen URLs an. Außerdem sind sie für die Erstellung des zu übertragenden Payloads zuständig. Dafür wird bei der Funktion `addOpenflowRules` ein Pfad zu einer JSON-Datei mit den Flow Rules verlangt, um diese Datei einzulesen und als Payload zu übermitteln sowie dem Benutzernamen und Kennwort für die Authentifizierung an ONOS. In der Funktion `addNetconfSwitch` werden wieder der Benutzername und das Kennwort für die Authentifizierung an ONOS benötigt, aber zusätzlich auch der Benutzername und das Kennwort für die SSH-Authentifizierung an dem Switch. Aus den SSH-Authentifizierungsdaten wird dann der zu übermittelnde Payload erstellt.
Die letzte nach außen verfügbare Treiberfunktion `sendNetconfRPC` ist für die Übertragung der GCL-Konfigurationsdateien zuständig. Hierfür stellt ONOS keine REST-API

bereit, daher kann die restapi-Hilfsfunktion in diesem Fall nicht genutzt werden. Es müssen die in Abbildung 4.2 dargestellten Subkomponenten Device, Driver und Device Cfg. genutzt werden. Das Device wird durch eine DeviceId bestehend aus dem "netconf" Präfix, der IP und dem Port (Standard: 830) angesprochen. Daraufhin kann ein DriverHandler auf der DeviceId erstellt werden. Zuletzt setzt ein ConfigSetter mit diesem DriverHandler die GCL-Konfigurationsdatei in Form einer Pfadangabe zur Datei auf dem Switch per NETCONF um.

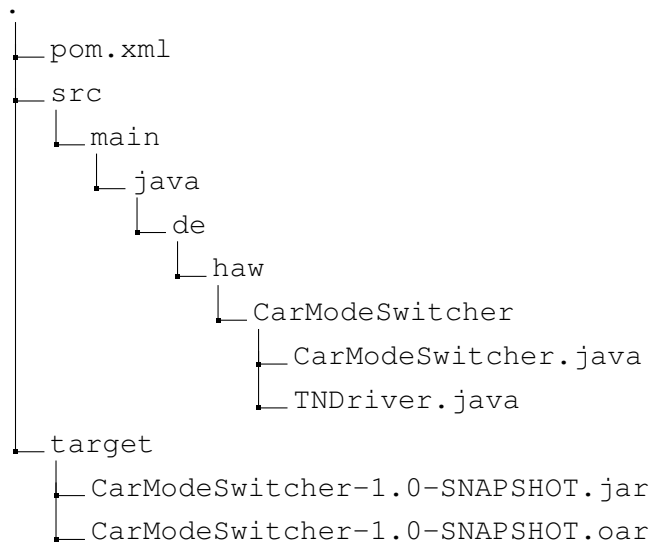
5.2.2 Applikation CarModeSwitcher

Die Applikation CarModeSwitcher soll über die Kommandozeile zwischen den verschiedenen Fahrzeugmodi umschalten können. Dafür wird der Fahrzeugmodus als Argument in Form eines Strings vorausgesetzt. Bei jedem Aufruf der Applikation wird zuerst geprüft, ob diese bereits einmal ausgeführt wurde. Falls die Applikation bisher noch nicht ausgeführt wurde, ruft sie vom TNDriver die Funktion `addNetconfSwitch` auf um die NETCONF-Verbindung mit dem Switch für die weiteren Anweisungen herzustellen. Daraufhin wird das Argument des Fahrzeugmodus ausgewertet. Je nachdem ob "drive" oder "updateDiag" als Argument übergeben wird, werden durch eine Fallentscheidung die zum Modus passende GCL-Konfigurationsdatei und Flow Rule-Datei festgelegt. Die passenden Konfigurationsdateien werden dann durch den Aufruf von den TNDriver Funktionen `sendNetconfRPC` und `addOpenflowRules` an den Switch übertragen. Wird ein nicht definierter Fahrzeugmodus angegeben, bleibt das System im letzten gültigen Zustand und es wird eine Fehlermeldung ausgegeben.

5.2.3 Kompilierung

Um den Java-Quellcode in eine installierbare und ausführbare ONOS SDN-Application zu übersetzen wird Apache Maven benötigt. Dieses Werkzeug sucht nach einer **P**roject **O**bject **M**odel Konfigurationsdatei "pom.xml", in welcher alle Abhängigkeiten stehen müssen sowie weitere Eigenschaften wie der Name Applikation. Der Name der Applikation in der pom.xml muss mit dem Package im Quellcode übereinstimmen und bestimmt somit auch die Ordnerstruktur des Projektes.

Beispielsweise muss durch den Eintrag:
`<onos.app.name>de.haw.CarModeSwitcher</onos.app.name>`
folgende Ordnerstruktur eingehalten werden:



Durch Angabe der benötigten ONOS-Dienste org.onosproject.netconf, org.onosproject.-drivers.netconf und org.onosproject.openflow-base werden diese bei der Installation und Aktivierung der Applikation automatisch gestartet. Die Buildabhängigkeiten werden Artefakte genannt und beinhalten: onos-api, onos-protocols-netconf-api, onos-cli und org.apache.karaf.log. Des Weiteren werden die Compiler-Argumente und Build-Plugins in dieser Datei angegeben. Nach der Kompilierung durch den Befehl mvn clean install befindet sich im Target-Ordner eine **O**NOS **A**pplication a**R**chive .oar Datei mit dem Namen aus der pom.xml. Dies ist die fertig kompilierte und gepackte Applikation, die in ONOS installiert und ausgeführt werden kann.

5.3 Durchführung der Testfälle

In Kapitel 3 wurden die Anforderungen und Testfälle vorgestellt, durch welche ermittelt werden soll, ob sich der Einsatz von TSSDN für den Automobilbereich eignet und ob der Switch diese Anforderungen ebenfalls erfüllt. Für die tatsächliche Durchführung müssen zuerst die passenden Konfigurationsdateien erstellt werden und diese daraufhin auf den Switch aufgespielt werden. Dann erst kann mit den Messungen begonnen werden.

5.3.1 Erstellung der Konfigurationsdateien

Die in Kapitel 3.1 genannten Anforderungen bezüglich Bandbreitenverteilung, Latenz und Jitter an die Architektur erfordern einen genauen Schedule sowie eine eindeutige Zuordnung jedes Hosts in seinen eigenen Zeitschlitz. Um dies umzusetzen, erhält jeder Host eine eigene Priorität durch die Flow Rules, welche von der GCL somit einzeln freigegeben werden kann.

Host / Fluss	Priorität	Anforderung
C	3	Steuerdaten 3 Mbit/s
A	2	LIDAR / Kamerasensoren 68 Mbit/s
B	1	weitere Anwendung Restkapazität
ARP	0	Adressauflösung, Schutzband

Tabelle 5.1: Flow Rules Testarchitektur

Es ist ein passender Schedule zu erstellen, der die Bandbreiten garantiert (siehe Tabelle 5.1) und dabei eine maximale Latenz von 600 µs gewährleistet. Somit muss eine GCL mit folgenden Eigenschaften generiert werden:

Eintrag ID	Priorität	Dauer
0	3, 0	16 µs
1	2, 0	450 µs
2	1, 0	75 µs
3	0	16 µs

Tabelle 5.2: GCL Testarchitektur

Die gesamte Dauer eines Zyklus der in Tabelle 5.2 dargestellten GCL ist 557 µs und somit unterhalb der geforderten 600 µs. Da die Steuerdaten von Host C mit der Priorität 3 nur 3 Mbit/s von den verfügbaren 100 Mbit/s zugeteilt bekommen sollen, stehen ihnen 3 % der 557 µs zur Verfügung, also 16 µs. In diesen 16 µs können bei 100 Mbit/s nur 200 Byte übertragen werden, weshalb für diesen Testfall für alle Hosts 200 Byte als Paketgröße festgelegt wurde.

Im Folgenden ist ein Auszug für Host C aus der Flow Rule JSON-Datei dargestellt, welche als Payload an die ONOS REST-API verschickt wird:

```
{
  "flows": [
    {
      "isPermanent": true,
      "selector": {
        "criteria": [
          {
            "type": "IN_PORT",
            "port": 4
          }
        ]
      },
      "priority": 40000,
      "deviceId": "of:000000000000000a",
      "timeout": 0,
      "treatment": {
        "instructions": [
          {
            "type": "QUEUE",
            "queueId": 3
          },
          {
            "type": "OUTPUT",
            "port": 3
          }
        ]
      }
    },
    {
      ...hier weitere Regeln...
    }
  ]
}
```

Die dazugehörige GCL-Konfigurationsdatei im XML-Format hat folgenden Inhaltsauszug für einen der insgesamt zehn Ports:

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:rpc xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
message-id="1">
  <nc:edit-config>
    <nc:target><nc:running/></nc:target>
    <nc:default-operation>replace</nc:default-operation>
    <nc:config>
      <TNtas xmlns="urn:sysrepo:TrustNode:TNsysrepo">
        <ports>
          <id>0</id>
            <GCL> <id>0</id>
              <timeperiod>16000</timeperiod>
              <gatestates>9</gatestates> </GCL>
            <GCL> <id>1</id>
              <timeperiod>450000</timeperiod>
              <gatestates>5</gatestates> </GCL>
            <GCL> <id>2</id>
              <timeperiod>75000</timeperiod>
              <gatestates>3</gatestates> </GCL>
            <GCL> <id>3</id>
              <timeperiod>16000</timeperiod>
              <gatestates>1</gatestates> </GCL>
            <admin_base_time>0</admin_base_time>
            <admin_cycle_time_ext>0</admin_cycle_time_ext>
            <gate_enable>true</gate_enable>
        </ports>
        <ports>
          ...hier weitere Ports...
        </ports>
      </TNtas>
    </nc:config>
  </nc:edit-config>
</nc:rpc>
```

Für die in Kapitel 3.2 genannten Anforderungen bezüglich Domänenbildung und Kommunikationsflüsse der Fahrzeugmodi “Fahren“ und “Update / Diagnose“ werden zwei separierte Flow Rule-Dateien erstellt. Pro Fahrzeugmodus können die erlaubten Kommunikationspartner das Protokoll ARP untereinander nutzen, um sich zu finden. Auf jedem Host läuft zusätzlich ein Webserver auf Port 80, um prüfen zu können, ob der entfernte Host wirklich nur per ARP gefunden werden kann, nicht jedoch sein Dienst auf Port 80 erreichbar ist. Das gezielte erlauben von ARP, aber die Unterbindung von Kommunikation auf Port 80 zu entfernten Hosts kann nur durch einen SDN-Switch umgesetzt werden, nicht jedoch durch einen üblichen VLAN-fähigen Switch und ist somit ein Unterscheidungsmerkmal von SDN, welches getestet wird.
Der Inhalt weiterer Konfigurationsdateien für die Prüfung der restlichen Anforderungen wird bereits in Kapitel 3 unter den jeweiligen Testfällen beschrieben.

5.3.2 Aufspielen der Konfigurationsdateien

Um komfortabel und schnell zwischen unterschiedlichen Konfigurationsdateien wechseln zu können wurde eine weitere ONOS-Applikation geschrieben. Sie nutzt ebenfalls den entwickelten Treiber TNDriver und nimmt als erstes Argument den Pfad der aufzuspielenden GCL-Konfigurationsdatei und als zweites Argument den Pfad der Flow Rule JSON-Datei. Durch die Angabe der passenden Dateien können alle Testfälle nacheinander durchgeführt werden. Abbildung 5.5 zeigt die Architektur des Testprogramms.

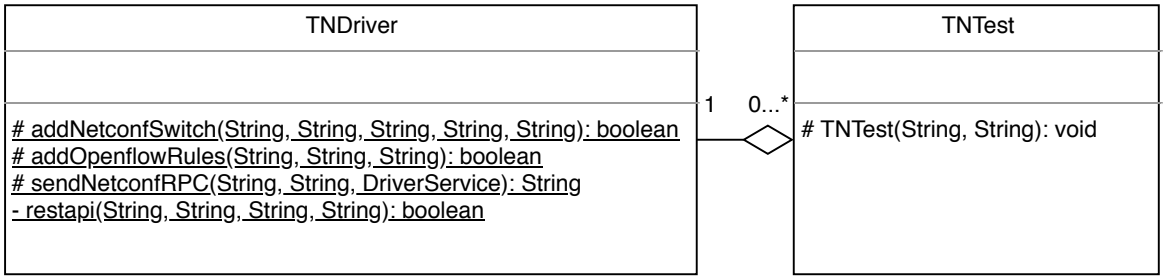


Abbildung 5.5: UML-Klassendiagramm der Test-Applikation

5.3.3 Durchführung der Messungen

Je nach Messung werden unterschiedliche Werkzeuge verwendet. Für die Testfälle bezüglich Bandbreitenverteilung, Latenz, Jitter und Priorisierung wird auf Host C das Programm “tcpdump“ alle eingehenden sowie ausgehenden Pakete seiner beiden Netzwerk-Interfaces für eine Sekunde pro Testdurchlauf aufnehmen und für die spätere Auswertung speichern. Mit “iperf“ wird auf allen Hosts UDP-Datenverkehr erzeugt, um die Links voll auszulasten. Es wird kein TCP verwendet, da die Flusskontrolle versuchen würde, gegen die Zeitschlitze des Schedules mit einer Minderung der Datenrate entgegenzuwirken ([18] Tabelle 4.17).
Um die Testfälle bezüglich Domänenbildung und Kommunikationsflüsse verifizieren zu können, wird in jedem Fahrzeugmodus per Nmap ermittelt, welche Kommunikationspartner sich tatsächlich gegenseitig erreichen können und welche Ports sichtbar sind.
Für die Überprüfung, ob ein Moduswechsel unterbrechungsfrei für in beiden Modi erlaubte Kommunikationsströme ist, werden von einem Sender 30.000 maximal große Ethernet-Frames an einen Empfänger verschickt, welcher mit tcpdump die Kommunikation speichert. Während der Übertragung wird ein neues Flow Rule Regelwerk mit OpenFlow und ein geänderter Schedule in Form einer anderen GCL per NETCONF übertragen. In dem gespeicherten Mitschnitt der Kommunikation beim Empfänger wird dann gezählt, ob tatsächlich 30.000 Ethernet-Pakete ankamen.
Um zu messen, wie lange es dauert, bis ein Moduswechsel durch eine Konfigurationsänderung vom Switch umgesetzt wird, sendet Host C mit iperf kontinuierlich UDP-Pakete über den Switch an sein zweites Netzwerk-Interface. Die Weiterleitungstabelle vom Switch ist anfänglich leer und somit wird diese Kommunikation nicht weitergeleitet. Host C installiert als SDN-Controller daraufhin den Fluss, der diese Kommunikation erlaubt und es wird die zeitliche Differenz zwischen dem Absenden der Änderung und dem tatsächlich ersten empfangenen Paket auf dem zweiten Netzwerk-Interface gemessen. Die Messung erfolgt wieder durch den Einsatz von tcpdump auf allen Netzwerk-Interfaces des Hosts. Für OpenFlow wird als Zeitpunkt zum Absenden der Änderungen der Moment gewählt, in dem das OpenFlow Paket mit der Modifikationsanweisung (FLOW_MOD-Paket) zum Switch versandt wurde .
Da die Übertragung von NETCONF per SSH verschlüsselt ist, kann nur der Zeitpunkt des Verbindungsaufbaus durch ein SYN-Paket vom Host zum Switch als Moment der Änderungsmitteilung gewertet werden.
In allen bisherigen Testfällen ist die Latenz der Übertragung inklusive der Verzögerung durch die Host-Betriebssysteme, deren Netzwerk-Stacks und der Datenübertragungsra-

te in die Messungen eingeflossen. Um zu ermitteln, wie hoch der Anteil vom Switch an den Messergebnissen ist, wird mit einem Oszilloskop die tatsächlich durch den Switch eingeführte Latenz gemessen. Dafür wird mit einer stetig wachsenden Größe der Nutzdaten der Zeitpunkt unmittelbar vor dem Eintritt des Ethernet-Paketes auf der physischen Übertragung (Netzwerkkabel) in den Switch und nach Austritt aus dem Switch genommen und die Differenz als Latenz berechnet. Der Messaufbau mit dem Oszilloskop ist in Abbildung 5.6 dargestellt.

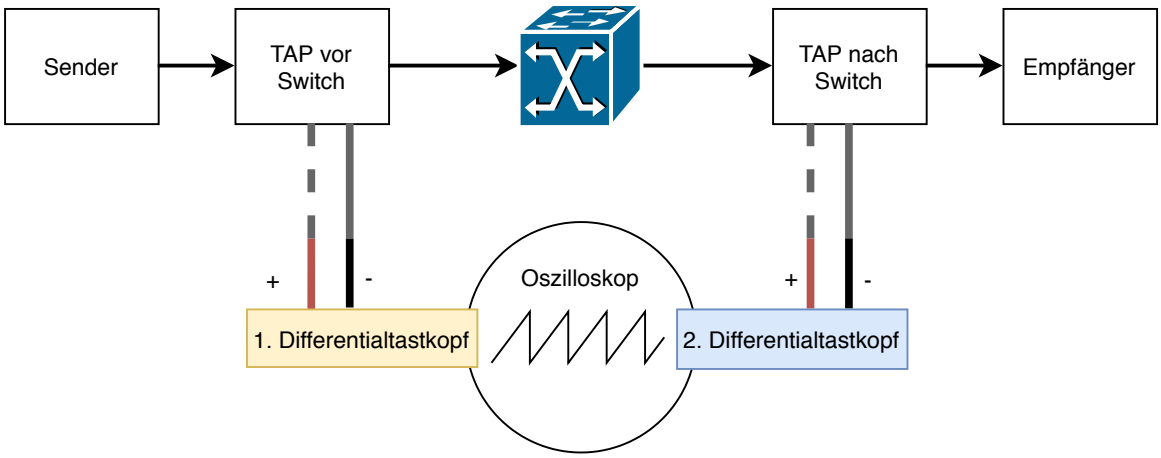


Abbildung 5.6: Aufbau der Latenzmessung mit Oszilloskop

Es werden zwei Differentialtastköpfe am Oszilloskop angeschlossen. Um an die Sende- und Empfangsleitung der physischen Übertragung zu gelangen, und somit den Datenverkehr zwischen den Teilnehmern analysieren zu können, werden Netzwerk **T**est **A**ccess **P**orts (TAPs) eingesetzt (siehe Abbildung 5.7).

Abbildung 5.7: Netzwerk-TAP

Die Netzwerk-TAPs erlauben es, die Differentialtastköpfe vom Oszilloskop parallel zum Netzwerkkabel anzuschließen.
Der 1. Differentialtastkopf ist vor dem Switch mit der Sendeleitung des Senders verbunden.
Der 2. Differentialtastkopf ist nach dem Switch mit der Empfangsleitung des Empfängers verbunden.
Für den 1. Differentialtastkopf ist ein Trigger für den Start Frame Delimiter eines Ethernet-Paketes konfiguriert. Wird nun ein Ethernet-Paket gesendet, legt der Trigger den Zeitpunkt $t_0 = 0$ fest. Das Ethernet-Paket wird daraufhin von dem 2. Differentialtastkopf nach der Verarbeitung vom Switch erfasst und der gemessene Zeitpunkt wird als t_1 notiert. Die Latenz L ergibt sich aus der Differenz zwischen t_0 und t_1.
Der Jitter wird durch die größte Differenz der Latenzen aller Durchläufe dieses Testfalls ermittelt:

$$Jitter = max(L) - min(L) \tag{5.1}$$

Alle in diesem Kapitel beschriebenen Testfälle werden zehnmal durchgeführt und daraus die minimale, durchschnittliche und maximale Latenz sowie der Jitter berechnet.

6 Evaluation

In diesem Kapitel werden die Testergebnisse der unterschiedlichen Testfälle vorgestellt und evaluiert. Es wird darauf eingegangen, ob die erwarteten Testergebnisse sowohl vom Switch als auch von TSSDN allgemein erreicht werden konnten und sich diese somit für den Einsatz im Automobilbereich eignen. Es wurde immer eine Datenübertragungsrate von 100 Mbit/s gewählt, um mit dem Oszilloskop die genaue Latenz des Switches messen und mit den Ergebnissen vergleichen zu können.

6.1 Bandbreitenverteilung, Latenz und Jitter

Mit dem erstellten Schedule sollte geprüft werden, ob sich die Anforderungen der Bandbreitenverteilung im Automobil durch eine geeignete TAS-Konfiguration umsetzen lassen.

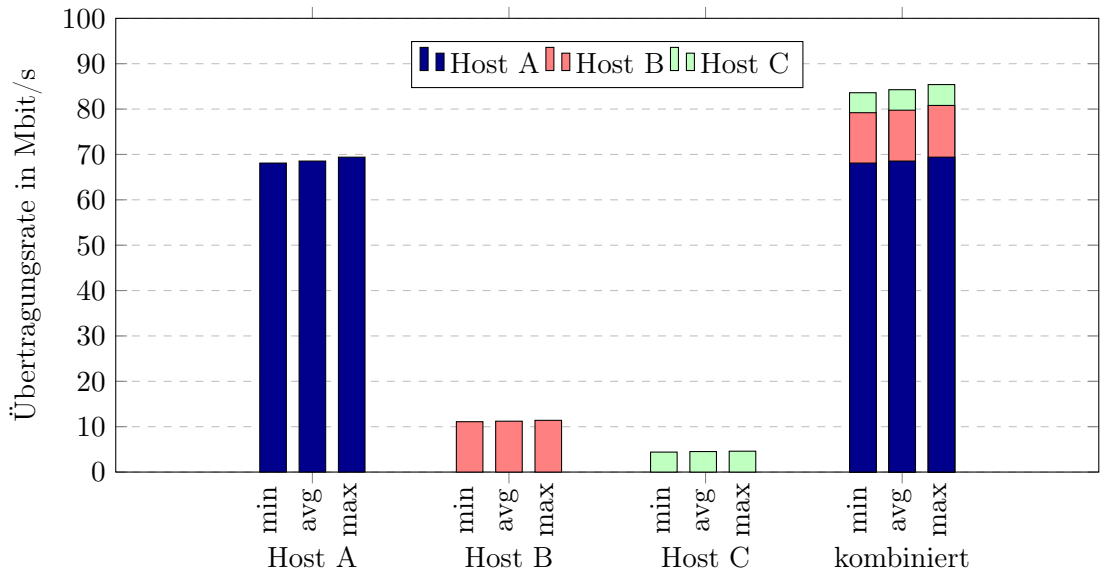


Abbildung 6.1: Bandbreitenverteilung mit Automobil-Schedule

Die Messergebnisse sind in Abbildung 6.1 dargestellt. Host A steht für Daten von LIDAR-

und Kamerasensoren, für welche eine Bandbreite von 68 Mbit/s garantiert werden muss. Steuerdaten vom CAN-Bus werden durch Host C dargestellt, welche eine garantierte Bandbreite von 3 Mbit/s benötigen. Die restliche Bandbreite steht für weitere Anwendungen und somit Host B zur Verfügung. Es ist zu sehen, dass die Anforderungen bezüglich der Bandbreitenverteilung mit dem Schedule korrekt umgesetzt werden konnten, durch das Schutzband die maximal mögliche Datenrate von 100 Mbit/s allerdings nicht erreicht wird.
Eine weitere Anforderung an den Schedule war eine maximale Latenz von 600 µs und ein Jitter von höchstens 60 µs für die Steuerdaten.

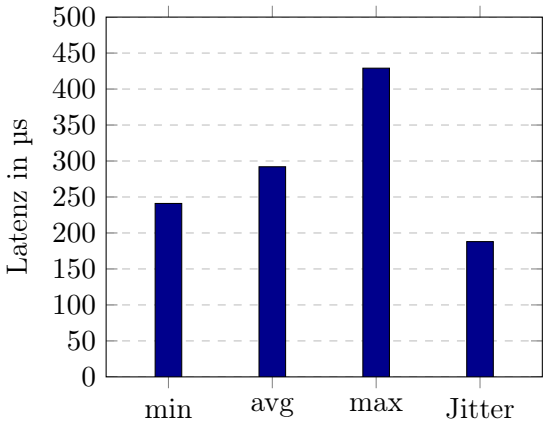


Abbildung 6.2: Latenz und Jitter mit Automobil-Schedule

Die Ergebnisse der Latenzmessung und der daraus berechnete Jitter für die Steuerdaten sind in Abbildung 6.2 dargestellt. Es konnte eine maximale Latenz von unter 600 µs erreicht und somit die Anforderung erfüllt werden. Allerdings wurde das Ziel von höchstens 60 µs Jitter verfehlt. Das Ergebnis wird durch den fehlenden Einsatz einer durchgängigen Zeitsynchronisierung zwischen den sendenden Hosts und dem Switch hervorgerufen, wodurch der Zeitschlitz zum Senden verpasst und somit eine Verzögerung bis zum nächsten Zeitschlitz verursacht wird.

6.2 Domänenbildung und Kommunikationsflüsse

In dieser Testreihe sollte zunächst geprüft werden, ob die Anforderungen bezüglich Domänenbildung und Beschränkung der Kommunikationsflüsse umgesetzt werden können. Es wurde geprüft, ob die Kommunikation zwischen erlaubten Kommunikationspartnern

funktioniert und ob die Kommunikation zwischen verbotenen Kommunikationspartnern unterbunden ist. Pro Domäne wurde auf jeweils einem Host ein Netzwerkscan inklusive Portscan mit dem Befehl:
`nmap -n 10.0.0.0/24 -oG -`
durchgeführt. Für jeden Fahrzeugmodus wurden diese Scans wiederholt. Im Folgenden ist das Ergebnis während des Fahrzeugmodus "Fahren" aufgelistet. Die Testarchitektur wurde in Kapitel 3.2 bereits vorgestellt (siehe Abbildung 3.2).
Nmap Scan von Host aus der Antrieb-Domäne (10.0.0.1):

```
Host: 10.0.0.2 () Status: Up
Host: 10.0.0.3 () Status: Up
Host: 10.0.0.4 () Status: Up
Host: 10.0.0.7 () Status: Up
Host: 10.0.0.1 () Ports: 80/open/tcp//http///
```

Nmap Scan von Host aus der Komfort-Domäne (10.0.0.5):

```
Host: 10.0.0.7 () Status: Up
Host: 10.0.0.5 () Ports: 80/open/tcp//http///
```

Nmap Scan von Host aus der Diagnose-Domäne (10.0.0.6):

```
Host: 10.0.0.6 () Ports: 80/open/tcp//http///
```

Nmap Scan vom Host Infotainment (10.0.0.7):

```
Host: 10.0.0.1 () Status: Up
Host: 10.0.0.2 () Status: Up
Host: 10.0.0.3 () Status: Up
Host: 10.0.0.4 () Status: Up
Host: 10.0.0.5 () Status: Up
Host: 10.0.0.7 () Ports: 80/open/tcp//http///
```

Wie erwartet ist die Sicht eines Hosts nur auf seine Domäne sowie den Host der Infotainment-Domäne beschränkt, sofern die Domäne aktiv ist. Ist die Domäne nicht aktiv, sieht der Host nur sich selbst und keine weiteren Domänenmitglieder oder den Host der Infotainment-Domäne. Nur der Host der Infotainment-Domäne hat eine Sicht auf alle Teilnehmer der aktiven Domänen, allerdings ebenfalls nicht auf deaktivierte Domänen, wie in diesem Modus an der Diagnose-Domäne mit dem Host 10.0.0.6 zu sehen ist. Bei einem herkömmlichen Switch ohne SDN würden alle Hosts sich gegenseitig sehen. Die Trennung von Domänen ist prinzipiell auch mit einem VLAN-fähigen Switch möglich, allerdings nicht die gezielte Blockierung von den Webservern anderer Hosts innerhalb der gleichen Domäne. Ein Host sieht jeweils nur seinen lokalen Webserver durch den Netzwerk- und Portscan. Die Anforderungen bezüglich der erlaubten und verbotenen

Kommunikation sind somit erfüllt.
Im weiteren Verlauf dieser Testreihe sollte geprüft werden, ob der Wechsel eines Fahrzeugmodus unterbrechungsfrei für einen Strom ist, welcher in beiden Modi erlaubt ist. Dafür wurden beim Sender genau 30.000 UDP-Pakete zum empfangenden Host gesendet und beim Empfänger gezählt, um zu ermitteln, ob ein Paketverlust aufgetreten ist.

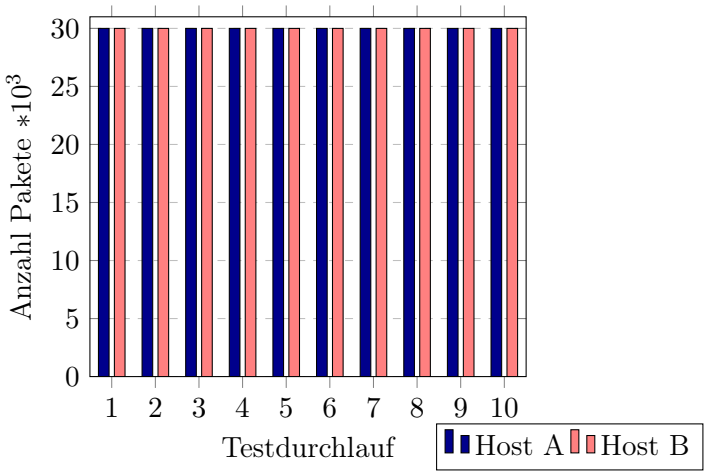


Abbildung 6.3: Messung gesendete und empfangene Pakete während Moduswechsel

Die Messergebnisse in Abbildung 6.3 zeigen, dass es zu keiner Unterbrechung bei einem Moduswechsel kam und alle gesendeten UDP-Pakete beim Empfänger angekommen sind. Zuletzt solle ermittelt werden, wie lange es dauert, einen Moduswechsel mit geänderter Konfiguration umzusetzen. Dafür wird die zeitliche Differenz zwischen dem Abschicken einer Flow-Installation und dem ersten beim Host empfangenen Datenpaket gemessen.

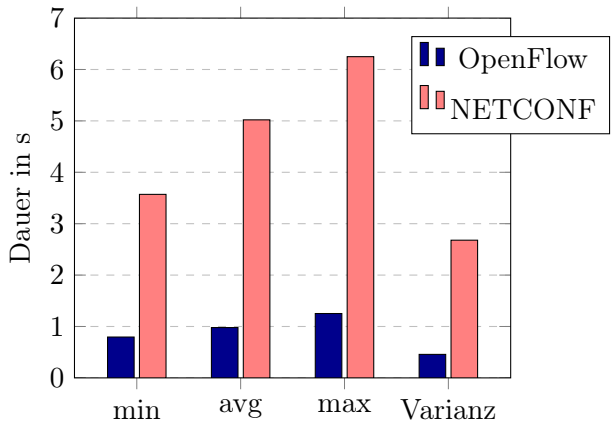


Abbildung 6.4: Dauer bis zur Umsetzung der Konfigurationsänderung

Eine Konfigurationsänderung dauert mit OpenFlow maximal 1,3 Sekunden, während

NETCONF maximal über 6 Sekunden benötigt (siehe Abbildung 6.4). Für OpenFlow ist diese hohe Dauer unüblich, da andere am Markt verfügbare SDN-Switches bis zu 40.000 Flow-Modifikationen pro Sekunde durchführen können und somit nur 25 µs benötigen [15]. Die Dauer der Konfigurationsänderung mit NETCONF konnte nicht mit anderen am Markt verfügbaren Switches verglichen werden, da sich keine Vergleichswerte finden ließen. Somit besitzt der in dieser Arbeit eingesetzte Switch eine zu lange Wartezeit, sodass nur in nicht zeitkritischen Situationen, wie etwa zur Diagnose in einer Werkstatt, der Fahrzeugmodus gewechselt werden darf. Mit einem anderen Switch könnten während der Fahrt Flüsse geändert werden, aber ob ein geänderter Schedule währenddessen per NETCONF übertragen werden dürfte, bleibt offen.

6.3 Prioritäten

Da der CAN-Bus die Priorisierung von Nachrichten erlaubt, um somit die Latenz bei Hintergrunddatenverkehr (Cross-Traffic, CT) auf 600 µs zu begrenzen, soll geprüft werden, ob durch den Einsatz von IEEE 802.1Q Prioritäten dies ebenfalls erreicht werden kann. Zudem soll ermittelt werden, ob die Latenz und der Jitter durch einen optimierten Schedule weiter verbessert werden kann. Es ist anzumerken, dass durch den Einsatz von maximalen Ethernet-Frames (1522 Byte) in diesem Test bei einer Datenübertragungsrate von 100 Mbit/s die Übertragungszeit eines Ethernet-Frames bereits 124 µs beträgt. Dagegen ist eine CAN-Nachricht maximal 64 Byte groß und kann somit bessere Latenzen erzielen. Weil aber auch große Pakete wie etwa Bilder von HD-Rückfahrkameras in einem TSSDN-Automobilnetzwerk übertragen werden sollen, wurden für diesen Test maximale Ethernet-Frames gewählt.

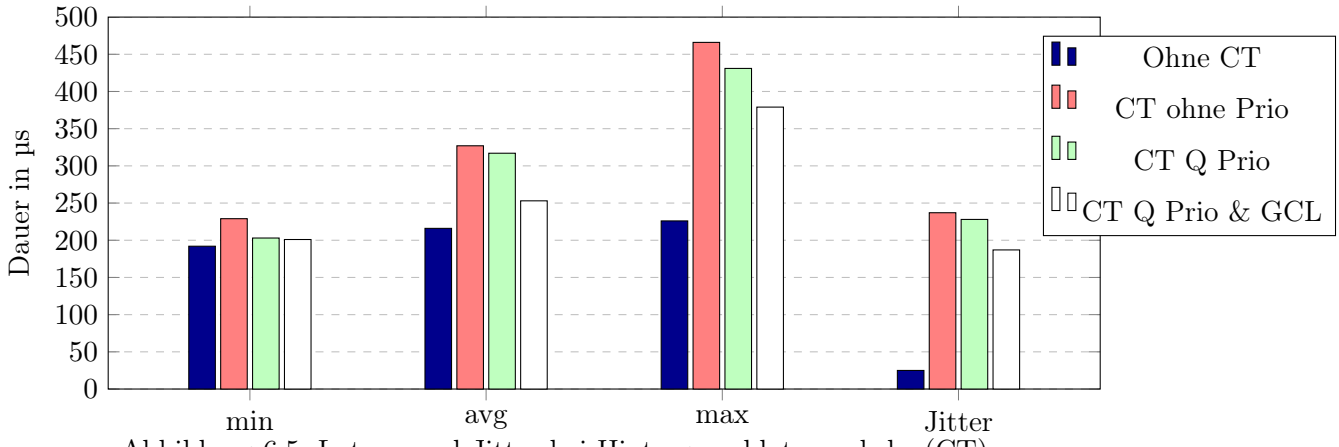


Abbildung 6.5: Latenz und Jitter bei Hintergrunddatenverkehr (CT)

Der Test ist in vier Phasen durchgeführt worden und die Messergebnisse sind in Abbildung 6.5 dargestellt. In der ersten Phase wurde die Latenz und der Jitter ohne CT gemessen, um einen Richtwert zu erhalten. Wie erwartet ist der Jitter mit nur 25 µs sehr niedrig. In der zweiten Phase wurde CT erzeugt und der Datenverkehr war gleichberechtigt, wodurch eine wesentlich höhere Latenz sowie Jitter resultieren, die maximale Latenz jedoch weiterhin unter den geforderten 600 µs liegt. Die dritte Phase führt wie bei CAN eine höhere Priorität für die zeitkritischen Nachrichten ein, wodurch eine Verbesserung der Latenz und des Jitters für diese Nachrichten erkennbar ist. Durch die Einführung eines Schedules in Phase vier konnte die Latenz und der Jitter weiter reduziert werden. Allerdings konnte der Jitter nicht unter die geforderten 60 µs gesenkt werden, was abermals durch das Fehlen der Zeitsynchronisation zu begründen ist, wie bereits in Kapitel 6.1 erläutert. Somit reicht sowohl die Einführung von Prioritäten als auch der Einsatz eines Schedules ohne Zeitsynchronisierung nicht für die Jitter-Anforderungen von 60 µs bei maximalen Ethernet-Frames aus.

6.4 Leistungs- und Funktionstests Switch

6.4.1 Bandbreitenverteilung

Die Umsetzung der Prioritäten und die Funktion der GCL wurde weiterführend analysiert (siehe Abbildung 6.6).

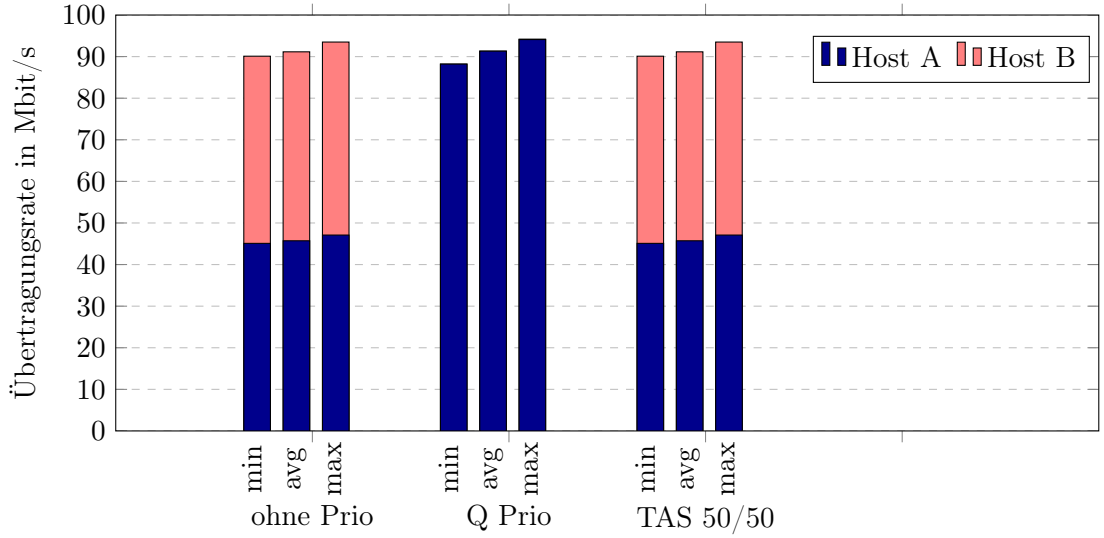


Abbildung 6.6: Vergleich Datenübertragungsraten

Im ersten Test mit zwei gleichberechtigten Hosts ohne eine Prioritätsvergabe haben beide Hosts die gleiche Datenübertragungsrate. Da im Ethernet-Standard IEEE 802.3 dieses Verhalten nicht spezifiziert oder gefordert ist, konnte im Vorfeld keine Erwartung definiert werden. Im zweiten Test hat Host A eine höhere 802.1Q Priorität, wodurch der Switch mit seiner strikten Prioritätsumsetzung den Datenverkehr von Host B wie erwartet vollständig verdrängt. Der dritte Test zeigt, wie durch die Vergabe von 50 % der verfügbaren Zeit an Host A und 50 % an Host B dieses Verhalten wieder ausgeglichen wird. Die Vergabe fester Zeitschlitze pro Priorität produziert das erwartete Ergebnis.
In Abbildung 6.7 ist ein Ausschnitt des mitgeschnittenen Datenverkehrs über die Zeit dargestellt, wodurch die Zeitschlitze der Hosts deutlich werden.

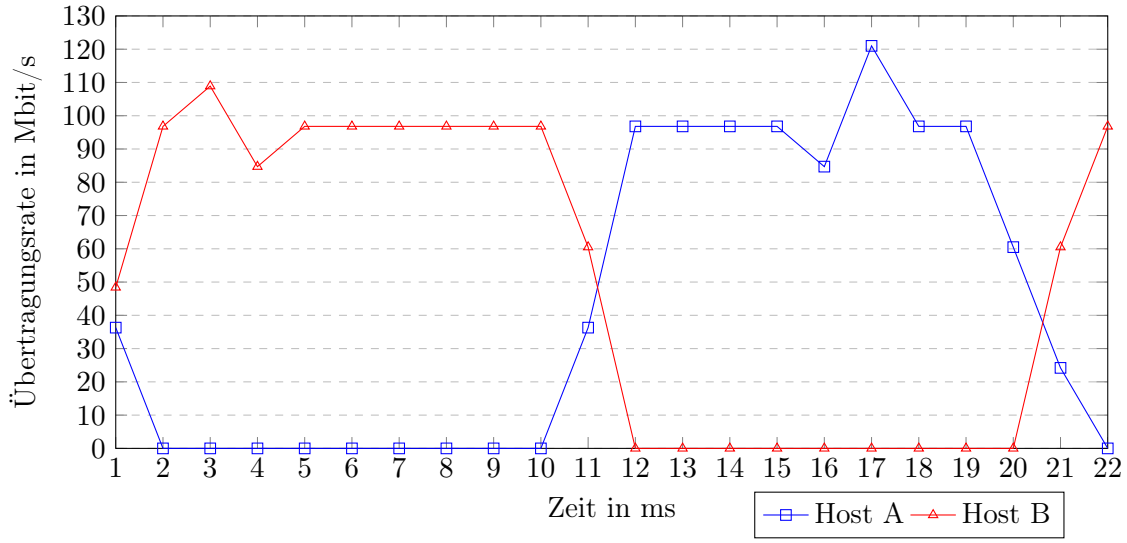


Abbildung 6.7: Datenübertragungsrate pro Host / Zeit

Es sind die gesonderten Zeitschlitze beider Hosts zu sehen. Die Priorität ist somit nicht mehr ausschlaggebend für die Datenübertragungsrate. Manche Messpunkte zeigen eine Datenübertragungsrate von über 100 Mbit/s, während die umliegenden Messpunkte darunter liegen. Dies deutet auf eine Pufferung durch das Host-Betriebssystem oder dessen Netzwerk-Stack hin, da alle Messpunkte summiert die korrekte Datenübertragungsrate von 100 Mbit/s liefern.

6.4.2 Latenz und Jitter des Switches

Die bisherigen Latenzmessungen wurden mit tcpdump durchgeführt und beinhalten somit die Latenzen, welche durch das Betriebssystem und den Netzwerk-Stack eingeführt wurden. Um zu ermitteln, wie groß der Anteil des Switches an diesen Latenzen ist, wurde mit einem Oszilloskop der Zeitpunkt unmittelbar vor und nach dem Switch auf der Netzwerkleitung gemessen, in dem ein Ethernet-Paket übertragen wurde. Es befindet sich kein Hintergrunddatenverkehr im Test-Netzwerk und nur ein Host ist angeschlossen. Somit kann geprüft werden, ob die Herstellerangaben von 2.5 µs Latenz und 1 µs Jitter, welche sich auf 1 Gbit/s beziehen, auch bei 100 Mbit/s eingehalten werden. Die Messung der Latenz eines minimalen Ethernet-Frames mit einer Größe 64 Byte durch das Oszilloskop ist in Abbildung 6.8 dargestellt.

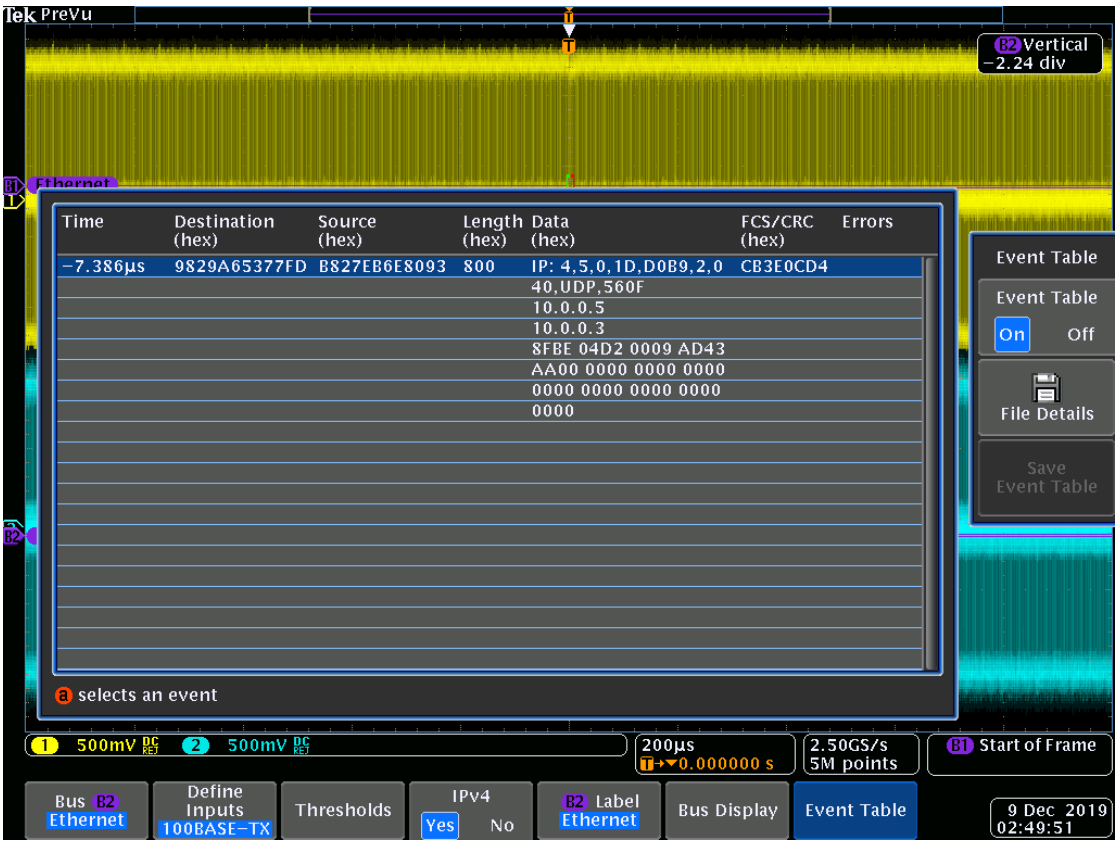


Abbildung 6.8: Latenz des Switches für einen minimalen Ethernet-Frame

Da bei 100 Mbit/s die Übertragung von Präambel, Start Frame Delimiter, minimalen Ethernet Frame und Interpacket Gap, also 672 Bit (84 Byte) bereits 6.72 µs dauert, muss diese Zeit vom Ergebnis abgezogen werden: 7.386 µs − 6.72 µs = 0.666 µs Latenz.

Es wurden zudem Messungen mit verschieden großen Nutzdaten durchgeführt und die vom Switch eingeführte Latenz berechnet (siehe Abbildung 6.9).

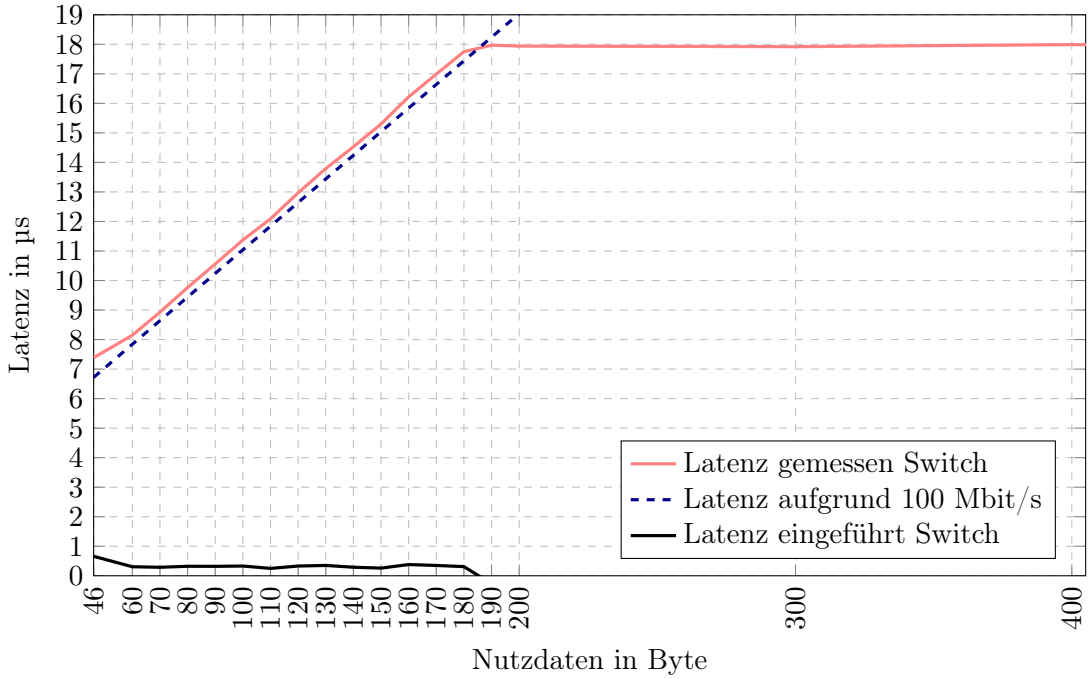


Abbildung 6.9: Latenz in Abhängigkeit der Nutzdaten

Es ist zu sehen, dass die Herstellerangabe von maximal 2.5 µs Latenz auch bei 100 Mbit/s eingehalten wird. Ab 190 Byte Nutzdaten wird vom Switch eingeführte Latenz berechnet (theoretisch) sogar negativ, da es sich um einen Cut-Through Switch handelt. Der Jitter wurde aus 10 Messungen mit jeweils 1500 Byte Nutzdaten im Ethernet-Frame berechnet:

$$18.06\,\mu s - 17.95\,\mu s = \underline{0.11\,\mu s} \tag{6.1}$$

Somit liegt auch der Jitter innerhalb der Herstellerangabe von 1 µs.

7 Fazit und Ausblick

In dieser Arbeit wurde Time-Sensitive Software-Defined Networking Architektur für den Automobilbereich implementiert und evaluiert. Es wurde ein Algorithmus zur Berechnung gültiger Schedules erstellt und eine Applikation zur Konfiguration von TSSDN-Switches mit der entworfenen Architektur implementiert. Damit die Lösung wiederverwendbar ist, wurden die herstellerunabhängigen Protokolle OpenFlow sowie NETCONF eingesetzt und es wurde ein Treiber als Abstraktionsebene eingeführt. Nachdem die Anforderungen aus dem Automobilbereich ermittelt wurden, konnte eine Architektur zur Umsetzung dieser Anforderungen erstellt werden. Dabei wurden auch diverse Testfälle vorgestellt, um zu prüfen, ob die erarbeitete Lösung und der eingesetzte Switch die Anforderungen bezüglich Bandbreitenverteilung, Latenz, Jitter, Domänenbildung und Priorisierung erfüllen können.
Die Evaluation der Testergebnisse hat gezeigt, dass die entworfene TSSDN-Architektur und der Switch die Anforderungen aus dem Automobilbereich bezüglich Latenz, Bandbreitenverteilung, Priorisierung und Domänenbildung erfüllen. Aufgrund der hohen Dauer, bis Konfigurationsänderungen umgesetzt werden, eignet sich der Switch nicht für den Wechsel vom Fahrzeugmodus in zeitkritischen Situationen, wie etwa während der Fahrt, sondern nur in Werkstätten zur Diagnose oder zum Update. Ein anderer Switch könnte zumindest mit OpenFlow die Änderungen schnell genug umsetzen. Wenn also nur ein Schedule eingesetzt wird, können die Vorteile von SDN in Form von erhöhter Sicherheit und Flexibilität durch dynamische Flüsse zusammen mit den Echtzeit-Vorteilen von TSN ohne Leistungseinbußen von der Automobilindustrie verwendet werden. Die Anforderungen bezüglich des Jitters konnten nicht erfüllt werden, weil ohne Zeitsynchronisierung der sendenden Hosts und des Switches die Zeitschlitze beim Senden oft verfehlt wurden und ein Sender so einen weiteren Durchlauf des Schedules abwarten muss.
Durch den Einsatz von 1 Gbit/s anstelle von 100 Mbit/s Datenübertragungsrate könnten die Werte bezüglich Latenz und Jitter um den Faktor zehn verbessert werden, auch wenn diese Bandbreite nicht von der Datenmenge, die in einem Auto erzeugt wird, erforderlich ist ([18] Kapitel 6.2.3). Das zur Verfügung stehende Oszilloskop kann allerdings lediglich

Datenübertragungsraten von bis zu 100 Mbit/s decodieren, sodass es bei 1 Gbit/s nicht möglich gewesen wäre, den vom Switch eingeführten Anteil von Latenz und Jitter an den Messergebnissen zu ermitteln.
Eine weitere sinnvolle Technologie zur Vermeidung hoher Latenzen und Jitter aufgrund von verfehlten Zeitschlitzen beim Senden ist Frame Preemption, welche aber vom Switch nicht unterstützt wird. Auch eine Verkleinerung der Maximum Transmission Unit (MTU) kann die Latenz und den Jitter verringern. Die in dieser Arbeit implementierte Lösung zur Konfiguration könnte ohne Anpassungen von Frame Preemption oder einer kleineren MTU profitieren und zur Verbesserung der Latenzen und des Jitters durch eine Datenübertragungsrate von 1 Gbit/s müssen lediglich die Zeitschlitze im erarbeiteten Schedule um den Faktor 10 verkürzt werden.
Als weitere Arbeit steht eine Evaluation offen, in der ermittelt wird, ob Frame Preemption, eine Datenübertragungsrate von 1 Gbit/s, eine kleinere MTU, oder eine Kombination aus den Maßnahmen für die Minimierung des Jitters auf die geforderten 60 µs ausreicht oder ob dennoch eine durchgängige Zeitsynchronisierung zwischen sendenden Hosts und dem Switch erforderlich ist, um dieses Ziel zu erreichen.

Literaturverzeichnis

[1] M. Bjorklund. 2010. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).* RFC 6020. IETF.

[2] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. 2011. *Network Configuration Protocol (NETCONF).* RFC 6241. IETF.

[3] Open Networking Foundation. 2016. ONOS System Components. `https://wiki.onosproject.org/display/ONOS/System+Components`. Zugriffsdatum 31.01.2020.

[4] Open Networking Foundation. 2019. Project ONOS. `https://www.opennetworking.org/onos/`. Zugriffsdatum 31.01.2020.

[5] The Open Networking Foundation. 2015. OpenFlow Switch Specification. `https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf`. Zugriffsdatum 24.11.2019.

[6] Timo Häckel, Philipp Meyer, Franz Korf, and Thomas C. Schmidt. 2019. Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication. In *Proc. of the IEEE 89th Vehicular Technology Conference: VTC2019-Spring* (Kuala Lumpur, Malaysia). IEEE Press, Piscataway, NJ, USA, 1–5.

[7] IEEE 802.1 Working Group. 2016. *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption.* Technical Report Std 802.1Qbu-2016. IEEE. 1–57 pages. (Amendment to IEEE Std 802.1Q-2014).

[8] IEEE 802.1 Working Group. 2016. *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic.* Technical Report Std 802.1Qbv-2015. IEEE. 1–57 pages. (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015).

[9] IEEE 802.1 Working Group. 2018. *IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks*. Technical Report Std 802.1Q-2018. IEEE. 1–1993 pages. (Revision of IEEE Std 802.1Q-2014).

[10] IEEE 802.1 Working Group. 2018. *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*. Technical Report Std 802.1Qcc-2018. IEEE. 1–208 pages. (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018).

[11] IEEE 802.3 Working Group. 2018. *IEEE Standard for Ethernet*. Technical Report Std 802.3-2018. IEEE. 1–5600 pages. (Revision of IEEE Std 802.3-2015).

[12] IEEE 802.3 Working Group. 2018. *ISO/IEC/IEEE International Standard - Part 3: Standard for Ethernet - Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)*. Technical Report ISO/IEC/IEEE 8802-3:2017/Amd 1:2017(E). IEEE. 1–92 pages. (Amendment to IEEE Std 802.3-2015).

[13] Rihab Maaloul, Raouia Taktak, Lamia Fourati, and Bernard Cousin. 2018. Energy-Aware Routing in Carrier-Grade Ethernet using SDN Approach. *IEEE Transactions on Green Communications and Networking* 2, 3 (May 2018), 844–858.

[14] Kirsten Matheus and Thomas Königseder. 2017. *Automotive Ethernet* (2 ed.). Cambridge University Press, City of Cambridge.

[15] NoviSwitch. 2018. NoviSwitch 2128 High Performance OpenFlow Switch. `https://noviflow.com/wp-content/uploads/NoviSwitch-2128-Datasheet.pdf`. Zugriffsdatum 18.02.2020.

[16] Paul Pop, Michael Raagaard, Silviu Craciunas, and Wilfried Steiner. 2016. Design Optimization of Cyber-Physical Distributed Systems using IEEE Time-sensitive Networks (TSN). *IET Cyber-Physical Systems: Theory and Applications* 1, 1 (December 2016), 86–94.

[17] R. Serna Oliver, S. S. Craciunas, and W. Steiner. 2018. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)* (Porto, Portugal). IEEE Press, Piscataway, NJ, USA, 13–24.

[18] Till Steinbach. 2018. *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil*. Springer Vieweg, Wiesbaden.

[19] Marian Ulbricht. 2019. TrustNode Short Datasheet. `https://innoroute.com/wp-content/uploads/2019/10/short_ds.pdf`. Zugriffsdatum 24.11.2019.

[20] Marian Ulbricht. 2019. TrustNode YANG Modelle. `https://github.com/InnoRoute/packages/blob/master/TNsysrepo/yangmodel.html`. Zugriffsdatum 30.01.2020.

[21] P. Zhang, Y. Liu, J. Shi, Y. Huang, and Y. Zhao. 2019. A Feasibility Analysis Framework of Time-Sensitive Networking Using Real-Time Calculus. *IEEE Access* 7 (July 2019), 90069–90081.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Implementierung und Evaluation einer Time-Sensitive Software-Defined Networking Architektur für den Automobilbereich

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original