HAW HAMBURG

BACHELORTHESIS
Sandra Christine Reider

Integration von TSN-basierter Kommunikation in eine Netzwerkbeschreibungssprache

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

Sandra Christine Reider

Integration von TSN-basierter Kommunikation in eine Netzwerkbeschreibungssprache

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

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

Eingereicht am: 23.03.2023

Sandra Christine Reider

Thema der Arbeit

Integration von TSN-basierter Kommunikation in eine Netzwerkbeschreibungssprache

Stichworte

TSN, Simulation, Dateigenerierung, ANDL

Kurzzusammenfassung

In modernen Fahrzeugnetzwerken wird verstärkt Ethernet eingesetzt, das Echtzeitanforderungen erfüllen muss, um die Funktionssicherheit des Fahrzeuges zu gewährleisten. Der Time-Sensitive Networking (TSN)-Standard fasst diverse Strategien zur Erfüllung solcher Echtzeitanforderungen zusammen. Die Fahrzeugnetzwerke werden in der Forschung zunächst simuliert. Diese Netzwerke bestehen aus einer großen Anzahl an Netzwerkteilnehmern, zwischen denen sehr viele Nachrichten versendet werden. Da durch diese Komplexität das Erstellen und Konfigurieren aller für die Simulation notwendigen Dateien sehr zeitaufwendig ist, werden diese mit der Abstract Network Description Language (ANDL) automatisch generiert. Die ANDL kann allerdings noch keine Netzwerke mit TSN-basierter Kommunikation generieren. Diese Arbeit befasst sich mit einer Erweiterung der ANDL, um zwei Aspekte des TSN (Gate Scheduling und Credit Based Shaping (CBS)) automatisch zu generieren.

Sandra Christine Reider

Title of Thesis

Integration of TSN-based communication into a network description language

Keywords

TSN, Simulation, File Generation, ANDL

Abstract

In modern vehicle networks, Ethernet is increasingly used, which must meet real-time requirements to ensure the functional safety of the vehicle. The TSN standard outlines strategies for meeting such requirements. Vehicle networks consist of a large number of nodes, between which a large number of messages are sent. Preparing the files necessary to simulate networks of this complexity can be very time and labor intensive, which is why the ANDL was developed to automate the process. However, the ANDL is not yet able to generate networks with TSN-based communication. This thesis addresses this by expanding the functionality of the ANDL to include Gate Scheduling and Credit Based Shaping.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Für Fahrzeuge werden stetig weitere (Komfort-)Features entwickelt und eingesetzt. Dadurch steigen auch die Anforderungen an die internen Fahrzeugnetzwerke und die zur Verfügung stehende Bandbreite in diesen. Die viel verwendete Controller Area Network (CAN)-Bus Technologie trifft dadurch an ihre Grenzen und wird nach und nach durch Ethernet-Netzwerke ersetzt [2]. Um die Sicherheit der Fahrzeuge zu gewährleisten, muss allerdings die Kommunikation vieler Steuergeräte, wie bspw. der Bremssysteme, unter Einhaltung von Echtzeitanforderungen stattfinden [11]. Die neuen Ethernet-Netzwerke müssen daher diesen Echtzeitansprüchen gerecht werden. Dafür wird von der TSN-Task-Group [12] der TSN-Standard entwickelt, der sich aus einer Reihe Substandards zusammensetzt und verschiedene Strategien zur Erfüllung von Echtzeitansprüchen beinhaltet.

Um an neuen Netzwerkaufbauten oder Umsetzungen dieser TSN-Substandards zu forschen, werden die Netzwerke zunächst simuliert. Dies ermöglicht es schnell Änderungen durchzuführen und deren Effekt zu überprüfen. Das Erstellen und Konfigurieren aller Simulationsdateien ist aufgrund der Größe der Automobilnetzwerke sehr zeitintensiv, weshalb eine automatische Generierung dieser Dateien sinnvoll ist.

In der Communication over Real-Time Ethernet (CoRE)-Gruppe erfolgt das automatische Generieren von Netzwerken mit der eigens dafür entwickelten ANDL [3]. Netzwerke, die mit dem TSN-Standard arbeiten, werden allerdings noch nicht von der ANDL unterstützt. Das Ziel dieser Arbeit ist es, eine automatische Generierung TSN-basierter Kommunikation mit der ANDL zu ermöglichen. Insbesondere das Gate Scheduling und CBS sollen implementiert werden. Dadurch können Netzwerke, die diese TSN-Substandards verwenden ebenfalls automatisch generiert und somit leichter an ihnen geforscht werden.

Die Arbeit setzt sich aus acht Kapiteln zusammen. Zunächst werden im Kapitel 2 die für diese Arbeit relevanten theoretischen Grundlagen erläutert. Dazu zählt vor Allem der

TSN-Standard und die verwendete Simulationsumgebung. In Kapitel 3 wird beschrieben, wie die ANDL aufgebaut ist und mit welcher Syntax ein Netzwerk für die automatische Generierung beschrieben werden muss. Anschließend folgt, welche Erweiterungen an der ANDL vorgenommen werden sollen und warum bestimmte Herangehensweisen gewählt wurden, in Kapitel 4. Wie diese Erweiterungen tatsächlich implementiert wurden wird im Kapitel 5 beschrieben. Kapitel 6 beschäftigt sich damit, wie während der Implementierung sichergestellt wurde, dass sie den Anforderungen entspricht, wie erfolgreich die Erweiterungen umgesetzt wurden und an welchen Stellen noch Limitationen existieren. Zum Abschluss folgt das Kapitel 7, in dem ein kurzer Überblick darüber geschaffen wird, was in der Arbeit erreicht wurde und welche Verbesserungen und Erweiterungen zukünftig sinnvoll sein könnten.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Konzepte beschrieben, die für das Verständnis dieser Arbeit benötigt werden. Zunächst wird dafür der TSN-Standard und einige relevante Substandards erläutert. Dann folgt eine Erklärung des verwendeten Frameworks für Netzwerksimulationen.

2.1 Time-Sensitive Networking

Das TSN ist eine Sammlung von Substandards die von der TSN-Task-Group [12] spezifiziert werden. Sie definieren verschiedene Mechanismen, mit denen Echtzeitanforderungen in Ethernet-Netzwerken erfüllt werden können. In den folgenden Unterkapiteln 2.1.1 und 2.1.2 werden zwei dieser Substandards näher betrachtet. Zum einen der IEEE802.1Q-Standard [5], der die Aufteilung eines Netzwerkes in VLANs, sowie das Festlegen von Prioritäten für einzelne Nachrichten definiert. Und zum anderen der IEEE802.1Qbv-Substandard [6], der eine zeitgesteuerte Transmission Selection einführt. Die Erläuterungen dieser Standards basieren auf der Arbeit von Till Steinbach [10].

2.1.1 Priorisierung von Ethernet-Nachrichten nach IEEE 802.1Q

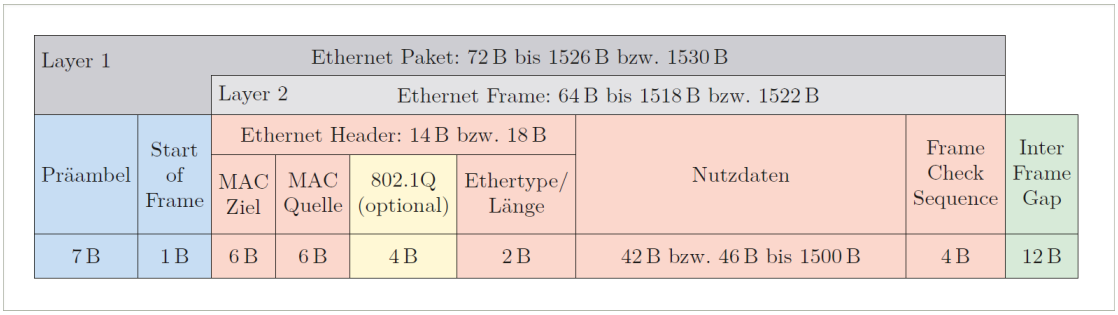

Abbildung 2.1: Aufbau eines Ethernet-Pakets ([10], Abb. 2.6).

Im IEEE802.1Q-Standard [5] wird der Ethernet-Header, wie in Abbildung 2.1 gezeigt, um einen 4 Byte großen Q-Tag erweitert. Dieser Q-Tag setzt sich aus dem 2 Byte großen Tag Protocol Identifier (TPID) und der 2 Byte großen Tag Control Information (TCI) zusammen. Der TPID gibt dabei an, dass es sich um ein Q-Nachrichtenpaket handelt, während sich die TCI in drei weitere Elemente aufteilt. Das erste Element ist der 3 Bit große Priority Code Point (PCP), der es ermöglicht dem Paket eine Priorität zuzuweisen. Das zweite Element ist der Drop Eligible Indicator (DEI), der nur 1 Bit groß ist und angibt, ob das Paket verworfen werden darf, wenn sich der Nachrichtenverkehr aufstaut. Das dritte Element ist der 12 Bit große Virtual Local Area Network (VLAN) Identifier (VID), der die Zuordnung des Pakets zu einem VLAN ermöglicht.

Mit dem PCP kann einem Paket eine Priorität von 0 bis 7 zugewiesen werden, wobei 0 die niedrigste und 7 die höchste Priorität darstellt. Durch diese Prioritäten kann Nachrichtenverkehr, der Echtzeitanforderungen genügen muss, bevorzugt gesendet werden.

Anstelle einer einzigen Warteschlage, in die sämtliche Nachrichtenpakete nach dem First In First Out (FIFO) Prinzip eingeordnet werden, wird für jede der acht Prioritäten eine eigene Warteschlange angelegt. Die Nachrichtenpakete mit gleicher Priorität werden weiterhin nach dem FIFO Prinzip versendet, aber Pakete höherer Prioritäten werden gegenüber Paketen geringerer Prioritäten bevorzugt versendet. Die Wahl des zu sendenden Nachrichtenpakets erfolgt durch einen Scheduler, der wahlweise strikt prioritätsbasiert, gewichtet oder bandbreitenbasiert arbeiten kann. Unabhängig von der Wahl des Verfahrens ist es möglich, dass Nachrichtenpakete niedriger Prioritäten stark verzögert oder gar

nicht gesendet werden, wenn Pakete mit hohen Prioritäten die zur Verfügung stehende Bandbreite zu stark auslasten.

2.1.2 Zeitgesteuerte Transmission Selection nach IEEE 802.1Qbv

Der IEEE802.1Qbv-Standard [6] ist ein Substandard von IEEE802.1Q. Er beschreibt ein Transmission Selection Verfahren, das auf den in Kapitel 2.1.1 beschriebenen Prioritäten aufbaut. Zusätzlich zu den eigenen Warteschlagen erhält jede Priorität einen Transmisi-on Selection Algorithm und ein eigenes Gate, das zeitgesteuert geöffnet oder geschlossen wird. Als Transmission Selection Algorithm kann zwischen mehreren Algorithmen gewählt werden. Dazu gehört ein strikt prioritätsbasierter Algorithmus, in dem immer das Nachrichtenpaket mit der höchsten Priorität ausgewählt wird. Alternativ kann auch ein CBS- oder ein Enhanced Transmission Selection (ETS)-Algorithmus ausgewählt werden. Der CBS wird in Unterkapitel 2.1.2 näher erklärt. Um zu entscheiden, wann welche Gates geöffnet oder geschlossen werden, wird wie in Abbildung 2.2 dargestellt eine GCL verwendet.

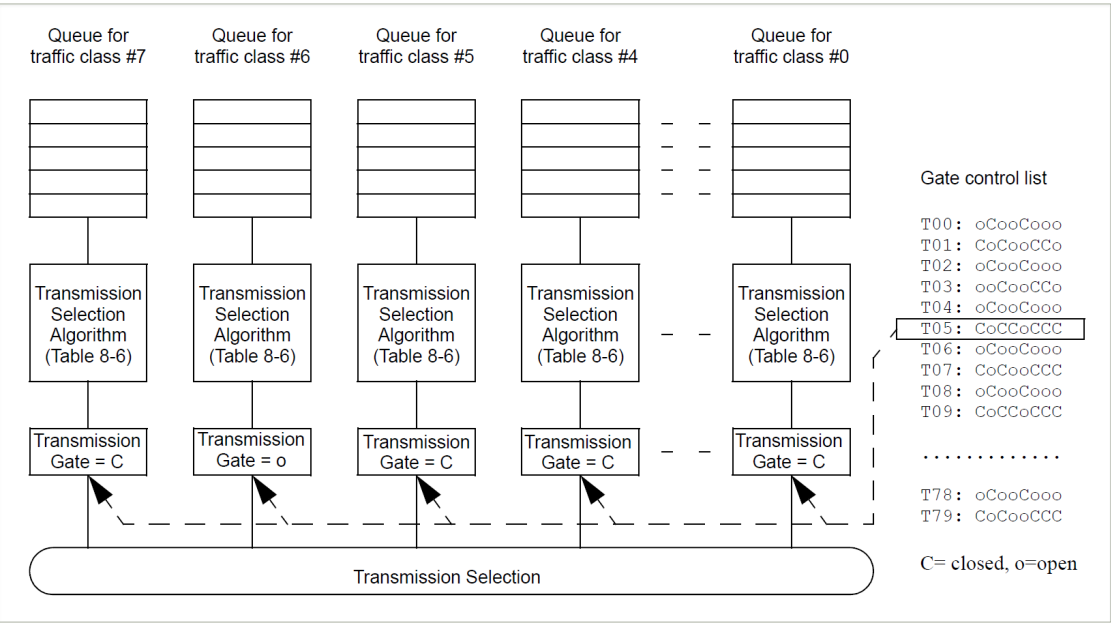

Abbildung 2.2: Transmission Selection mit GCL ([5], Abb. 8-14).

Die GCL listet alle Zeitpunkte innerhalb einer Periode auf, zu denen sich die Zustände der Gates verändern. Für jeden Zeitpunkt werden die Zustände von allen acht Gates gespeichert. Die Zustände können dabei entweder geöffnet (o) oder geschlossen (C) sein. Sobald ein Zeitpunkt aus dieser GCL erreicht wird, werden die Zustände der Gates entsprechend angepasst. Dadurch können zeitkritischem Nachrichtenverkehr ausreichende Sendezeiten zugesichert werden, zu denen keine anderen (niedriger priorisierten) Nachrichten gesendet werden können.

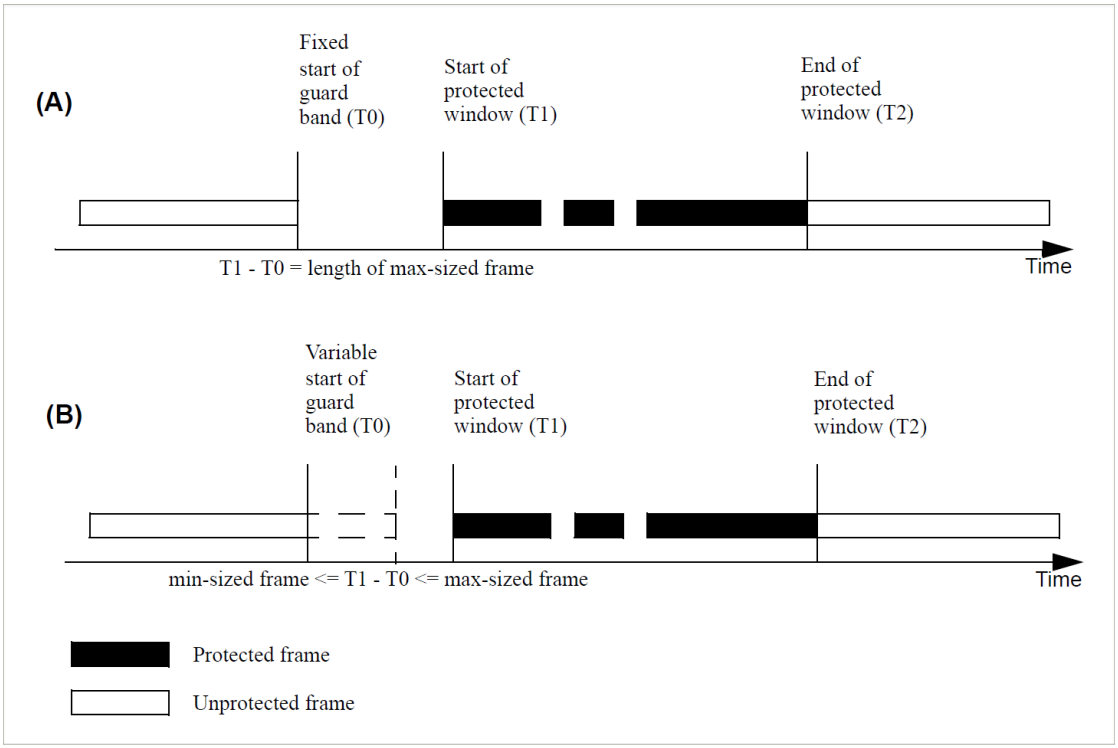

Abbildung 2.3: Mögliche Umsetzungen eines Guardbands vor Sendezeitfenstern von zeitkritischem Nachrichtenverkehr. ([5], Abb. Q-1).

Da das Übertragen von Nachrichtenpaketen nicht abgebrochen werden kann, muss sichergestellt werden, dass alle laufenden Übertragungen abgeschlossen sind, sobald ein Sendezeitfenster für zeitkritischen Nachrichtenverkehr startet. Dies erfolgt, wie in Abbildung 2.3 gezeigt mit einem Guardband. Dieses kann in der einfachsten Ausführung immer der maximalen Größe eines Nachrichtenpakets entsprechen. Alternativ kann dynamisch anhand der Größe des nächsten zu senden Nachrichtenpakets entschieden werden, ob

dieses vollständig übertragen werden kann, bevor das Sendezeitfenster des zeitkritischen Nachrichtenverkehrs beginnt.

Credit Based Shaping

Das CBS kann als Transmission Selection Algorithmus ausgewählt werden. Er wählt die zu sendenden Nachrichtenpakete anhand von Credits der zugehörigen Warteschlange aus. Nachrichtenpakete können dabei nur übertragen werden, wenn deren Warteschlange über eine positive Anzahl von Credits verfügt.

Credits werden von den Warteschlagen gesammelt, wenn drei Kriterien erfüllt sind. Es muss sich mindestens ein Nachrichtenpaket in der Warteschlange befinden und das zugehörige Gate muss geöffnet sein, sodass prinzipiell Nachrichten dieser Warteschlange übertragen werden können. Zusätzlich muss eine andere Übertragung laufen, wodurch die Nachrichtenpakete dieser Warteschlange warten müssen. Sobald ein Nachrichtenpaket dieser Warteschlange gesendet wird, werden die Credits ausgeben. Es ist möglich, dass sich die Anzahl an Credits einer Warteschlange nach einer Übertragung im negativen Bereich befindet. Sollten sich keine Nachrichtenpakete mehr in einer Warteschlange befinden, keine Übertragung aus dieser laufen, diese aber noch über positive Credits verfügen, werden die Credits auf null gesetzt.

Das CBS stellt dadurch sicher, dass der Nachrichtenverkehr einzelner Prioritäten reguliert wird. Dies ermöglicht das Senden von Nachrichtenpakten mit geringerer Priorität, auch bei einem hohen Volumen an Nachrichtenpaketen mit hoher Priorität.

2.2 Framework für Netzwerksimulationen

Das Objective Modular Network Testbed in C++ (OMNeT++) ist ein diskreter Ereignissimulator, der durch weitere Open-Source-Frameworks erweitert werden kann [1]. In Abbildung 2.4 sind die Frameworks dargestellt, die zusätzlich benötigt werden, um Fahrzeugnetzwerke in OMNeT++ zu simulieren. In dem INET-Framework [7] werden einige Standard-Protokolle für Netzwerk-Kommunikation umgesetzt, z. B. Ethernet, das Transmission Control Protocol (TCP) oder das User Datagram Protocol (UDP). Das CoRE4INET-Framework [4] basiert auf dem INET-Framework und erweitert dieses unter anderem um Substandards des in Kapitel 2.1 beschriebenen TSN. Dadurch wird die

Nutzung von Echtzeit-Ethernet in der Simulation ermöglicht. Die beiden Frameworks FiCo4OMNeT [4] und SignalsAndGateways [4] werden benötigt, um Kommunikation mit CAN-Bussen zu simulieren.

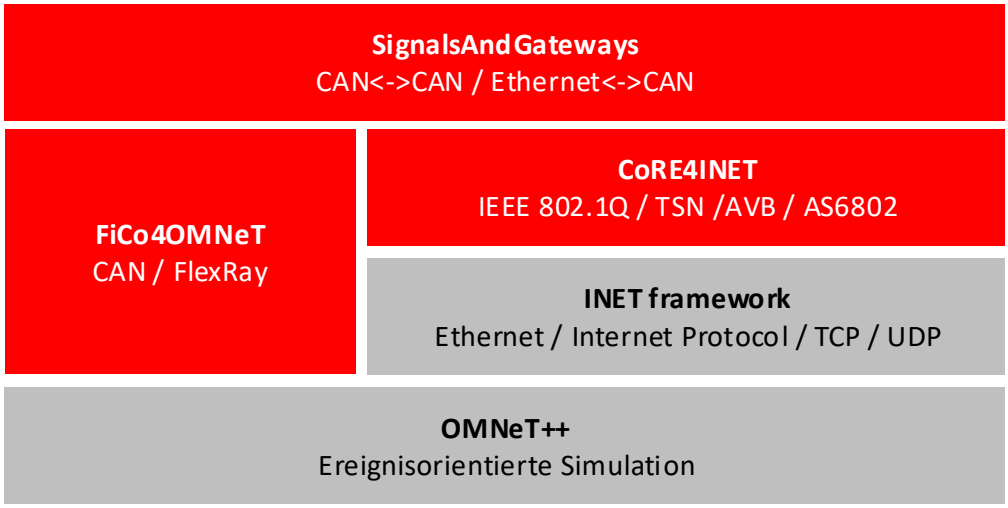

Abbildung 2.4: Für die Simulation von Fahrzeugnetzwerken benötigte Frameworks ([9], Abb. 1).

Für die Simulation eines Netzwerks in OMNeT++ muss für alle Netzwerkteilnehmer eine Network Description (NED) erstellt werden. Diese wird in separaten .ned-Dateien gespeichert und enthält die Beschreibung des Netzwerks oder der Netzwerkteilnehmer mit deren Modultypen, eventuellen Untermodulen und Verbindungen. Hier können ebenfalls Parameter wie Bandbreiten gesetzt werden, sowie Parameter, die bestimmen, wie und wo die Module in der Simulation dargestellt werden. Zusätzlich werden .ini-Dateien benötigt. Diese sind Konfigurationsdateien in denen ebenfalls Parameter gesetzt werden können, aber auch der Simulationsablauf konfiguriert werden kann. Sie konfigurieren unter Anderem die Applications der Nodes, die Nachrichten senden oder empfangen, und deren zugehörigen Parameter, wie Sendeintervalle, Nachrichtengröße und ggf. Prioritäten. Weitere für die Simulation benötigte Dateien (z.B. .xml-Dateien) können ebenfalls in die .ini-Dateien eingebunden werden. Dies ermöglicht es, durch Wahl einer anderen Konfiguration beim Start der Simulation, das gleiche Netzwerk mit anderem Nachrichtenverkehr zu simulieren.

3 Abstract Network Description Language

Die ANDL wurde von der CoRE-Gruppe entwickelt, weil das Erstellen und Anpassen von großen Netzwerken für die Simulation sehr aufwendig ist. Diese ist ein Tool, um die für die Simulation benötigten .ned- und .ini-Dateien automatisch zu generieren. Sie kann einfach als eclipse-plugin in die OMNeT++-Umgebung eingebunden werden.

3.1 Syntax

Das zu generierende Netzwerk wird zunächst in einer einzigen ANDL-Datei beschrieben. Diese enthält Beschreibungen von allen Netzwerkteilnehmern, deren Verbindungen und Nachrichten, die zwischen diesen versendet werden. Um mit der ANDL ein Netzwerk zu generieren, wird dieses zunächst mit allen Teilnehmern und Nachrichten in einer einzigen ANDL-Datei beschrieben.

```
types {...}
settings {...}
network TestNetzwerk {
    devices {...}
    connections {...}
    communication {...}
}
```

Listing 3.1: Aufbau einer ANDL-Datei.

Die ANDL-Datei ist wie in Listing 3.1 dargestellt in vier Abschnitten strukturiert: Die allgemeinen Parameter in **types** und **settings**, die Netzwerkteilnehmer in **devices**, die Verbindungen der Teilnehmer in **connections** und die gesendeten Nachrichten in **communication**.

3.1.1 Types und Settings

```
types tsntypes {
    node TSNNODE {moduleType TSNEtherHost;}
    switch TSNSWITCH {moduleType TSNEtherSwitch;}
}
types links {
    ethernetLink ETH_100MBit {
        bandwidth 100Mb/s;
        length 20m;
    }
}
settings {
    tdmaScheduling true;
}
```

Listing 3.2: Beispiel für den ersten Abschnitt.

Im ersten Abschnitt **types** und **settings** können Typen definiert und allgemeine Netzwerkparameter festgehalten werden. Es ist bspw. möglich, einen Link-Typ mit bestimmter Bandbreite zu definieren oder Typen von Netzwerkteilnehmern, in denen deren Modultypen und/oder weitere Parameter festgehalten werden. Dadurch ist es möglich bestimmte Parameter für alle Links, Nodes etc. einer Art anzupassen, ohne den gesamten Abschnitt zu durchsuchen und ggf. einen zu übersehen.

In den **settings** kann unter Anderem festgelegt werden, ob für das Netzwerk TDMA-Scheduling (Time Division Multiple Access) existiert oder nicht.

3.1.2 Devices

```
devices {
    node node1 {
        moduleType TSNEtherHost;
        inline ini {
            ```
            **.period = "period[1]"
            ```
        }
    }
    node node2 extends tsntypes.TSNNODE {}
    switch switch1 extends tsntypes.TSNETHERSWITCH {}
    ethernetLink link1 extends links.ETH_100MBit;
    ethernetLink link2 extends links.ETH_100MBit;
}
```

Listing 3.3: Beispiel für den zweiten Abschnitt.

Im zweiten Abschnitt **devices** werden die Netzwerkteilnehmer beschrieben. Abhängig von der Art des Teilnehmers können unterschiedliche Parameter gesetzt werden. Für einen CAN-Bus (Controller Area Network) kann bspw. die Bandbreite festgelegt werden und für einen Switch das Hardware-Delay. Für alle Netzwerkteilnehmer gibt es die Möglichkeit über den Parameter **inline ini** weitere Einstellungen zu beschreiben. Die Einstellungen werden unverändert als Konfiguration in die, in Kapitel 2.2 beschriebenen, **.ini**-Datei des jeweiligen Netzwerkteilnehmers geschrieben. Das Netzwerk kann dann in OMNeT++ wahlweise mit oder ohne diese Konfiguration simuliert werden. Dies ermöglicht die Verwendung von neuen Modultypen und Parametern, die in der ANDL noch nicht integriert wurden.

3.1.3 Connections

```
connections {
    segment default {
        node1 <--> link1 <--> switch1;
        node2 <--> link2 <--> switch1;
    }
}
```

Listing 3.4: Beispiel für den dritten Abschnitt.

Im dritten Abschnitt **connections** werden die Verbindungen der Netzwerkteilnehmer beschrieben. Dafür wird festgelegt welche Netzwerkteilnehmer über welche Links miteinander verbunden sind. Diese Zuordnung erfolgt in unterschiedlichen Segmenten, die für den vierten Abschnitt der ANDL relevant sind. Wenn in einem Netzwerk sowohl CAN-, als auch Ethernet-Knoten enthalten sind, kann bspw. ein CAN-Bus- und ein Ethernet-Segment erstellt werden.

3.1.4 Communication

```
communication {
    message msg1 {
        sender node1;
        receivers node2;
        payload 350B;
        period 600us;
        mapping {
            default: q{
                priority 2;
                vid 0;
            };
        }
    }
}
```

Listing 3.5: Beispiel für den vierten Abschnitt.

Im vierten Abschnitt **communication** werden die im Netzwerk ausgetauschten Nachrichten beschrieben. Für jede Nachricht muss neben dem Sender (**sender**) und den Empfängern (**receivers**) die absolute Größe (**payload**) der einzelnen Nachrichtenpakete angegeben werden und in welchem zeitlichen Abstand sie erneut gesendet werden soll (**period**). Die Größe und die Periode können alternativ auch mit einer Gaußschen Normalverteilung angegeben werden. Die Beschreibung jeder Nachricht wird mit dem Parameter **mapping** abgeschlossen.

Im **mapping** wird für jedes Segment aus Abschnitt 3.1.3 (**connections**), über das die Nachricht übertragen wird, festgelegt, ob dies als bspw. CAN, Best Effort, Time-Triggered oder mit Prioritäten erfolgt. Je nach Übertragungsart werden ggf. weitere Parameter festgelegt, z. B. die CAN-ID für CAN-Nachrichten oder die Critical-Traffic-ID

für Time-Triggered Ethernet. Wird eine Nachricht über CAN-Ethernet-Gateways übertragen, müssen diese ebenfalls im **`mapping`** angegeben werden.

3.2 Dateigenerierung

Wenn das Netzwerk fertig beschrieben wurde, kann die Automatische Generierung gestartet werden. Abbildung 3.1 zeigt den groben Zusammenhang der Klassen, die an dieser beteiligt sind.

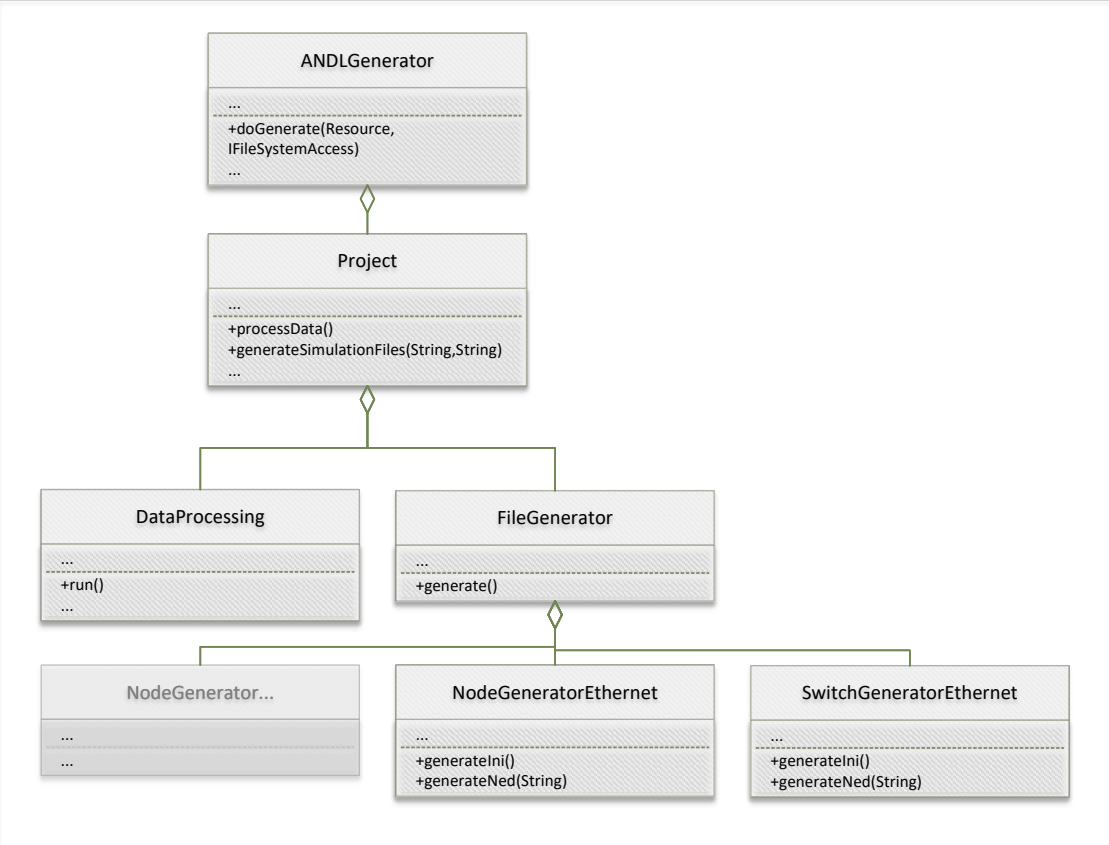

Abbildung 3.1: Klassen der Dateigenerierung.

Zunächst wird die gesamte Datei geparst und alle Netzwerkteilnehmer, Nachrichten und sonstigen Parameter des Netzwerks werden in die Datenstruktur eingelesen. Anschließend wird die `doGenerate(...)`-Methode der Klasse `ANDLGenerator` ausgeführt.

Diese erstellt eine Instanz der Klasse `Projekt` und führt interne Berechnungen, wie bspw. die Erstellung eines Schedules durch. Sie ruft die `processData()`-Methode des Projektes auf, die wiederum die `run()`-Methode der Klasse `DataProcessing` aufruft. Hier werden unter anderem den Nodes randomisierte MAC-Adressen zugewiesen. Im letzten Schritt wird die `generateSimulationFiles(...)`-Methode des Projektes aufgerufen und damit die Generierung der Simulationsdateien gestartet. Hier wird eine Instanz der Klasse `FileGenerator` erstellt, in dessen `generate()`-Methode alle Dateien für die Simulation erstellt werden. Dafür wird in dieser durch die Listen aller Netzwerkteilnehmer iteriert. Um für jeden Netzwerkteilnehmer die benötigten **`.ini`**, **`.ned`** und **`.xml`**-Dateien zu generieren, gibt es für die unterschiedlichen Typen von Netzwerkteilnehmern jeweils eigene Generator-Klassen. Die Generator-Klassen bieten jeweils die beiden public Methoden `generateIni()` und `generateNed()` an, die vom `FileGenerator` aufgerufen werden. Im `GatewayGenerator` gibt es zusätzlich eine `generateXML()`-Methode, die ebenfalls vom `FileGenerator` aufgerufen wird. Diese `generate`-Methoden lesen die für den spezifischen Teilnehmertyp relevanten Informationen aus der Datenstruktur aus, und geben einen String an den `FileGenerator` zurück, der diese Informationen enthält und dem Dateityp entsprechend formatiert ist. Abschließend nutzt der `FileGenerator` den String, um die jeweilige **`.ini`**-, **`.ned`**- oder **`.xml`**-Datei zu erstellen.

4 Konzept

In OMNeT++ können Automobilnetzwerke mit TSN-basierter Kommunikation simuliert werden. Die dafür benötigten Komponenten, wie Nodes oder Switches wurden bereits implementiert. Allerdings ist es derzeit nicht möglich, Netzwerke mit TSN-basierter Kommunikation mit der ANDL zu generieren und zu konfigurieren.

In dieser Arbeit soll eine Erweiterung der ANDL entworfen und implementiert werden, welche die automatische Generierung und Konfiguration von Netzwerkteilnehmern mit TSN-basierter Kommunikation ermöglicht. Die beiden Haupterweiterungen sind dabei das CBS und das Gate Scheduling. Außerdem soll es möglich sein die ANDL um zusätzliche Aspekte TSN-basierter Kommunikation einfach zu erweitern.

Dafür muss festgelegt werden, mit welcher Syntax die TSN-basierte Kommunikation in der ANDL beschrieben wird, wie die Datenstruktur aufgebaut wird, in der die relevanten Daten abgelegt werden und welche Dateien für die Simulation neu oder in abgeänderter Form generiert werden müssen. Auf diese drei Punkte wird in den folgenden Unterkapiteln näher eingegangen.

4.1 Erweiterung der ANDL-Syntax

Die bestehende Syntaxbeschreibung der ANDL soll nach Möglichkeit nicht verändert, sondern nur erweitert werden, sodass alte Netzwerkbeschreibungen ohne Anpassungen weiterverwendet werden können. Um dies sicher zu stellen, werden alle für die TSN-basierte Kommunikation benötigten Parameter in einem eigenen Konfigurations-Block **`tsnConfig`** beschrieben. Dieser neue Block wird wie in Listing 4.1 zu sehen zu Beginn des Abschnitts **`communication`** eingefügt.

```
types {...}
settings {...}
network TestNetzwerk {
    devices {...}
    connections {...}
    communication {
        tsnConfig {...}
        ...
    }
}
```

Listing 4.1: Position des Konfigurationsblocks in der Netzwerkbeschreibung.

Um mit der ANDL Gate Scheduling umsetzen zu können, muss in die Netzwerkbeschreibung aufgenommen werden, welche Nachrichten gescheduled werden sollen und welcher Scheduling-Algorithmus dafür verwendet werden soll. Für die Umsetzung vom CBS muss ebenfalls in die Netzwerkbeschreibung aufgenommen werden, für welche Nachrichten dies erfolgen soll.

Gemäß dem TSN-Standard wird nicht für jeden Nachrichtenstream einzeln entschieden, ob dieser gescheduled oder geshaped werden soll, sondern für eine Prioritätsklasse. Um dies in der ANDL abbilden zu können wird für jeden Nachrichtenstream in dessen Beschreibung eine Q-Priorität nach dem IEEE802.1Q-Standard ausgewählt, welche bereits mit der ANDL-Syntax beschrieben werden kann. Damit die neuen Parameter für das Gate Scheduling und das CBS an die Wahl dieser Q-Priorität geknüpft werden können, wird ein neuer Parameter **`tsnPriority`** eingeführt. Es gibt nach dem IEEE802.1Q-Standard genau acht Prioritäten zwischen 0 und 7 (siehe Kapitel 2.1.1), daher soll jede neue TSN-Priorität bei der Definition eine ID zwischen 0 und 7 erhalten, die automatisch der entsprechenden IEEE802.1Q Priorität zugeordnet wird. Dadurch können maximal acht TSN-Prioritäten definiert werden.

4.1.1 TSN Gruppen

```
tsnConfig {
    group important {
        tsnPriority 2 {}
        tsnPriority 4 {}
        cbs {
            bandwidth 1Mb/s;
        };
    }
    group other {
        tsnPriority 5 {
            cbs {
                bandwidth 700Kb/s;
            };
        }
    }
    ...
}
```

Listing 4.2: Aufbau des ersten Abschnitts des TSN-Konfigurationsblocks.

Der erste Abschnitt des neuen Konfigurationsblocks ist beispielhaft in Listing 4.2 dargestellt. Hier können mehrere TSN-Prioritäten in einer TSN-Gruppe **group** zusammengefasst werden. Diese erhält als ID einen frei wählbaren Namen. In einer TSN-Gruppe können dabei mehrere TSN-Prioritäten definiert werden, jede TSN-Priorität kann aber nur in einer TSN-Gruppe vertreten sein. Da maximal acht TSN-Prioritäten definiert werden können, können auch maximal acht TSN-Gruppen definiert werden. Eine Definition von TSN-Prioritäten außerhalb einer TSN-Gruppe ist nicht möglich.

Das CBS kann entweder für einzelne TSN-Prioritäten oder TSN-Gruppen und damit alle enthaltenen TSN-Prioritäten eingestellt werden. Die Bandbreite wird mit einer Einheit von entweder Bit pro Sekunde oder Kilo-, Mega-, Gigabit pro Sekunde angegeben.

4.1.2 Gate Scheduling

```
tsnConfig {
    ...
    gateScheduling {
        schedulingAlgorithm beispielAlgorithmus {
            ...
        };
    };
}
```

Listing 4.3: Aufbau des zweiten Abschnitts des TSN-Konfigurationsblocks.

In dem zweiten Abschnitt des Konfigurationsblocks kann, wie in Listing 4.3 gezeigt, der neue Block **gateScheduling** definiert werden. Dieser Block beinhaltet den Parameter **schedulingAlgorithm**, mit dem ausgewählt wird, mit welchem Algorithmus das Scheduling erfolgen soll. Algorithmusspezifische Parameter werden ebenfalls hier gesetzt. Die Wahl der TSN-Gruppen, die gescheduled werden, ist ein solcher algorithmusspezifischer Parameter und wird daher hier und nicht bei der Definition der TSN-Gruppen gesetzt. Dadurch kann sichergestellt werden, dass einem Algorithmus nur so viele TSN-Gruppen übergeben werden, wie von diesem erwartet werden.

4.1.3 Zuordnung von Nachrichten zu TSN-Prioritäten

Alle Nachrichten, die bei der Definition im **communication** Abschnitt im **mapping** eine Q-Priorität enthalten, die nicht einer TSN-Priorität entsprechen, werden wie bisher gemäß IEE802.1Q generiert. Für Nachrichten, die mit Gate Scheduling und/oder CBS versendet werden sollen, muss im **mapping** eine Q-Priorität ausgewählt werden, die einer TSN-Priorität entspricht. Für das Gate Scheduling muss bei dessen Definition die zugehörige TSN-Gruppe ausgewählt worden sein. Für das CBS müssen dessen Parameter in dieser TSN-Priorität oder der zugehörigen TSN-Gruppe enthalten sein.

Wird ein Parameter mehrfach gesetzt, wird die Einstellung der direktesten Ebene gewählt: Die Einstellung im **mapping** überschreibt die Einstellung der TSN-Priorität und diese würde die Einstellung der TSN-Gruppe überschreiben. Dadurch können Ausnahmen für einzelne TSN-Prioritäten oder Nachrichten definiert werden.

4.2 Datenstruktur

Die neu eingefügten Parameter müssen in die Datenstruktur aufgenommen werden. Dafür sollen neue TSN-spezifische Klassen erstellt werden, welche die in der Syntax beschriebene Hierarchie der Parameter abbilden. In Abbildung 4.1 ist der Aufbau dieser neuen Klassen dargestellt.

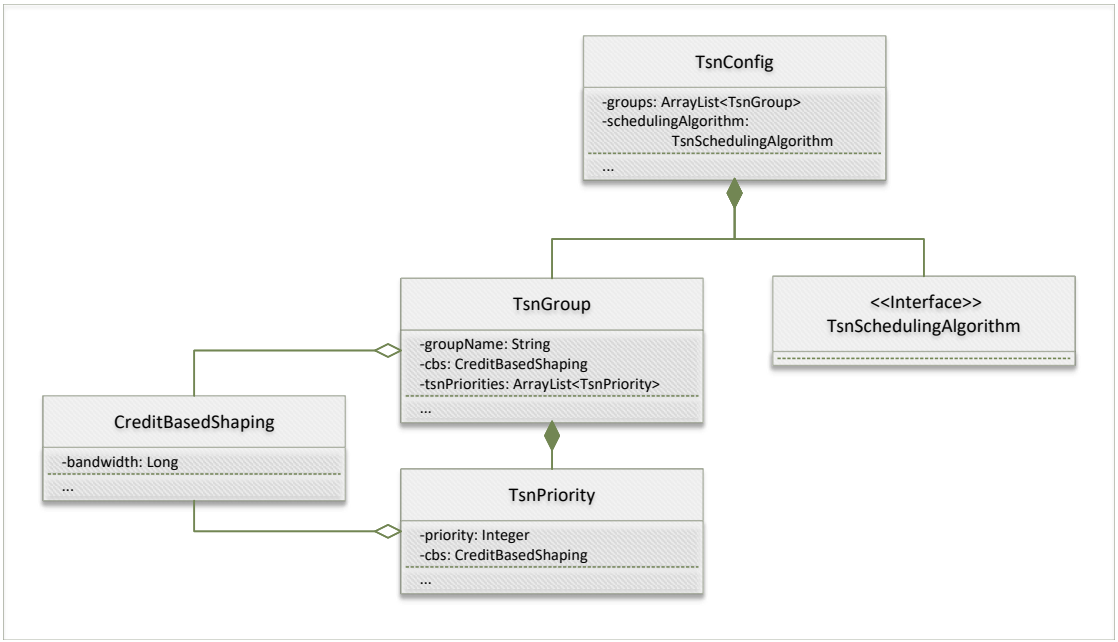

Abbildung 4.1: Aufbau des TSN-Konfigurationsblocks in der Datenstruktur.

Die übergreifende Klasse `TsnConfig` enthält analog zur Syntaxbeschreibung eine Liste von TSN-Gruppen und einen Parameter für den Scheduling-Algorithmus. Dieser Scheduling-Algorithmus muss eine Implementierung des Interfaces `TsnSchedulingAlgorithm` sein. Dadurch ist sichergestellt, dass neue Algorithmen einfach in die Datenstruktur eingebunden werden können. Für die TSN-Gruppen wird die neue Klasse `TsnGroup` erstellt, die wiederum eine Liste von TSN-Prioritäten und einen Parameter für CBS beinhaltet. Die TSN-Prioritäten werden entsprechend in der neuen Klasse `TsnPriority` dargestellt, die neben einem Parameter für die zugehörige IEEE802.1Q-Priorität ebenfalls einen Parameter für CBS enthält.

4.3 Dateigenerierung

In Kapitel 2.2 wurden bereits die für die Simulation benötigten Dateien beschrieben. Für die Einbindung der TSN-basierten Kommunikation in Netzwerkbeschreibungen müssen nun Änderungen an den **.ned**- und **.ini**-Dateien durchgeführt und ggf. neue **.xml**-Dateien erstellt werden. In den **.ned**-Dateien dieser Netzwerkbeschreibungen müssen die Nodes dem Typ **TSNEtherHost** und die Switches dem Typ **TSNEtherSwitch** entsprechen. Die Funktionalität diese Typen auszuwählen und in die **.ned**-Dateien zu schreiben, ist in der ANDL bereits vorhanden. Wie die Konfiguration des TSN-basierten Nachrichtenverkehrs in die **.ini**-Dateien und das Einfügen neuer **.xml**-Dateien umgesetzt werden soll, wird in den Unterkapiteln 4.3.1 und 4.3.2 beschrieben.

4.3.1 Gate Control List

Die GCL muss in der **.ini**-Datei des Switches oder Nodes eingebunden werden. Für die Simulation muss die Angabe in der in Listing 4.4 gezeigten Form erfolgen. Dabei wird zunächst angegeben, für welchen Switch bzw. Node und welchen Port sie berechnet wurde. Anschließend folgt die GCL. Diese setzt sich aus mehreren Segmenten zusammen, die jeweils der Form `x,x,x,x,x,x,x,x:0.000000` entsprechen und durch Semikola getrennt sind. Die acht `x` entsprechen dabei dem Gatezustand der acht Prioritäten in aufsteigender Reihenfolge. Sie können jeweils (C) für geschlossen oder (o) für offen annehmen. Nach dem `:` folgt die Angabe, zu welchem Zeitpunkt innerhalb einer Periode die Gates den beschriebenen Zustand annehmen sollen. Dieser Zeitpunkt wird in Sekunden angegeben.

```
**sw1.phy[1].shaper.gateControlList.controlList =
    "C,C,C,C,C,C,C,C:0.000000;C,C,C,C,C,C,C,o:0.000040;
     o,o,o,o,o,o,o,C:0.000072;C,C,C,C,C,C,C,C:0.000215"
```

Listing 4.4: Beispiel einer GCL in einer ini-Datei (mit Zeilenumbrüchen für die Übersichtlichkeit).

In dem in Listing 4.4 gezeigten Beispiel wird in einer Periode von 300 μs genau ein Nachrichtenpaket mit der Priorität 7 gescheduled. Zu Beginn sind alle Gates geschlossen. Das Gate der Priorität 7 wird bei 40 μs geöffnet. Bei 72 μs wird das Gate wieder geschlossen

und alle anderen Gates für den restlichen Nachrichtenverkehr geöffnet. Um dem in Kapitel 2.1.2 beschriebenen Guardband gerecht zu werden, das in diesem Beispiel 125 μs lang ist, werden alle Gates bei 215 μs geschlossen.

4.3.2 Credit Based Shaping

Die derzeitige Implementierung des CBS in der Simulation wurde für Audio Video Bridging (AVB) entwickelt und verwendet SRP Tables, um die Parameter zu berechnen. Der **`TSNEtherHost`** und **`TSNEtherSwitch`** können diese Implementierung für das CBS verwenden, benötigen im Gegensatz zum **`AVBEtherHost`** und **`AVBEtherSwitch`** allerdings nur die Prioritäten, die geshaped werden sollen, und die jeweilige Bandbreite. Dementsprechend kann eine vereinfachte SRP-Table generiert und verwendet werden.

5 Umsetzung

In diesem Kapitel wird beschrieben, wie die in Kapitel 4 erläuterten Konzepte umgesetzt wurden. Unterkapitel 5.1 befasst sich dabei mit der Syntax der ANDL, in Unterkapitel 5.2 wird darauf eingegangen, wie die Einbindung eines konkreten Scheduling-Algorithmus erfolgt, in Unterkapitel 5.3 wird die endgültige Datenstruktur der TSN-Konfiguration beschrieben und in Unterkapitel 5.4 wird erläutert, wie die GCL und die für das CBS benötigte **`.xml`**-Datei generiert werden.

5.1 Erweiterung der ANDL-Syntax

Die Erweiterungen an der Syntax der ANDL wurden, wie in Kapitel 4.1 beschrieben, umgesetzt. Das Listing 5.1 zeigt an einem Beispiel, wie ein TSN-Konfigurationsblock aussehen kann. An der restlichen Syntax-Beschreibung der ANDL wurden keine Änderungen vorgenommen. Der explizite Scheduling Algorithmus **`basicScheduling`** wird in Unterkapitel 5.2 beschrieben.

```
tsnConfig {
    group important {
        tsnPriority 0 {}
        tsnPriority 1 {}
        cbs {
            bandwidth 1Mb/s;
        };
    }
    gateScheduling {
        schedulingAlgorithm basicScheduling {
            exclusiveWindows important;
        };
    };
}
```

Listing 5.1: Beispiel für die finale Syntax-Beschreibung des TSN-Konfigurationsblocks.

Gemäß der Definition des TSN-Konfigurationsblocks in der ANDL-Syntax ist es nicht möglich in einer Netzwerkbeschreibung einen leeren **`tsnConfig`**-Block zu definieren. Jeder TSN-Konfigurationsblock muss mindestens eine TSN-Gruppe enthalten, ebenso muss jede TSN-Gruppe mindestens eine TSN-Priorität enthalten. Das Gate Scheduling und die Definition von CBS in einer TSN-Gruppe oder TSN-Priorität ist allerdings optional. Jede gewählte ID für die TSN-Gruppen und die TSN-Prioritäten muss einzigartig sein. Die IDs der TSN-Prioritäten müssen zusätzlich einer IEEE802.1Q-Priorität entsprechen.

5.2 Gate Scheduling

Jeder verwendete Algorithmus muss in die Syntaxbeschreibung der ANDL aufgenommen und ggf. benötigte Parameter definiert werden. Aktuell ist der Algorithmus von Jan Kamieth [8] der einzige implementierte Scheduling-Algorithmus. Zukünftig können hier weitere Algorithmen definiert werden.

```
tsnConfig {
    tsnGroup important {...}
    ...
    gateScheduling {
        schedulingAlgorithm basicScheduling {
            exclusiveWindows important;
        };
    };
}
```

Listing 5.2: Das Gate Scheduling mit dem Algorithmus von Jan Kamieth.

Der Algorithmus von Jan Kamieth berechnet individuelle Sendezeitfenster für Nachrichtenstreams und wird, wie in Listing 5.2 gezeigt, als **`basicScheduling`** ausgewählt. Damit der Algorithmus Sendezeitfenster für die Nachrichtenstreams aller TSN-Prioritäten einer TSN-Gruppe berechnen kann, wird der algorithmusspezifische Parameter **`exclusiveWindows`** definiert, in dem diese TSN-Gruppe ausgewählt wird.

Der Algorithmus berechnet die Sendezeitfenster nur für Nachrichten, die intern durch eine `MessageRepresentation` vom Typ `TTRepresentation` dargestellt werden. Eine solche `MessageRepresentation` wird jeder Nachricht beim Einlesen in die Datenstruktur zugeordnet. Sie wird dabei von dem im mapping der Nachricht gewählten Typen bestimmt und speichert die Typ-spezifischen Parameter, wie beispielsweise die Priorität für Q-Nachrichten ab. Die unterschiedlichen Typen von `MessageRepresentations` sind in Abbildung 5.1 abgebildet.

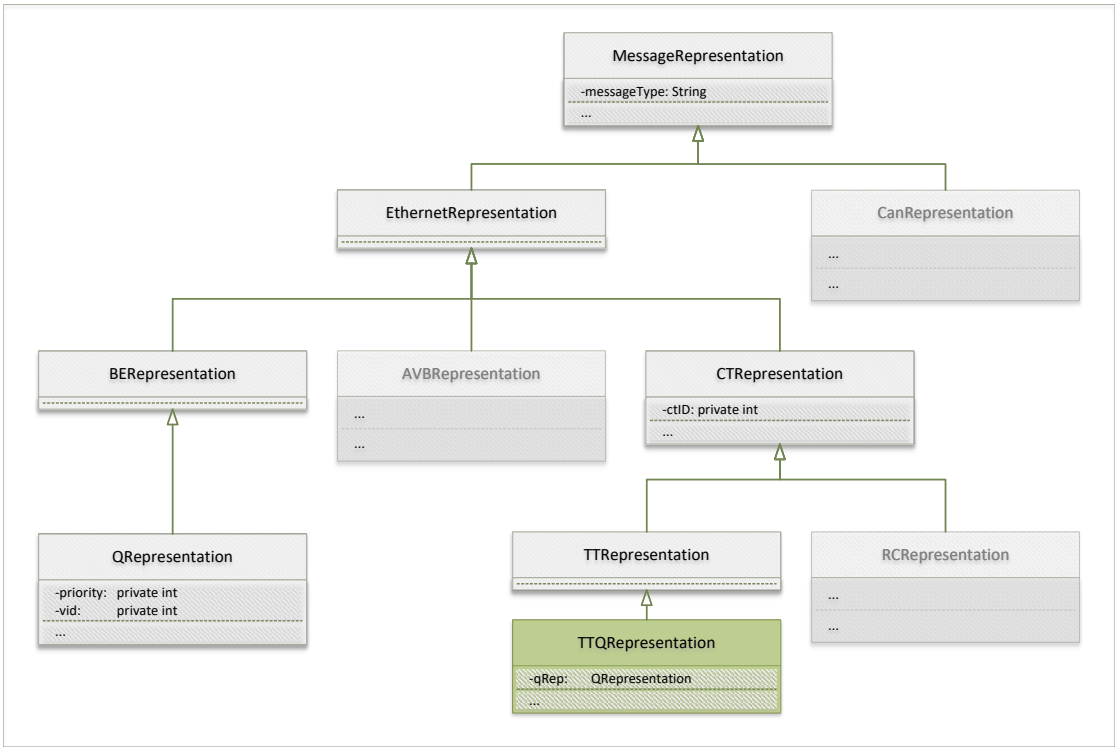

Abbildung 5.1: Vererbung der MessageRepresentations.

Die TSN-basierten Nachrichtenstreams sind an IEEE802.1Q-Prioritäten geknüpft, die durch `QRepresentations` dargestellt werden. Der Scheduler berechnet allerdings nur Sendezeitfenster für Nachrichten, die von einer `TTRepresentation` dargestellt werden. Um ein Scheduling der TSN-basierten Nachrichtenstreams mit dem **`basicScheduling`**-Algorithmus zu ermöglichen, wurde eine `TTQRepresentation` implementiert, die von der `TTRepresentation` erbt. Damit die Parameter der `QRepresentation` in der nachfolgenden Dateigenerierung zur Verfügung stehen, ist diese als Parameter in der neuen `TTQRepresentation` enthalten. Alle Nachrichten mit Q-Prioritäten, die einer TSN-Priorität entsprechen, deren TSN-Gruppe für das Scheduling ausgewählten wurde, erhalten nun beim Einlesen in die Datenstruktur eine `TTQRepresentation` anstelle der `QRepresentation`.

5.3 Datenstruktur

Die Datenstruktur wurde, wie in Kapitel 4.2 beschrieben, umgesetzt. Für den Algorithmus von Jan Kamieth wurde eine Implementierung des `TsnSchedulingAlgorithm`-Interfaces als `BasicSchedulingAlgorithm` erstellt. Diese enthält den Parameter `exclusiveWindowsGroup` und damit die Information, welche Gruppe und damit welche Prioritäten gescheduled werden sollen. Für einen vereinfachten Zugriff auf diese Information wurden ein zusätzlicher Parameter `gateScheduled` und eine Liste aller durch die Gruppe abgebildeten Q-Prioritäten eingefügt.

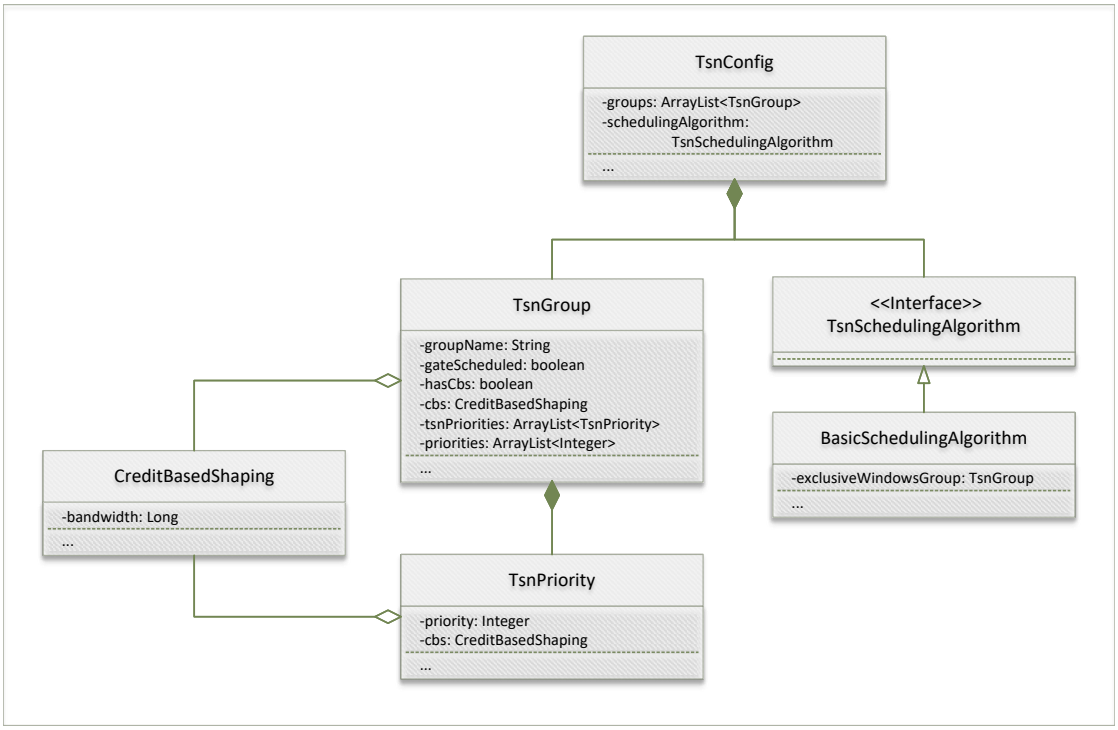

Abbildung 5.2: Aufbau der neuen Klassen für die TSN-Konfiguration.

5.4 Dateigenerierung

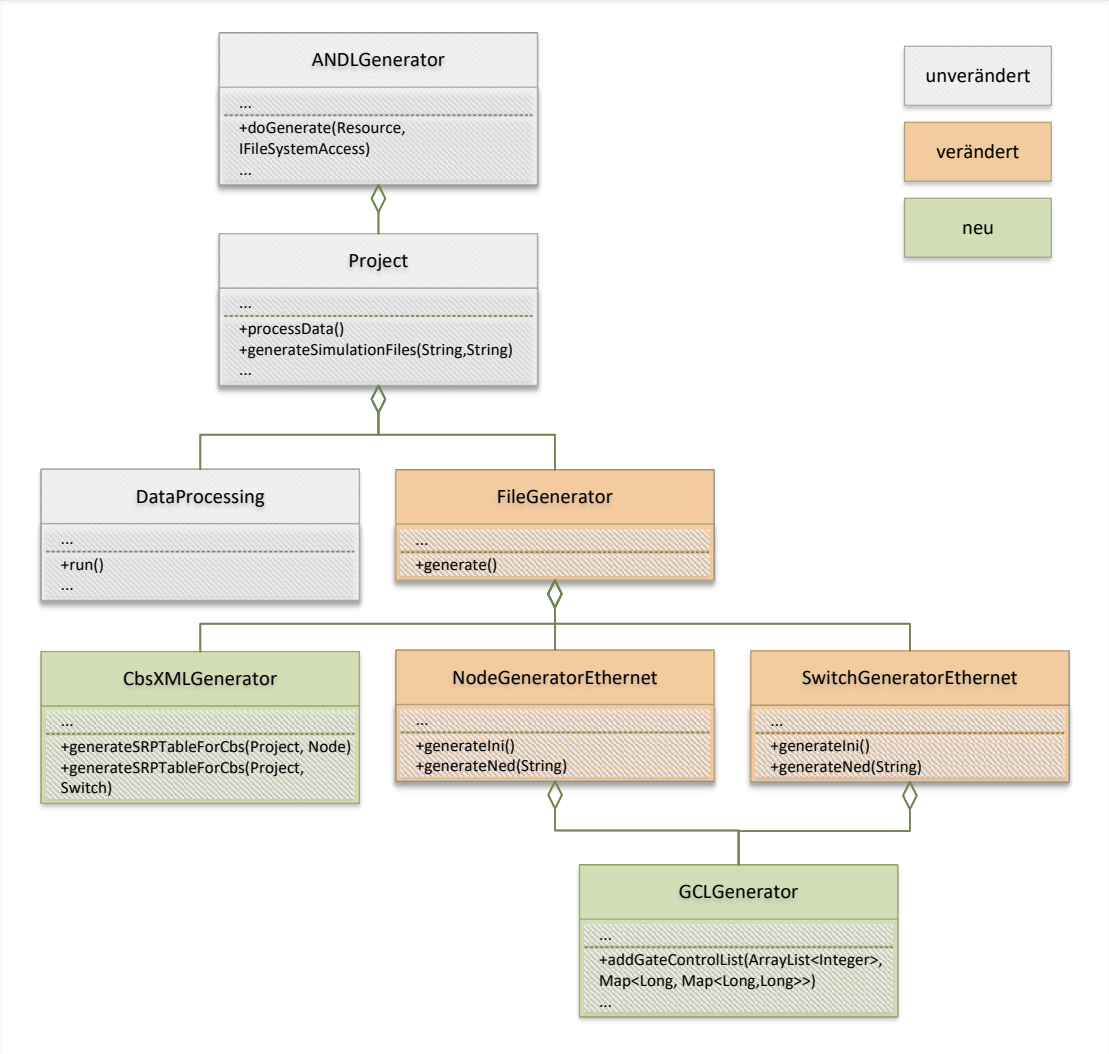

Abbildung 5.3: Klassen der Dateigenerierung mit Änderungen.

In Abbildung 5.3 wird Abbildung 3.1 aus Kapitel 3.2 aufgegriffen und gezeigt an welchen Klassen des Generierungsprozesses Änderungen durchgeführt oder neue Klassen eingebunden wurden. Diese neuen Klassen und die Änderungen an den bestehenden werden in den Unterkapiteln 5.4.1 und 5.4.2 erläutert.

5.4.1 Gate Control List

Die Berechnung der GCL erfolgt in der neu implementierten Klasse `GCLGenerator`. Diese bietet eine statische Methode `addGateControlList(...)` an, die jeweils von den `generateIni()`-Methoden des `NodeGenerators` und des `SwitchGenerators` aufgerufen wird. Dieser Methode wird eine Liste aller gescheduleten Prioritäten übergeben und eine Map, die alle vom Scheduler berechneten Sendezeitfenster enthält. Diese Sendezeitfenster sind nach der zugehörigen Periodenlänge sortiert. Die für die Berechnung der GCL verwendete Hyperperiode entspricht der längsten Periode in dieser Map. Daher müssen alle Perioden von Nachrichten, die ein Gate Scheduling erhalten sollen, ein ganzzahliger Teiler dieser Hyperperiode sein.

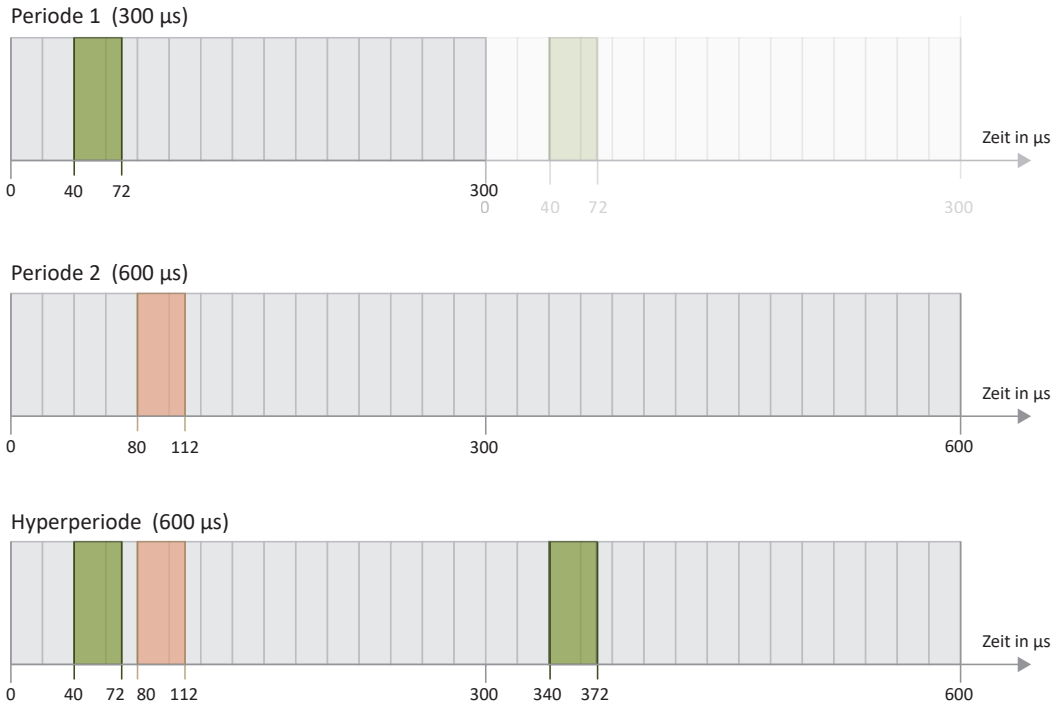

Abbildung 5.4: Bestimmung aller Sendezeitfenster in der gemeinsamen Hyperperiode.

Während eines Durchlaufs der Hyperperiode werden die kürzeren Perioden mehrfach durchlaufen und die entsprechenden Nachrichtenpakete mehrfach gesendet. Um diese zusätzlichen Sendezeitfenster für die Hyperperiode zu berechnen, wird die jeweilige ur-

sprüngliche Periodenlänge zum ursprünglichen Sendezeitfenster addiert. Abbildung 5.4 zeigt dies beispielhaft für zwei Periodenlängen von 300 μs und 600 μs, in denen jeweils ein gescheduletes Nachrichtenpaket versendet werden soll.

Alle Sendezeitfenster werden in aufsteigender Reihenfolge sortiert. In diesem Beispiel beginnt das erste Sendezeitfenster bei 40 μs. Zu Beginn eines Sendezeitfensters sind ausschließlich die Gates geöffnet, die zu den gescheduleten Prioritäten gehören. Da das Gate Scheduling für eine TSN-Gruppe und nicht einzelne TSN-Prioritäten definiert wird, ist dabei nicht relevant, welche Priorität das konkrete Nachrichten-Paket hat. Es werden in der GCL zu Beginn jedes Sendezeitfensters immer alle Gates geöffnet, die zu einer TSN-Priorität dieser TSN-Gruppe gehören. In diesem Beispiel werden Nachrichten mit den Prioritäten 4, 5, und 6 gescheduled. Dementsprechend ist der erste Eintrag in der GCL `C,C,C,C,o,o,o,C:0.000040`. Sobald das Sendezeitfenster endet, werden die Gates der gescheduleten Prioritäten wieder geschlossen.

Wäre der Abstand zum darauffolgenden Sendezeitfenster länger als das auf 125 μs eingestellte Guardband, würden nach dem Ende des jeweiligen Sendezeitfensters alle anderen Gates für den restlichen Nachrichtenverkehr geöffnet werden. Diese Gates würden bis genau 125 μs vor dem Beginn des nachfolgenden Sendezeitfensters geöffnet bleiben.

Da in diesem Beispiel aber das zweite Sendezeitfenster bereits bei 80 μs beginnt, werden die restlichen Gates nicht geöffnet. Die nächsten zwei Einträge in der GCL sind daher `C,C,C,C,C,C,C,C:0.000072` und `C,C,C,C,o,o,o,C:0.000080`. Der Abstand zwischen dem Ende des zweiten und dem Beginn des dritten Sendezeitfensters beträgt 228 μs, und ist somit länger als das eingestellt Guardband. Dadurch werden die Gates für den restlichen Nachrichtenverkehr geöffnet und die GCL wird entsprechend um die Einträge `o,o,o,o,C,C,C,o:0.000112`, `C,C,C,C,C,C,C,C:0.000215` und `C,C,C,C,o,o,o,C:0.000340` erweitert.

Jetzt wird die Hyperperiode mit den drei Sendezeitfenstern wiederholt. Damit beim erneuten Beginn des ersten Sendezeitfensters alle Gates für die Dauer des Guardbands geschlossen sind, müssen sie bereits zum Ende des vorherigen Durchlaufs bei 515 μs geschlossen werden. Listing 5.3 zeigt die vollständig generierte GCL.

```
"C,C,C,C,C,C,C,C:0.000000;C,C,C,C,o,o,o,C:0.000040;
 C,C,C,C,C,C,C,C:0.000072;C,C,C,C,o,o,o,C:0.000080;
 o,o,o,o,C,C,C,o:0.000112;C,C,C,C,C,C,C,C:0.000215;
 C,C,C,C,o,o,o,C:0.000340;o,o,o,o,C,C,C,o:0.000372;
 C,C,C,C,C,C,C,C:0.000515"
```

Listing 5.3: Die vollständige GCL für das Beispiel.

Nachdem die GCL berechnet wurde, wird sie an den jeweiligen `NodeGenerator` oder `SwitchGenerator` zurückgegeben und in die **`.ini`**-Datei eingefügt.

Da für das Gate Scheduling die neue `TTQRepresentation` eingeführt wurde, damit die prioritätsbasierten Nachrichtenstreams gescheduled werden können, wurden für diese TT-spezifische Einträge in der **`.ini`**-Dateien generiert. Dazu zählt u. A. die Critical-Traffic-ID und konkrete Einträge für die Sendezeitfenster, die zusätzlich zum Erstellen und Einfügen der GCL aus der **`.ini`** entfernt werden. In der konkreten Umsetzung wurde für alle TT-spezifischen Einträge geprüft, ob diese auch dem neuen TTQ-Typen entsprechen und dann ggf. der Generierungsabschnitt übersprungen.

5.4.2 Credit Based Shaping

Für das CBS können der **`TSNEtherHost`** und der **`TSNEtherSwitch`** die AVB-Implementierung verwenden, benötigen für diesen Zweck allerdings nicht alle Angaben, die in einer vollständigen SRP-Table enthalten sind. Die korrekte Generierung des CBS benötigt nur die Prioritäten und die für diese reservierten Bandbreiten. Bei den für das CBS generierten SRP-Tables wird deshalb, anstelle einer netzwerkübergreifenden Stream-ID, ein Integer genutzt, die pro Netzwerkteilnehmer für jeden Stream von null hochzählt.

```
1   <srpTable>
2     <talkerTable vlan_id="0">
3       <talkerEntry stream_id="2" srClass="A"
           address="86-18-8C-82-81-18"
           module="network.node1.phy[0]" framesize="125"
           intervalFrames="1" pcp="7"/>
4       <talkerEntry stream_id="2" srClass="A"
           address="FA-5F-FF-C1-C4-AF"
           module="network.node1.phy[0]" framesize="0"
           intervalFrames="1" pcp="7"/>
5     </talkerTable>
6     <listenerTable vlan_id="0">
7       <listeners stream_id="1">
8         <listenerEntry module="network.node1.phy[0]" />
9       </listeners>
10    </listenerTable>
11  </srpTable>
```

Listing 5.4: Eine generierte SRP-Table (mit Zeilenumbrüchen für bessere Lesbarkeit).

In der Simulation wird aus dem durch die `srClass` festgelegten Sendeintervall, der Anzahl an Frames pro Sendeintervall (`intervalFrames`) und der Größe eines Frames (`frameSize`) die Bandbreite pro Priorität ermittelt. Bei der Generierung der SRP-Table müssen entsprechend diese Parameter aus der Bandbreite bestimmt werden. Die Berechnung erfolgt gemäß Formel (5.1).

$$frameSize = \frac{Bandbreite * Sendeintervall}{intervalFrames} \tag{5.1}$$

Das Sendeintervall wird durch die `srClass` bestimmt und wird auf 125 μs festlegt, indem ausschließlich die `srClass 'A'` verwendet wird. Für die Berechnung der Bandbreite in der Simulation ist nur die Summe aller `frameSizes` einer Priorität relevant. Diese Summe kann sich beispielsweise aus der `frameSize` eines großen Frames eines einzigen Streams oder den `frameSizes` mehrerer Frames unterschiedlicher Streams zusammensetzen. Zur Vereinfachung wird pro Priorität für den ersten Stream genau ein Frame in der SRP-Table eingetragen, dessen `frameSize` anhand der Formel (5.1) aus

der in der ANDL definierten Bandbreite berechnet wird. Alle anderen Streams werden in der SRP-Table mit einer `frameSize` von '0' aufgeführt.

Die Generierung der SRP-Table erfolgt in den `generateSRPTableForCbs(...)`-Methoden der neu implementierten Klasse `CbsXMLGenerator`. Es gibt je eine Methode für die Generierung der SRP-Table für einen Switch und einen Node. Diese Methoden benötigen das Project und den entsprechenden Node oder Switch, für den die SRP-Table generiert werden soll. Sie werden in der `generate()`-Methode des `FileGenerators` aufgerufen und erstellem dann die SRP-Table in einem formatierten String, der an die `generate()`-Methode zurück gegeben wird. In der `generate()`-Methode wird der String in eine neue **`.xml`**-Datei gespeichert. Damit diese **`.xml`**-Datei in der Simulation genutzt werden kann, muss sie in die **`.ini`**-Datei des jeweiligen Nodes oder Switches eingebunden werden. Listing 5.5 zeigt, wie diese Einbindung erfolgt.

```
1   **.node1.**.srpTableFile = xmldoc("node1_SRPTable.xml")
```

Listing 5.5: Beispielhafte Einbindung einer .xml-Datei in eine .ini-Datei.

6 Qualitätssicherung und Evaluation

In diesem Kapitel wird betrachtet, ob die Ziele der Arbeit erfüllt wurden. Unterkapitel 6.1 erklärt, wie während der Implementierung die Funktionalität dieser geprüft wurde. Dann folgt die Simulation eines konkreten Demo-Netzwerkes in Unterkapitel 6.2 und abschließend eine Einschätzung der Limitationen der aktuellen Implementierungen.

6.1 Qualitätssicherung

Zur Sicherstellung, dass die Ziele dieser Arbeit erreicht wurden, wurde begleitend zur Implementierung die korrekte Dateigenerierung geprüft und abschließend die Funktionalität getestet.

Während des Einfügens des **`tsnConfig`**-Blocks und der damit verbundenen Parameter in die Syntaxbeschreibung der ANDL wurden Netzwerkbeschreibungen mit diesen Parametern erstellt und überprüft, ob die neuen Parameter fehlerfrei in die Beschreibung aufgenommen werden können. Es wurde ebenfalls getestet, ob bei der Eingabe von doppelten oder falschen Bezeichnern, beispielsweise einer zehn für die **`tsnPriority`**, Fehlermeldungen angezeigt werden.

Bevor mit der Erweiterung der automatisierten Dateigenerierung der ANDL um TSN-spezifischen Einträge begonnen wurde, wurden einfache Beispielnetzwerke ausgewählt, die bereits in der Simulation liefen und bestimmte Aspekte TSN-basierter Kommunikation umsetzen.

Zusätzlich wurden neu generierte Netzwerke mit diesen Beispielnetzwerken verglichen, um zu überprüfen, ob sie korrekt generiert wurden. Wenn Abweichungen entdeckt wurden, wurde entsprechend evaluiert, wodurch diese entstanden sind, und die Fehler behoben. Dadurch wurde sichergestellt, dass alle benötigten Parameter mit den richtigen Werten und an den richtigen Stellen eingefügt wurden. Dieser Prozess wurde während

der gesamten Implementierung begleitend durchgeführt und für jeden Aspekt, der implementiert wurde, wurden passende Netzwerke verwendet.

Nachdem die automatische Generierung der GCL und des CBS implementiert war und die generierten Netzwerke mit den manuell erstellten Beispielnetzwerken übereinstimmten, wurden die generierten Netzwerke in der Simulation gestartet und getestet. Im folgenden Unterkapitel wird die Funktionalität des implementierten Gate Schedulings anhand eines Demo-Netzwerkes gezeigt.

6.2 Beispielsimulation

Um die Funktionalität der implementierten GCL zu testen, wurde ein Demo-Netzwerk mit 5 Nodes und 3 Switchen generiert (Abbildung 6.1). Die Nachrichten, die zwischen den Nodes versendet werden, sind in Tabelle 6.1 aufgelistet. Alle Nachrichten werden mit einer Periode von 300 μs und einer Payload von 350 *Byte* versendet. In dem Demo-Netzwerk werden Nachrichten mit den Prioritäten 5, 6 und 7 gescheduled und entsprechend GCLs für die Nodes 1, 3 und 5 sowie die drei Switches generiert.

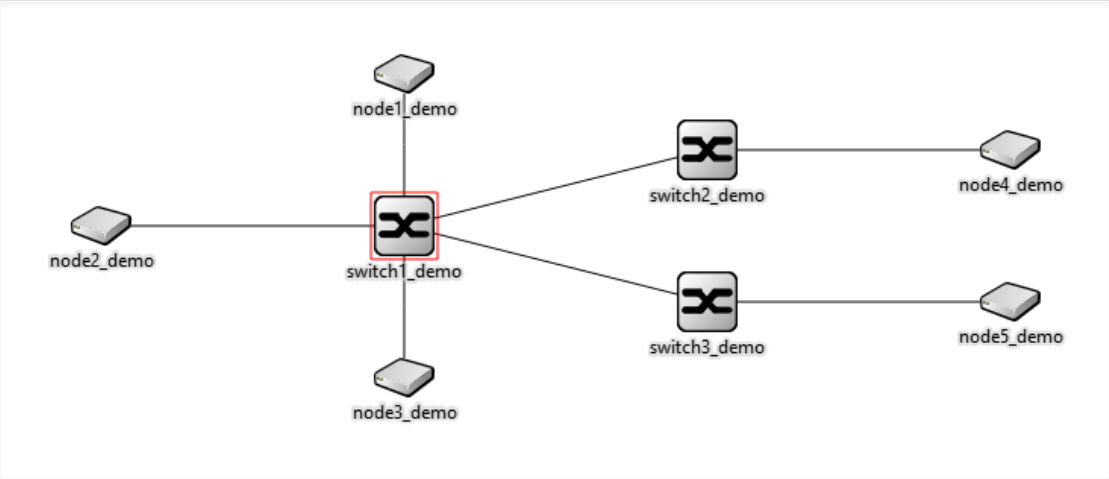

Abbildung 6.1: Darstellung des Demo-Netzwerkes in der Simulation.

Nachricht	Sender	Empfänger	Priorität	Sendezeitpunkt [μs]
msg1	node1_demo	node2_demo	7	0
msg2	node1_demo	node3_demo	6	40
msg3	node3_demo	node2_demo	7	0
msg4	node3_demo	node2_demo	7	40
msg5	node5_demo	node3_demo	7	0
msg6	node5_demo	node2_demo	7	40
msg7	node5_demo	node3_demo	3	-
msg8	node4_demo	node2_demo	3	-
msg9	node1_demo	node3_demo	2	-
msg10	node3_demo	node5_demo	4	-

Tabelle 6.1: Tabelle aller im Demo-Netzwerk gesendeter Nachrichten.

Das Netzwerk wurde für 3 s simuliert und die Ende-zu-Ende-Latenzen aufgezeichnet der Nachrichten aufgezeichnet. Bei einer Simulationsdauer von 3 s und einer Periode von 300 μs, wird die Periode 10000-mal durchlaufen.

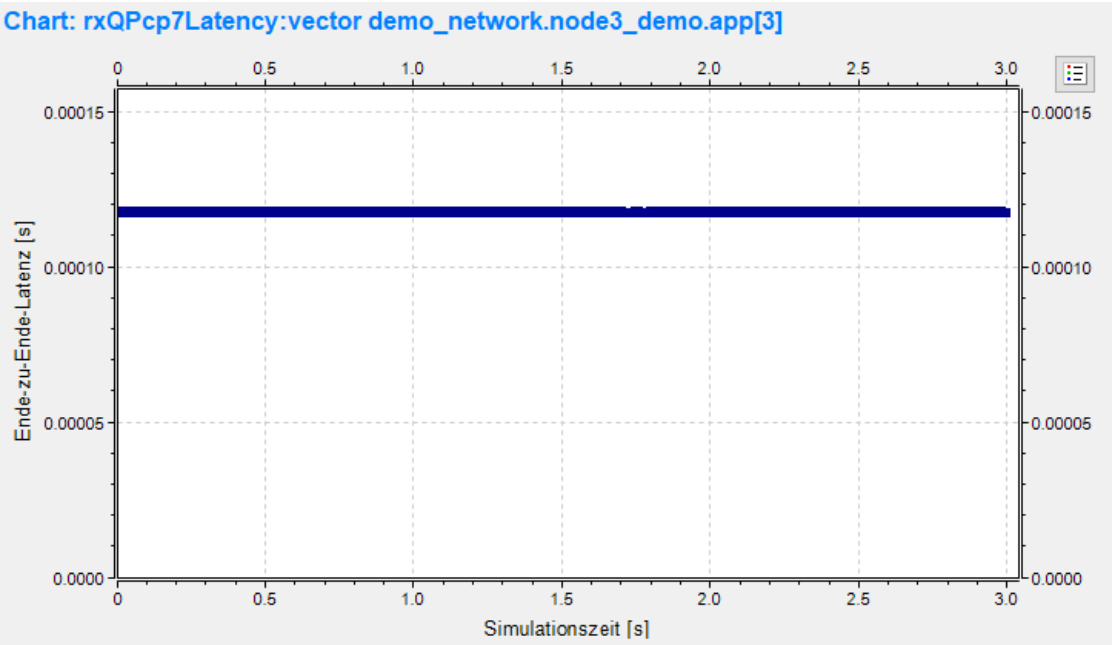

Abbildung 6.2: Die Ende-zu-Ende-Latenz von Nachricht 5.

Die Abbildung 6.2 zeigt die Ende-zu-Ende-Latenz für Nachricht 5. Diese liegt im Durchschnitt bei $117,6\ \mu s$ und erreicht einen Maximalwert von $118,1\ \mu s$.

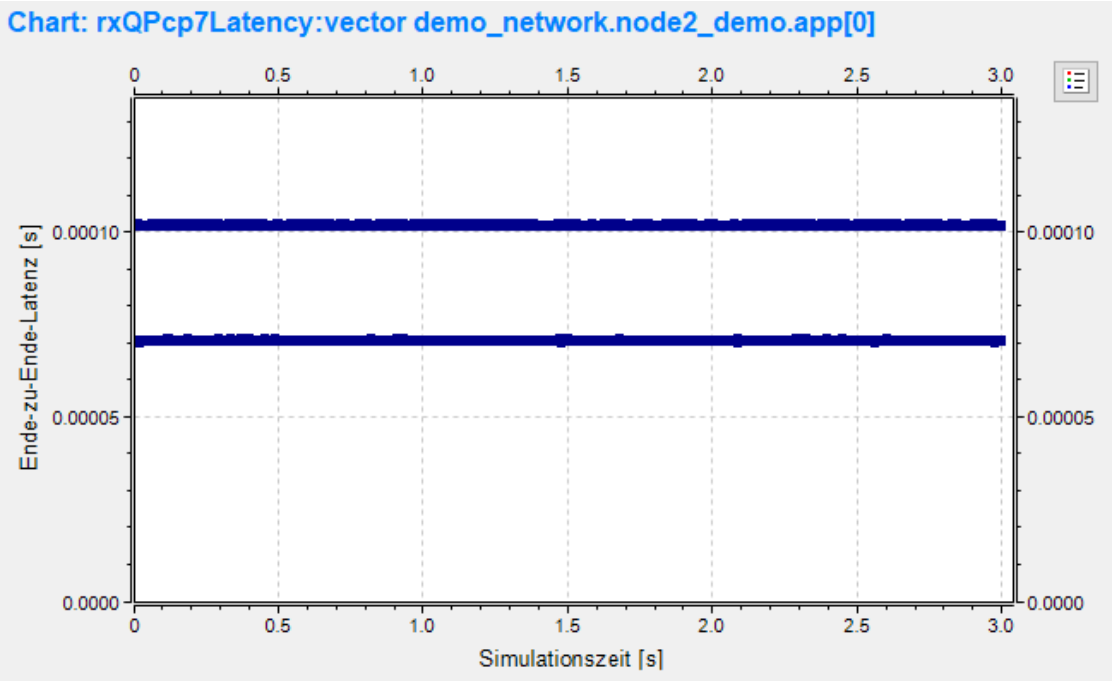

Abbildung 6.3: Die Ende-zu-Ende-Latenz von Nachricht 1.

In Abbildung 6.3 wird die Ende-zu-Ende-Latenz von Nachricht 1 gezeigt. Es ist zu sehen, dass die empfangenen Nachrichten jeweils einer von zwei Ende-zu-Ende-Latenzen entsprechen. Diese liegen bei ca. 77 μs und 102 μs. Dies könnte dadurch erklärt werden, dass die berechneten Sendezeitfenster sehr dicht beieinander liegen. So kann es sein, dass ein Nachrichtenpaket direkt zu Beginn des eigenen Sendezeitfensters gesendet wird und im Switch noch durch ein früheres Sendezeitfenster eines anderen Nachrichtenpakets übertragen werden kann. Hier müssten weitere Tests durchgeführt werden, um herauszufinden, ob diese Theorie stimmt und mit einer anderen Konfiguration des Scheduling-Algorithmus das Problem umgangen werden kann.

6.3 Evaluation

Die automatische Generierung von GCLs für Gate Scheduling in der Simulation wurde erfolgreich umgesetzt. Die GCLs wurden dabei anhand von Sendezeitfenstern berechnet, die von dem Algorithmus von Jan Kamieth berechnet wurden. Anschließend wurden sie entsprechend der Anforderungen der Simulation formatiert und in die jeweiligen `.ini`-Dateien eingebunden.

Das manuelle Erstellen der GCLs kann mit steigender Anzahl an gescheduleten Nachrichten sehr zeitaufwendig und auch fehleranfällig werden. Zum einen, weil für jeden Port an jedem Node oder Switch, über den der geschedulete Nachrichtenverkehr gesendet wird, eine eigene GCL erstellt werden muss. Und zum anderen weil die einzelnen GCLs entsprechend sehr lang und unübersichtlich werden können. Bei mehreren unterschiedlichen Perioden wird außerdem das Berechnen der Hyperperiode und aller in dieser enthaltenen Sendezeitfenster für die GCL komplex.

Die automatische Generierung stellt daher eine drastische Vereinfachung dar und ermöglicht die realistische Nutzung von Gate Scheduling in der Simulation größerer Netzwerke. Sie ermöglicht außerdem die Nutzung des implementierten Scheduling-Algorithmus. Dadurch kann auch der Schedule automatisch berechnet werden, auf dessen Basis die GCL erstellt wird.

Außerdem wurde das Generieren und Einbinden von SRP-Tables für die Nutzung von CBS automatisiert. Auch hier entsteht besonders bei größeren Netzwerken ein hohes Fehlerpotential bei der manuellen Erstellung, da für jeden Switch und Node, der geshapete Nachrichten-Pakete empfängt oder sendet eine SRP-Table erstellt werden muss. Dafür muss nicht nur ermittelt werden, welche Nachrichten-Streams von dem jeweiligen Node oder Switch empfangen oder gesendet werden, sondern auch mit welchen VIDs und über welchen Port dies erfolgt. Für Nachrichten-Streams, die gesendet werden muss zusätzlich ermittelt werden, an welche Empfänger-Adressen diese gesendet werden und welche Frame-Größe zulässig ist.

Bei der Simulation eines generierten Netzwerkes ist aufgefallen, dass bei einigen generierten GCLs zwei Einträge für den Zeitpunkt 0 enthalten sind. Dies führt dazu, dass bei jedem erneuten Periodendurchlauf einer der beiden Einträge ausgewählt wird. Dadurch kann es passieren, dass die Gates für ein berechnetes Sendezeitfenster nicht geöffnet werden und sich die Nachrichten in der entsprechenden Warteschlange aufstauen.

Die doppelten Einträge entstehen in der Generierung dadurch, dass für jede GCL zunächst alle Gates geschlossen werden. Dies sollte ursprünglich sicherstellen, dass ein Eintrag für den Zeitpunkt 0 und damit den Simulationsbeginn existiert und ein ggf. benötigtes Guardband, das am Ende der Periode startet, eingehalten wird. Allerdings wurde keine Überprüfung implementiert, ob direkt zum Periodenbeginn bereits ein Sendezeitfenster startet. Durch ein Löschen des ggf. doppelten ersten Eintrags, lässt sich ein Netzwerk dennoch korrekt simulieren. Die automatische Generierung der GCL sollte um eine entsprechende Abfrage erweitert werden, damit zukünftig keine Überprüfung der generierten GCLs mehr notwendig ist.

6.3.1 Limitationen

Während des Testens von Demo-Netzwerken in der Simulation sind einige Limitationen der aktuellen Implementierung von Sende-Applications, des Scheduling-Algorithmus und der neuen GCL aufgefallen:

Für das Gate Scheduling werden synchronisierte Sende-Applications verwendet, die Nachrichten nach IEEE802.1Q versenden. In der Implementierung dieser Sende-Applications muss ein Parameter `actionTime` gesetzt werden. Dieser gibt an, zu welchem Zeitpunkt in der Periode die Nachricht gesendet wird. Wenn allerdings eine größere Hyperperiode gesetzt wird, und dementsprechend mehrere Nachrichten in einem Periodendurchlauf gesendet werden müssen, müssten mehrere `actionTimes` definiert werden. Dies wird von der aktuellen Implementierung jedoch nicht unterstützt.

Die Bestimmung der Hyperperiode, die für die Generierung der GCL verwendet wird, erfolgt zur Zeit für jeden Port einzeln. Dadurch kann es passieren, dass zwei GCLs an unterschiedlichen Ports des gleichen Switches unterschiedliche Hyperperioden verwenden. Diese müssen in den einzelnen **`.ini`**-Dateien für die jeweiligen Ports gewählt werden. Es wäre daher sinnvoll, die Hyperperiode einmalig für das gesamte Netzwerk zu bestimmen und für alle GCLs zu verwenden.

Der Schedule berechnet nur ein Sendezeitfenster pro Nachricht, auch wenn diese als Multicast versendet wird. Da bei einem Multicast in der Simulation aber für jeden Empfänger eine eigene Sende-Application erstellt wird, müsste auch für jeden Empfänger ein Sendezeitfenster berechnet werden. Mit der aktuellen Implementierung ist das Generieren von Gate Scheduling in den Netzwerken nicht multicast-fähig.

7 Fazit

Zu Beginn dieser Arbeit wurde die ANDL vorgestellt, mit der automatisch Dateien generiert werden können, die für die Simulation von (Automobil-)Netzwerken benötigt werden. Diese wurde erfolgreich so erweitert, dass eine automatisch Generierung von Netzwerken mit TSN-basierter Kommunikation ermöglicht wird. Dafür wurde zunächst ein neuer Konfigurationsblock in die Syntax-Beschreibung aufgenommen und in eine Datenstruktur eingebunden. Spezifisch wurden Gate Scheduling und CBS implementiert.

Für das Gate Scheduling wurde die Berechnung einer GCL umgesetzt. Diese verwendet Sendezeitfenster, die von einem Scheduling Algorithmus berechnet werden. Die GCLs werden während der Generierung von den `.ini`-Dateien der Netzwerkteilnehmer in diese eingebunden.

Aktuell müssen alle Perioden ein ganzzahliger Teiler der größten Periode sein, da diese als Hyperperiode verwendet wird. Zukünftig könnte eine automatische Berechnung einer Hyperperiode implementiert werden, um bei der Beschreibung des Nachrichtenverkehrs eines Netzwerks mehr Flexibilität zu erhalten.

Für die Berechnung des Schedules steht zur Zeit der Algorithmus von Jan Kamieth [8] zur Verfügung. Da sowohl die neue Syntax als auch die Implementierung der neuen Datenstruktur eine einfache Erweiterbarkeit unterstützen, können zukünftig weitere Algorithmen mit anderen Schedulingstrategien mit geringerem Aufwand implementiert werden. So könnte beispielsweise ein Algorithmus implementiert werden, der nur jeweils ein Sendezeitfenster pro Priorität statt einem pro Nachrichtenpaket berechnet.

Literaturverzeichnis

[1] : *OMNeT++*. – URL https://omnetpp.org/. – Zugriffsdatum: 2023-03-22

[2] Brunner, Stefan ; Roder, Jurgen ; Kucera, Markus ; Waas, Thomas: Automotive E/E-Architecture Enhancements by Usage of Ethernet TSN. In: *2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES)* IEEE (Veranst.), 2017, S. 9–13

[3] CoRE-Arbeitsgruppe: *Abstract Network Description Language*. – URL https://core-researchgroup.de/projects/simulation/andl/. – Zugriffsdatum: 2023-03-22

[4] CoRE-Arbeitsgruppe: *CoRE Simulation Models for Real-time Networks*. – URL https://core-researchgroup.de/projects/simulation/. – Zugriffsdatum: 2023-03-22

[5] IEEE 802.1 Working Group: IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks / IEEE. Juli 2018 (Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)). – Standard. – 1–1993 S

[6] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic / IEEE. März 2016. – Standard. – 1–57 S

[7] Internet Technologies Group: *INET Framework*. – URL https://inet.haw-hamburg.de/. – Zugriffsdatum: 2023-03-22

[8] Kamieth, Jan: *Scheduling von TDMA Kommunikation in Switch basierten Netzwerken*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, mastersthesis, Januar 2015

[9] Reider, Sandra ; Meyer, Philipp ; Häckel, Timo ; Korf, Franz ; Schmidt, Thomas C.: Integration realer Angriffe in simulierte Echtzeit- Ethernet-Netzwerke.

In: *Echtzeit 2020*. Wiesbaden : Springer Vieweg, Januar 2021 (Informatik aktuell), S. 51–60. – ISBN 978-3-658-32818-4

[10] Steinbach, Till: *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil*. Wiesbaden : Springer Vieweg, Oktober 2018. – ISBN 978-3-658-23499-7

[11] Steinbach, Till ; Lim, Hyung-Taek ; Korf, Franz ; Schmidt, Thomas C. ; Herrscher, Daniel ; Wolisz, Adam: Beware of the Hidden! How Cross-traffic Affects Quality Assurances of Competing Real-time Ethernet Standards for In-Car Communication. In: *2015 IEEE Conference on Local Computer Networks (LCN)*, Oktober 2015, S. 1–9. – LCN Best Paper Award. – ISBN 978-1-4673-6770-7

[12] TSN Task Group: . – URL https://1.ieee802.org/tsn/. – Zugriffsdatum: 2023-03-22

Erklärung zur selbstständigen Bearbeitung

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