HAW HAMBURG

Bachelorarbeit

Michael Schmidt

Ein AVB-Stack für Mikrocontroller basierte automotive Anwendungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Michael Schmidt

Ein AVB-Stack für Mikrocontroller basierte automotive Anwendungen

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

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

Eingereicht am: 05. Oktober 2018

Michael Schmidt

Thema der Arbeit

Ein AVB-Stack für Mikrocontroller basierte automotive Anwendungen

Stichworte

Echtzeit-Ethernet, Fahrzeug-Netzwerke, TTEthernet, Audio/Video-Bridging, Netzwerk-Stack

Kurzzusammenfassung

Die Anzahl und Komplexität heute verbauter elektronischer Endgeräte nimmt stetig zu und traditionelle Bussysteme stoßen an ihre Grenzen. Eine Lösung ist das Audio/Video-Bridging Protokoll. Es ermöglicht eine dynamische Konfiguration. Diese erlaubt es während des Betriebs Netzwerkknoten hinzuzufügen oder zu entfernen. Diese Arbeit bietet ein Umsetzungskonzept und dessen Implementierung für einen AVB-Stack. Die Implementierung mit Tests evaluiert und auftretetende Probleme werden besprochen. . . .

Michael Schmidt

Title of Thesis

An AVB stack for micro controller based automotive applications

Keywords

realtime ethernet, in-vehicle networks, TTEthernet, Audio/Video Bridging, network stack

Abstract

The amount and complexity of electronics intregrated in modern vehicles is steadily increasing and traditional bus systems reaching their limits. One solution is the Audio/Video Bridging Protocol. It allows the dynamic configuration, and adding and removing of nodes while running. A concept for an AVB stack and its implementation will be elaborated. The implementation will be tested and possibly occuring problems will be discussed. . . .

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einführung

Abbildung 1.1: Vernetzung von elektronischen Endgeräten und derren Bussystem in einem modernem Oberklasse-Fahrzeug. (Quelle: [1])

Seit der Einführung des Mercedes-Benz W140, als erstes Auto mit serienmässig eingesetztem CAN-Bus und fünf Endgeräten, im Jahre 1991, hat sich die Anzahl der verbauten Endgeräte wie Sensoren, Kameras und Steuergeräte stetig erhöht. Die zunehmende Anzahl von in heutigen PKWs verbauten Fahrassistenz-Systemen, wie z.B. ESP, Einparkhilfen, Spurwechselassistenten, automatische Abstandswarner aber auch Infotainment- und Entertainment-Systemen haben zur Entwicklung weiterer Bus-Technologien wie LIN (siehe [12]), MOST (siehe [15]) und FLEX-Ray (siehe [3]) geführt. Diese neuen Bus-Systeme sollen designbedingte Beschränkungen von CAN wie z.B die beschränkte Bandbreite ausgleichen.

Damit Endgeräte, die an unterschiedlichen Bussen angeschlossen sind miteinander kommunizieren können sind Gateways, die diese Busse miteinander verbinden, notwendig. Die zunehmende Komplexität dieser heterogenen Fahrzeugnetze, das hohe Gewicht aber auch die Kosten werden für Autohersteller zunehmend zum Problem. Eine Lösung soll das heute im LAN-Bereich verbreitete IEEE 802.3 Standard Ethernet bieten (siehe [5]). Ethernet könnte eine Vielzahl der vorhandenen Bussen ersetzen. Bietet aber für die vielen im Fahrzeug vorhanden zeitkritischen Vorgänge wie z.B. das Auslösen eines Airbags, keine Echtzeitgarantien und ist somit erstmal unbrauchbar.

Eine Abhilfe schaffen Echtzeiterweiterungen des vorhanden Ethernetstandards. Das von der Firma TTTech entwickelte Time Triggered Ethernet Protocol (TTEthernet) (siehe [17]) ist eine solche Erweiterung. Bei TTE wird eine hochgenaue Zeitbasis innerhalb des Netzwerkes geschaffen. Das Senden und Empfangen geschiet mittels eines Schedulers, der über eine offline generierte Scheduling-Tabelle konfiguriert wird. Ethernet Frames die nicht in dieser Scheduling Tabelle erscheinen oder zu einer nicht korrekten Zeit versendet bzw. empfangen werden, werden von den vorhandenen TTE-Switches verworfen. TTE bestimmt ebenfalls unterschiedliche Nachrichtenklassen mit verschiedenen Prioritäten. So ist etwa ein priorisiertes Verschicken von Echtzeit-Netzwerkverkehr gegenüber Best-Effort Netzwerkverkehr möglich. Der Aufwand zum Generieren dieser Scheduling-Tabellen ist jedoch sehr hoch. Ein weiterer Nachteil von TTE ist, dass durch die Tatsache, das die Generierung der Scheduling-Tabellen offline geschiet ebenfalls ein in der Topologie statisches Netzwerk entsteht.

Ein Entfernen von Knoten ist zwar möglich, ein Hinzufügen aber nicht. Einen anderen Lösungsansatz bietet hier, das von der IEEE standardisierte und eher im Medienbereich angewendetet Audio/Video Bridging (siehe [10]), das im Verlauf dieser Arbeit AVB genannt werden soll. AVB ist eine Sammlung von Teilstandards die AVB beschreiben. AVB soll den Konfigurationsaufwand im Vergleich mit TTE entscheidend verringern, indem ein statisches Konfigurieren des Netzwerkes nicht mehr erforderlich ist. Ein Hinzufügen von Endgeräten kann sogar während der Laufzeit geschehen. In AVB vorhandene Mechanismen sorgen dafür, das die benötigte Bandbreite dynamisch reserviert wird.

1.1 Zielsetzung der Arbeit

Das Ziel dieser Arbeit, die im Rahmen der CoRE-Arbeitsgruppe der HAW Hamburg entsteht, ist es einen AVB-Stack aufbauend auf den von der IEEE bereitgestellten Standards

zu entwerfen, zu implementieren und dessen Funktionsweise zu evaluieren. Der Stack soll auf der bereitgestellten Hardware laufen und auf einer bestehenden Software-Basis aufbauen. Ggf. sind dazu Änderungen an der vorhandenen Software-Basis nötig, die hier ebenfalls besprochen werden sollen. Das genaue Umsetzten des in den IEEE-Standards beschriebenen Verhaltens ist sofern wichtig, da der enstandene AVB-Stack mit anderen dem Standard konformen Geräten funktionieren soll. Eine weitere aber nicht funktionale Anforderung ist ein sauber aufgebauter und gut dokumentierter Software-Stack, der auch von Dritten wartbar und erweiterbar ist.

1.2 Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende Abschnitte: In Kapitel 2 soll grundlegendes Wissen zu den Themen Echzeitanforderungen und Echtzeitverkehr vermittelt werden. Ebenfalls wird besprochen was AVB ist und aus welchen Teilstandards es besteht. Dieses Grundwissen ist zum weiteren Verständnis dieser Arbeit notwendig. In Kapitel 3 soll die Hardware und deren Eigenschaften für den Betrieb, des hier besprochenen AVB-Stacks vorgestellt werden. In Kapitel 4 werden die kongreten Anforderungen und das Konzept wie diese umgesetzt werden sollen gezeigt. In Kapitel 5 wird die kongrete Implementierung, des im vorhergehenden Kapitel vorgestellten Konzepts, gezeigt. In Kapitel 6 findet eine Evaluierung der Implementation statt. In Kapitel 7 wird ein Ausblick über den Inhalt dieser Arbeit hinaus gewährt und zukünftige Erweiterungsmöglichkeiten und Anwendungen besprochen.

2 Grundlagen

In diesem Kapitel werden die Grundlagen, die zum tiefgehenderen Verständnis dieser Arbeit nötig sind, hinreichend erklärt und mit Beispielen erläutert. Zu den Grundlagen zählt ebenfalls um was es sich bei Audio/Video-Bridging handelt, aus welchen Komponenten dieser Standard besteht und wie diese im Detail funktionieren.

2.1 Latenz und Jitter

Mit dem Begriff Latenz wird im allgemeinen die Verzögerung zwischen zwei aufeinanderfolgenden Events bezeichnet. In der Regel ist damit die Ursache und deren Wirkung gemeint. Zum Beispiel wird die Zeit, die zwischen dem Absenden und bis zum Eintreffen einer Nachricht als Latenz bezeichnet. Ein weiteres Beispiel währe ein Mausklick und eine Reaktion auf dem Bildschirm. Der Begriff Jitter beschreibt die zeitliche Variation dieser Zeit. Im Rahmen dieser Arbeit soll mit den Begriffen Latenz und Jitter die Ausführungszeit von Programmcode und deren zeitliche Variation bezeichnet werden. Eine grafische Erläuterung findet sich in Abbildung 2.1.

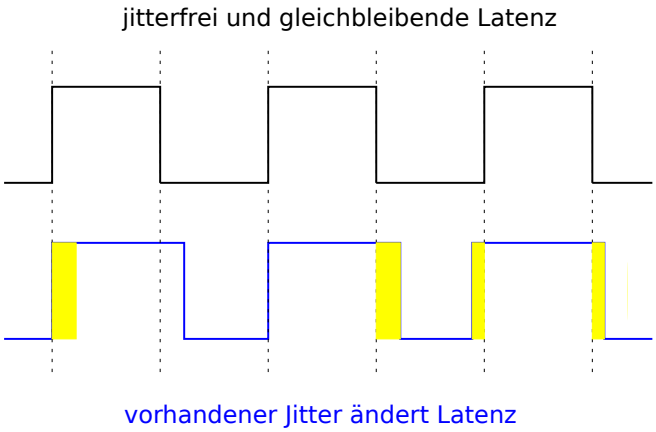

Abbildung 2.1: Darstellung von Jitter und dessen Einfluss auf die Latenz

2.2 Echzeitsysteme

In der Informatik werden Systeme als Echtzeitsysteme beschrieben, die eine vorgebene maximale Antwortzeit bis zum Vorhandensein eines Ergebnisses einhalten. Im allgemeinen spricht man bei dieser Antwortzeit von einer Reaktionszeit. Bei einem Überschreiten dieser Reaktionszeit wird das ermittelte Ergebnis grundsätzlich als ungültig betrachtet. Es gibt zwei Arten von Echtzeitsystemen, harte und weiche Echtzeitsysteme. Bei beiden Echtzeitsystemen wird bei einem Überschreiten der maximalen Reaktionszeit das Ergebnis einer Berechnung als ungültig betrachtet. Bei harten Echtzeitsystemen kann dieses Überschreiten der maximalen Reaktionszeit zu Sach- und/oder Personenschäden führen.Ein Airbag ist ein Beispiel für ein hartes Echtzeitsystem. Bei einem weichen Echtzeitsystem führt ein Überschreiten der maximalen Reaktionszeit ebenfalls zu einem Fehler, der jedoch in der Regel ohne schädliche Auswirkungen bleibt. Eine Rückfahrkamera stellt ein solches weiches Echtzeitsystem dar. Das Ausbleiben eins oder mehrerer Frames macht sich nur als kurzer Ruckler auf dem Bildschirm bemerkbar, sofern die Bildrate der Kamera hoch genug ist.

2.3 Echtzeit-Erweiterungen des Standard-Ethernets

Bei nach IEEE 802.3 standartisiertem Ethernet erfolgt der Versand der Netzwerk-Frames nach dem Best-Effort-Prinzip. Ein deterministisches Verhalten und das Beforzugen von zeitkritischen Traffic, auch im Falle von plötzlich auftretenden Bursts, ist so nicht möglich. Daher bietet sich keine Möglichkeit harte Echtzeit-Anforderungen, wie sie im Automotive-Bereich oder Produktionsumgebungen gefordert werden, einzuhalten. Eine Lösung bieten auf Standard-Ethernet aufbauende Echtzeit-Erweiterungen. Diese Echtzeit-Erweiterungen führen in der Regel neue Nachrichtenklassen ein, um eine höhere Priorität gegenüber Best-Effort-Traffic und ein vorhersagbares Netzwerk-Verhalten zu gewährleisten. Vorhandener Best-Effort-Traffic wird als niedrig priorisierte Nachrichtenklasse weitergeführt, um Kompatibilität mit vorhandenen Anwendungen zu waren.

2.4 Audio/Video-Bridging

Das in dieser Arbeit verwendete Audio/Video-Bridging wurde beim Institute of Electrical and Electronic Engineers, kurz IEEE, von der Audio Video Bridging Task Group im Rahmen des IEEE 802.1 Standatisierungs Komitees entwickelt. Es stellt eine Menge von Standards dar, um in IEEE 802.3 Full-Duplex Netzwerken zeitlich synchronisierten und priorisierten Traffic mit niedriger und garantierter Latenz zu ermöglichen.

2.4.1 Komponenten des AVB-Standards

Audio/Video-Bridging setzt sich aus folgenden Kern-Standards zusammen:

- **IEEE 802.1AS:** Timing and Synchronization for Time-Sensitive Applications (gPTP) (siehe [9])
  Wie eine gemeinsame Zeitbasis innerhalb eines AVB-Netzwerkes generiert wird, wird in diesem Standard definiert.
- **IEEE 802.1Qat:** Stream Reservation Protocol (SRP) (siehe [8])
  Der Standard definiert die Rollen von Talkern, als Anbieter von Streams und Listenern, als Abnehmer von Streams innerhalb eines AVB-Netzwerks. Ausserdem defininiert er, wie sich Informationen über einen Stream und dessen reservierte Bandbreite im AVB-Netzwerk verbreitet werden.

- **IEEE 802.1Qav** Forwarding and Queuing for Time-Sensitive Streams (FQTSS) (siehe [7])
  In diesem Standard wird definiert, wie ausgehende Frames verschiedenartig priorisiert werden und so ein garantiertes Echtzeitverhalten erzeugt wird.

- **IEEE 802.1BA:** Audio Video Bridging (AVB) Systems (siehe [10])
  Dieser Standard fasst die Grundlagen von Audio/Video-Bridging zusammen. Er beschreibt verschiedene Nachrichtenklassen, die Erkennung von AVB-fähigen Geräten, den Aufbau von AVB-Netzen, Anforderungen an diese Netze und deren Endgräte und die Framesauswahl beim Versenden von Frames.

Zum Verständnis, der an diese Arbeit gestellten Anforderungen aus Kapitel vier, ist es unerlässig, daß die vier hier genannten AVB-Standards näher im Detail zu betrachten. Die genaue Funktionsweise wird mit Beispielen dargestellt und erläutert.

2.4.2 IEEE 802.1AS: Timing and Synchronization for Time-Sensitive Applications (gPTP)

Um die an Audio/Video-Bridging gestellten Anforderungen, wie z.B. lippensynchrones synchronisieren zweier getrennter Audio- und Video-Streams oder das Wiedergeben eines auf mehrere Streams aufgeteilten Audiosignals eines Mehrkanal-Tons einhalten zu können, ist es notwendig, dass alle im Netzwerk über Audio/Video-Bridging kommunizierenden Knoten eine gemeinsame Zeitbasis besitzen. Das dafür verwendete IEEE 802.1AS generalized Precision Time Procotol bildet eine Teilmenge des IEEE 1588 Precision Time Protocol (siehe [6]). gPTP-kompatible Geräte kommunizieren innerhalb der in (siehe Abbildung 2.9) dargestellten gPTP-Domain, indem sie in 802.1AS spezifizierte Nachrichten miteinander austauschen. Innerhalb der gPTP-Domain gibt es unterschiedliche Rollen für IEEE 802.1AS kompatible Endgeräte (siehe Abbildung 2.2). Es existiert ein Grand Master und ein bis mehrere Slaves. Ein Knoten kann statisch als Grand Master konfiguriert werden oder er wird über ein in IEEE 802.1AS spezifiertes Auswahlverfahren zwischen den einzelnen Knoten ermittelt.

Der Grand Master enthält die Grand Master Clock auf die sich die in den Slaves enthaltenen Clocks synchronisieren. Ist ein Slave eine gPTP-kompatible Bridge an der mindestens ein weiterer Slave angeschlossen ist, so wird diese Bridge für dieses Slave wiederum zum Master. Auf diese Weise entsteht eine Clock-Hierarchie mit dem Grand Master an der Spitze.

Zum Synchronisieren der Clocks misst ein Master die Signallaufzeiten zwischen seinem Clock Master Port und dem am Clock Slave Port des angeschlossenen Slave. Danach sendet der Master eine Sync Message und eine darauffolgende Follow Up Message an seinen Slave. Auf diese Weise können Slaves Zeitabweichungen zwischen sich und dem Master feststellen und ihre Clocks dementsprechend korrigieren. Sollte ein Slave nicht antworten, so wird der angeschlossene Knoten als nicht 802.1AS capable markiert und der Clock Master Port des Masters zum Boundary Port an dem die gPTP-Domain endet.

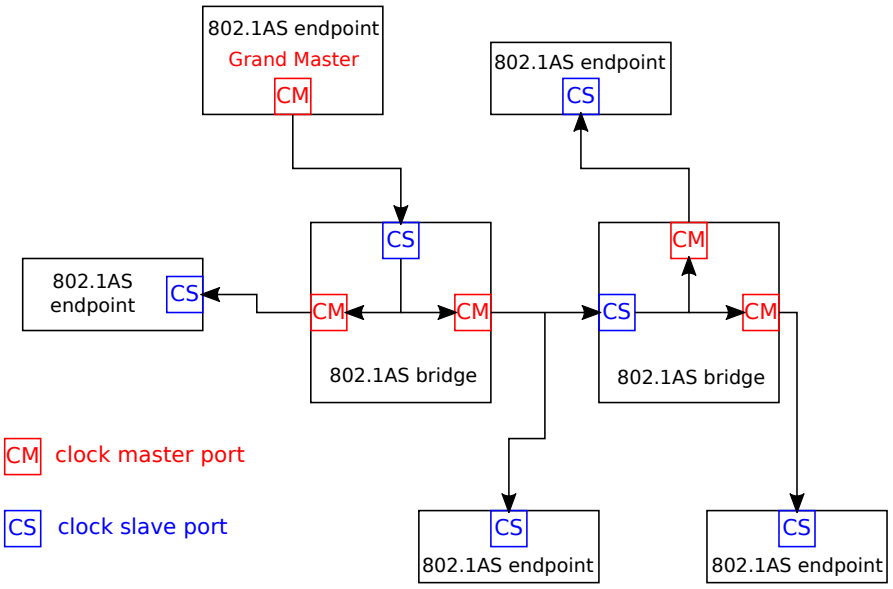

Abbildung 2.2: Master und Slaves in der gPTP-Domain (Darstellungsform aus [9])

2.4.3 IEEE 802.1Qat: Stream Reservation Protocol (SRP)

Das in IEEE 802.1Qat beschriebene Stream Reservation Protocol, kurz SRP, ist eine Erweiterung des IEEE 802.1Q in das es seit IEEE 802.1Q-2011 (siehe [11]) integriert ist. Die Aufgabe des SRP ist es, Informationen über vorhandene Endknoten die Streams bereitstellen (auch Talker genannt) und Endknoten, die diese Streams empfangen wollen (auch Listener gennant) über das Netzwerk zu propagieren.

Auf dem Weg zwischen Talkern und Listenern liegende SRP-kompatible A/V-Bridges

sammeln Informationen über von Streams verbrauchte Bandbreiten und dessen Verfügbarkeit und stellen so sicher, dass die für einen Stream garantierte Quality of Service (QoS) eingehalten wird.

SRP fügt dem Multiple Registration Protocol kurz MRP, das als IEEE 802.1Qak auch eine Erweiterung des IEEE 802.1Q ist einen weiteren Protokoll-Layer hinzu. MRP ist ein generisches Protokoll, dass es spezifischen Anwendungen erlaubt Attribute zu registrieren und diese über das Netzwerk zu propagieren. SRP benutzt das Multiple MAC Registration Protocol und das Multiple VLAN Registration Protocol und fügt das Multiple Stream Registration Protocol als weitere Anwendung hinzu. Die Abbildung 2.3 verdeutlich das Zusammenspiel von SRP, MMRP, MVRP, MSRP und MRP.

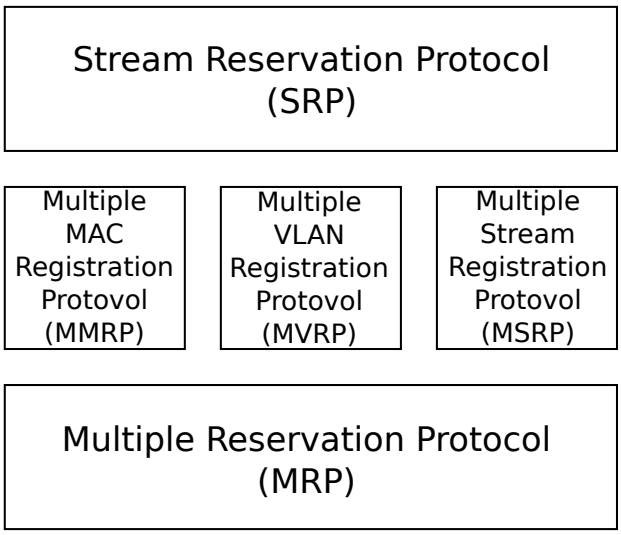

Abbildung 2.3: SRP-Stack (Darstellungsform nach [8])

Multiple Registration Protocol (MRP)

MRP ist so konzipiert, dass es zwischen einer Menge von Teilnehmern, Applicants genannt, operiert. Ein Applicant zeichnet sich durch seine konkrete MRP Application (z.B.

MMRP, MVRP oder MSRP) und dem an das Netz angeschlossenen Port aus. Ein MVRP-kompatibler Endknoten im Netz währe ein einzelner Applicant, während eine Bridge mit vier Ports vier Applicants besitzen würde.

Die Basisfunktionalität von MRP besteht darin, dass ein MRP Applicant ein Attribut deklarieren bzw. die Deklarierung eines Attributes wieder zurücknehmen kann. Die Folge daraus ist eine Registrierung dieses Attributes bzw. das Zurücknehmen der Registrierung dieses Attributes in dem Netzwerk-Peer dieses MRP Applicants.

Das Verhalten die Registrierung eines Attributs als Folge einer Deklaration dieses Attributs, erfolgt immer nur in einer Richtung. Vom deklarierenden Applicant zu seinem an seinem Port angeschlossenen registrierenden Peer. Ist dieser Peer ein Applicant, der Teil einer Bridge ist, so wird die Registrierung dieses Attributs innerhalb der Bridge anwendungsspezifisch bei den anderen Applicants in der Bridge ebenfalls deklariert. Ggf. kann dabei die Art des Attributs verändert werden. Das Propagieren von Attribut-Registrierungen innerhalb der Bridge erfolgt mittels des MRP Application Propagation Programs (MAP). Da es sich in dieser Arbeit um eine Stackimplementierung für einen Endknoten und keine Bridge handelt, soll auf die Funktionsweise des MAP nicht näher eingegangen werden.

Zum besseren Verständnis dieses Vorgangs dient die Abbildung 2.4.

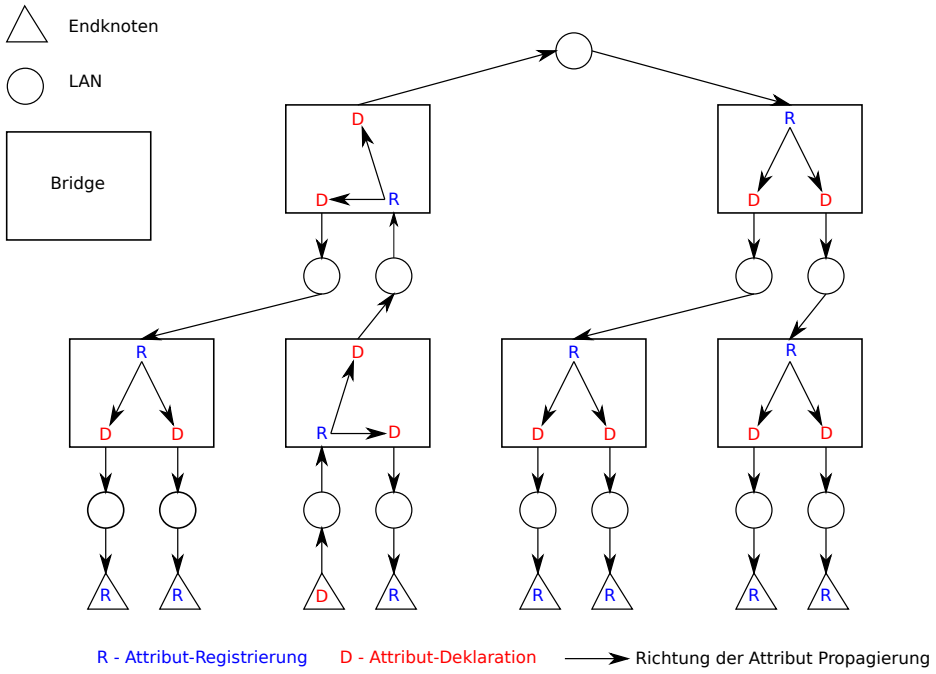

Abbildung 2.4: Propagierung eins MRP-Attributs (Darstellungsform nach [8])

Ein Applicant nimmt eine Attributdeklaration vor, die mit "D"gekennzeichnet ist. Der zu ihm gehörende Peer registriert das Attribut, gekennzeichnet mit "R". Dieser Applicant ist Teil einer Bridge. Dort sorgt die MAP-Komponente dafür, dass das Attribut von den beiden anderen Applicants in der Bridge neu deklariert wird. Auf diese Art entstehende Kettenreaktion sorgt dafür, dass die Deklaration des Attributs im Netzwerk propagiert wird und letztendlich alle Endknoten von dem Attribut "wissen".

Grundsätzlich sind Deklarationen von Attributen auf API-Ebene persistent, solange keine Deklarierung zurückgenommen wurde bleibt sie bestehen. Jedoch müssen Deklarationen von Attributen von MRP regelmässig aufgefrischt werden, da diese sonst veralten und automatisch entfernt werden. Verliert ein Applicant seine Verbindung zum Netzwerk werden auch die Deklarationen seiner Attribute nach gegebener Zeit entfernt.

Multiple MAC Registration Protocol (MMRP)

SRP benutzt MMRP optional. MMRP wird benutzt um Deklarationen von Talker-Attributen im Netzwerk zu begrenzen. Dies wird Talker Pruning genannt und per Default

deaktiviert. Ist ein Applicant an Talker-Deklarationen interessiert, so muss dieser vorher eine vorgegebene MAC-Adresse als Attribut mittels MMRP deklarieren.

Multiple VLAN Registration Protocol (MVRP)

Im Kontext von SRP wird MVRP dazu benutzt, vom VLAN-Attribut oder genauer die Mitgliedschaft zu einem VLAN zu deklarieren. Talker und Listener müssen Teil des gleichen VLANs sein, damit ein Stream vom Talker zum Listener gesendet werden kann.

Multiple Stream Registration Protocol (MSRP)

MSRP dient dazu Informationen über Talker und Listener über das Netzwerk zu propagieren. Dazu stellt es das Talker- und Listener-Attribut bereit. Ein weiteres Attribut, das Domain-Attribut exisitiert, um die in Unterabschnitt 2.4.5 beschriebene und in IEEE 802.1BA spezifierte SRP-Domain zu erzeugen.

Talker-Attribute werden von Talkern und Bridges deklariert und enthalten Information über Streams, die von Talkern angeboten werden. In ihnen sind die IDs der Streams enthalten, um diese untereinander unterscheiden zu können. Ausserdem enthalten sind Information zur Bandbreite, die ein Stream im Falle eines Empfangs durch einen Listener verbrauchen würde. Die Bandbreite wird in maximaler Framegrösse, eine maximale Anzahl von Frames, für ein in IEEE 802.1Qav festgelegtes Class Measurement Interval angegeben. Talker-Attribute existieren in zwei verschiedenen Formen, dem Talker Advertise und dem Talker Failed. Das Talker Advertise-Attribut ist das reguläre Attribut, das von einem Talker deklariert wird. Ein Talker Failed ensteht in einer Bridge und wird von der MAP-Komponente aus einem Talker Advertise erzeugt. Dies ist z.B. der Fall, wenn die von einem Stream benötigt Bandbreite nicht vorhanden ist oder ein Port eines anderen MRP-Teilnehmers einer Bridge ein Boundary Port ist.

Listener-Attribute werden von Listenern und Bridges deklariert und enthalten Information über Streams, die Listener empfangen wollen und enthalten die Stream ID. Von dem Listener existieren insgesamt drei verschiedene Formen. Diese sind: Listener Ready, Listener Ready Failed und Listener Asking Failed. Listener Ready und Listener Asking Failed sind die von einem Listener regulär deklarierten Attribute. Listener Ready als Antwort auf ein Talker Advertise und Listener Asking Failed als Antwort auf ein Talker Failed. Listener Asking Failed bedeutet, wenn ein Listener nicht benötigte Bandbreite besitzt

um einen Stream zu empfangen. Listener Ready Failed wird von der MAP-Komponente innerhalb einer Bridge erzeugt und als solches propagiert. Listener Ready Failed ist ein Zusammenschluss von Listener Ready und Listener Asking Failed. Dieser ensteht, wenn in einer Bridge mehr als ein Listener einen Stream empfangen möchte und sowohl ein Listener Ready als auch ein Listener Asking Failed propagiert werden müsste.

Domain-Attribute enthalten die numerische Darstellung der AVB-Nachrichtenklasse, der Priorität und der VLAN ID, die in einer SRP-Domäne benutzt werden soll. Das Attribut wird nicht propagiert und dient dazu, die Aussengrenzen der SRP-Domain festzulegen. Es wird von Endknoten an die Bridge gesendet. Im Falle des Ausbleibens der Deklarierung eines Domain Attributs oder unterschiedlicher Werte zum Rest, der in der SRP-Domain benutzten, wird der Port des entsprechenden Teilnehmers einer Bridge zum Boundary Port.

Zum besseren Verständnis von Talker- und Listener-Attributen sollen Abbildung 2.5 und Abbildung 2.6 beitragen.

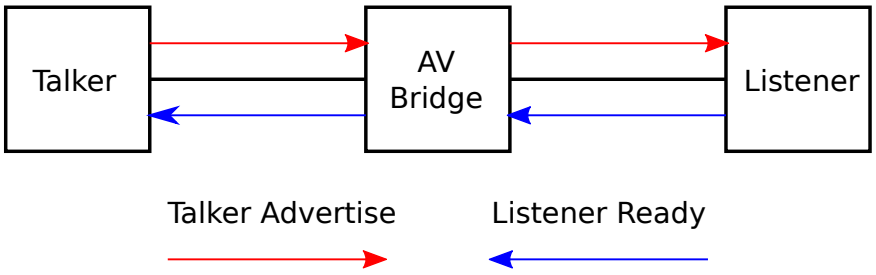

Abbildung 2.5: Eine erfolgreiche Verbindungsaufnahme zwischen Talker und Listener

In Abbildung 2.5 deklariert ein Talker ein Talker Advertise-Attribut, mit dem er mit-

teilen möchte, dass er einen Stream anzubieten hat. Da genügend Bandbreite auf der Strecke zwischen Talker und Listener bereitsteht und diese sich ebenfalls in der selben SRP-Domain befinden, erreicht das Talker Advertise-Attribut den Listener. Der Listener sendet daraufhin ein Listener Ready-Attribut an den Talker zurück, woraufhin dieser mit dem Streamen beginnen kann.

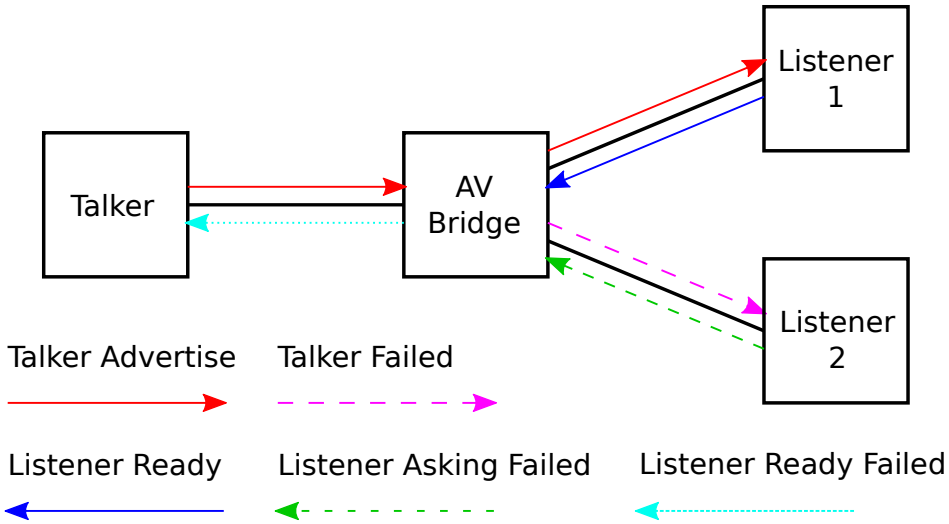

Abbildung 2.6: Eine teilweise erfolgreiche Verbindungsaufnahme zwischem einem Talker und zwei Listenern

In dem Beispiel von Abbildung 2.6 deklariert ein Talker ebenfalls ein Talker Advertise-Attribut. Jedoch erreicht es nur Listener 1 erfolgreich. Für Listener 2 wird das Talker Advertise-Attribut in der Bridge zu einem Talker Failed-Attribut verändert. Ein Grund kann sein, dass auf der Strecke zwischen Talker und Listener 2 nicht genügend Bandbreite zu Verfügung steht. Von Listener 2 wird deshalb ein Listener Asking Failed-Attribut als Antwort deklariert. Listener 1 schickt ein Listener Ready-Attribut. Innerhalb der Bridge werden beide Attribute zusammengefasst und es wird ein Listener Ready Failed-Attribut deklariert. Für den Talker ist es nicht relevant, ob eine oder mehrere Listener einen Stream empfangen möchten, er sendet den Stream nur einmal und dieser wird in der Bridge an verschiedene Ports weitergeleitet. Jedoch enthält das Listener Ready Failed-

Attribut Informationen, die auf einer höheren Verarbeitungsschicht ausgewertet werden können.

2.4.4 IEEE 802.1Qav: Forwarding and Queuing for Time-Sensitive Streams (FQTSS)

Während das in Unterabschnitt 2.4.3 behandelte SRP die Aufgabe hat Bandbreite auf dem Pfad von einem Talker zu einem oder mehreren Listenern zu reservieren, ist die Aufgabe dieses Standards, das diese Bandbreite bei dem Verschicken von Frames nicht überschritten wird. Dies wird erreicht, indem mit dem Credit Based Shaper kurz CBS ein Fair-Queuing Modell umgesetzt wird. Jeder ausgehenden, und mit dem CBS-Algorithmus arbeitenden, Queue wird ein Credit zugewiesen nachdem ausgewählt wird, ob die Queue bereit ist einen Frame zu senden oder nicht.

Den in Unterabschnitt 2.4.5 standartisierten AVB SR Class A und B Prioritäten wird jeweils ein vordefiniertes Class Measurement Interval zugewiesen. Dies ist ein Zeitabschnitt, für den bestimmt wird wieviele Frames mit welcher Maximalgrösse innerhalb eines solchen Abschnitts gesendet werden sollen. Für AVB SR Class A Traffic liegt dieser bei 125ms, und für AVB SR Class B bei 250ms.

Um nun die von einem Stream verbrauchte Bandbreite auszurechnen, ist es nötig, die mit jedem Frame anfallenden Overhead wie z.B. Ziel- und Quelladresse, Ethertype, CRC Felder mit in den Rechnung einzubeziehen.

Die für einen Stream konsumierte Bandbreite berechnet sich daher wie folgt:

$$assumedPayloadSize = MaxFrameSize \tag{2.1}$$

$$maxFrameRate = MaxIntervalFrames \times (1/classMeasurementInterval) \tag{2.2}$$

$$actualBandwidth = (perFrameOverhead + assumedPayloadSize) \times maxFrameRate \tag{2.3}$$

Der Credit einer Queue beschreibt, ob es einer Queue erlaubt ist einen Frame zu senden oder nicht. Ist der Wert des Credits >= 0 so darf die Queue einen Frame senden. Ist der Wert < 0 ist dies nicht der Fall. Die zwei den Credit einer Queue veränderten Parameter werden Idle Slope und Send Slope genannt. Idle Slope ist die Rate in Bits/Sec um die sich der Credit erhöht, wenn eine Queue sich im Idle Zustand befindet. Es ihr also nicht erlaubt ist einen vorrätigen Frame zu senden oder ein Sendevorgang abgeschlossen ist und abgewartet werden muss bis der Credit wieder 0 ist. Send Slope ist die Rate in Bits/Sec in der sich der Credit verringert während eine Queue einen Frame versendet. Sollte eine Queue keinen Frame zum Senden vorrätig haben und der Credit positiv sein, so wird der Creditwert auf 0 zurückgesetzt.

Idle Slope und Send Slope berechnen sich wie folgt:

$$idleSlope = portTransmitRate \tag{2.4}$$

$$sendSlope = idleSlope - portTransmitRate \tag{2.5}$$

portTransmitRate ergibt sich aus der an einem Port maximal verfügbaren Bandbreite, die für AVB-Traffic benutzt werden darf. Laut Standard sind dies 75%. Bei einer maximalen Bandbreite von 100MBit währen das also 75Mbit.

Die Abbildung 2.7 soll den Ablauf des CBS-Algorithmuses genauer verdeutlichen.

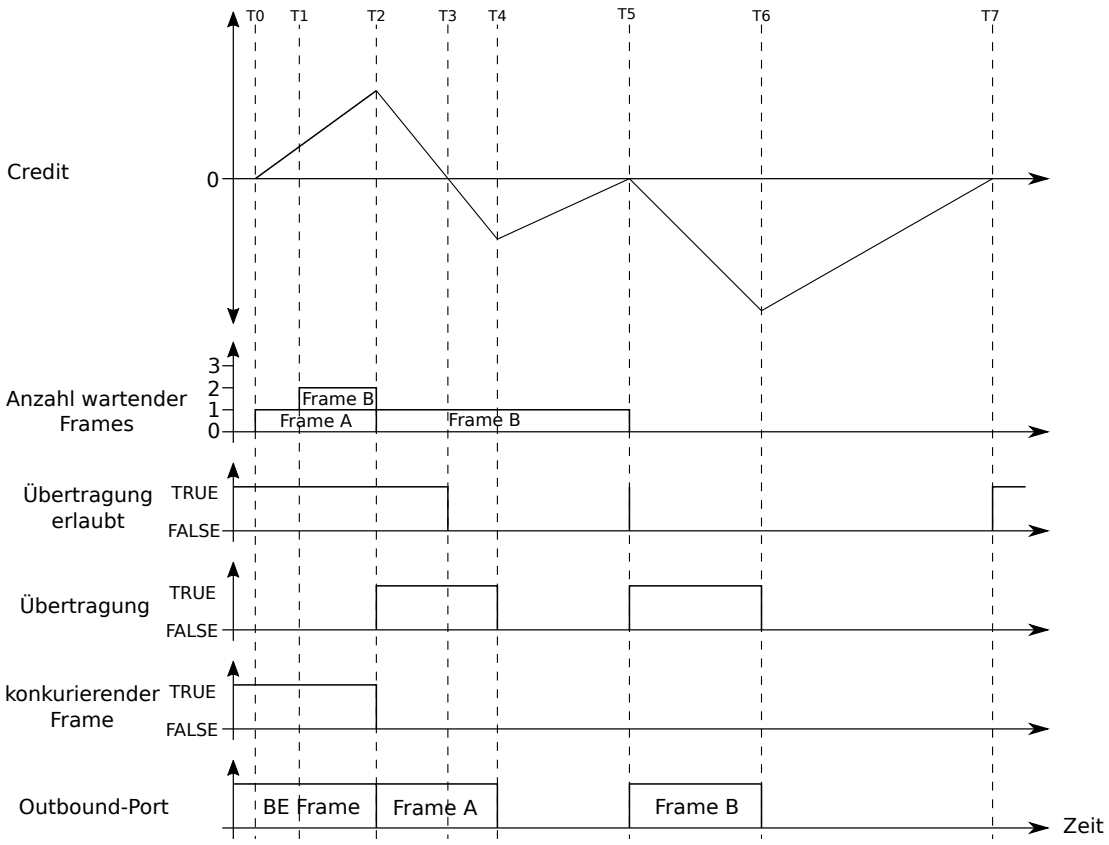

Abbildung 2.7: Der CBS-Algorithmus (Darstellungsform nach [7])

Die Grafik stellt Verlauf des Credits für eine höher als Best Effort kategorisierte Queue dar. Ob diese Queue AVB SR Class A oder B entspricht ist für dieses Beispiel unerheblich. Die mit Buchstaben nummerierten Frame-Bezeichner ensprechen nicht den AVB-Prioritätsklassen.

T0 Ein neuer Frame, mit der Bezeichnung Frame A, wird der Queue hinzugefügt. Da aber in diesem Moment schon ein Frame der Kategorie Best Effort gesendet wird und mangels Frame Preemption das Versenden nicht unterbrochen werden kann, muss die Queue warten und ihr Credit beginnt zu steigen.

T1 Ein weiterer neuer Frame mit der Bezeichnung Frame B wird der Queue hinzugefügt. Der Best Effort Frame wird immer noch gesendet, die Queue wartet und der Credit steigt stetig.

T2 Das Versenden des Best Effort Frames ist abgeschlossen und der Credit der Queue ist >= 0. Die Übertragung ist ihr wegen des positiven Credits erlaubt und die Übertragung beginnt. Frame A wird auf die Leitung gelegt.

T3 Das Senden von Frame A dauert an und Send Slope hat den Credit soweit verringert, dass dieser jetzt negativ ist.

T4 Das Versenden von Frame A ist abgeschlossen und Frame B wartet darauf versendet zu werden. Da der Credit aber negativ ist, ist dies der Queue nicht erlaubt. Die Queue ist idle und der Credit fängt an sich mittels Idle Slope zu erhöhen.

T5 Der Credit errreicht 0, die Übertragung ist der Queue wieder erlaubt und diese beginnt sofort mit der Übertragung von Frame B. Der Credit beginnt ebenfalls sich wieder zu verringern.

T6 Der Versand von Frame B ist abgeschlossen und der Credit ist negativ. Die Queue ist wieder idle. Es muss gewartet werden, auch wenn kein weiterer Frame vorliegt bis der Credit der Queue wieder 0 erreicht hat.

T7 Der Credit erreicht 0.

2.4.5 IEEE 802.1BA: Audio Video Bridging (AVB) Systems

Dieser Standard beschreibt den grundlegenden Aufbau von AVB-Netzen und deren Mindestanforderungen an die Teilnehmer.

Ein typisches AVB-Netzwerk besteht i.d.R. aus folgenden Komponenten (siehe Abbildung 2.8):

a) AVB-kompatible Endgeräte die Talker sind und AVB-Streams anbieten und versenden.

b) AVB-kompatible Endgeräte die Listener sind und AVB-Streams empfangen.

c) AVB-kompatible Endgeräte die sowohl Talker als auch Listener sind.

d) AVB-kompatible MAC-Bridges

e) Nicht-AVB-kompatible Endgeräte

f) Nicht-AVB-kompatible MAC-Bridges

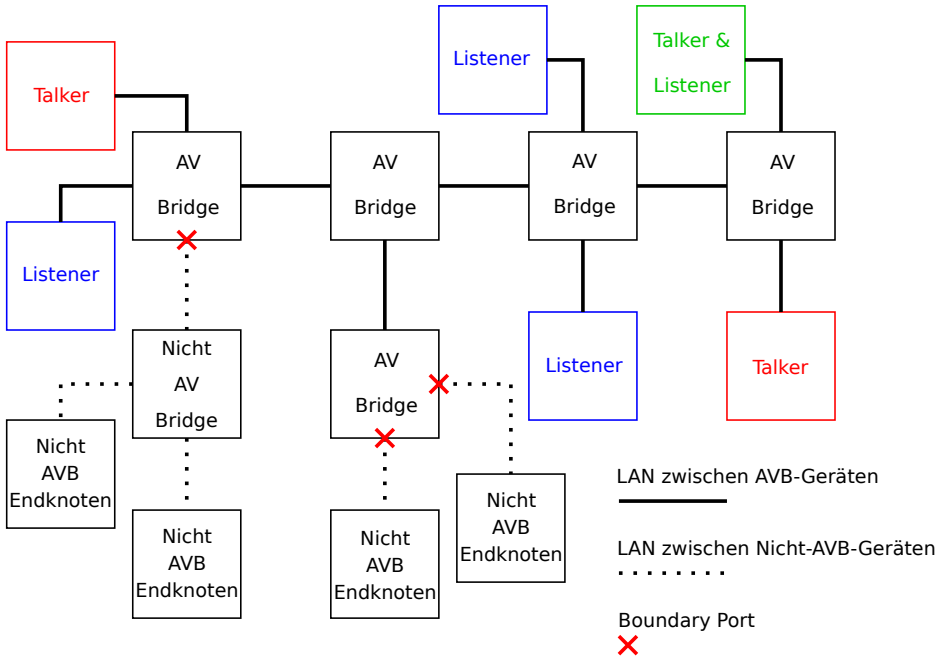

Abbildung 2.8: Beispiel eines AVB-Netzwerkes (Darstellungsform nach [10])

AVB-Streams können nur zwischen AVB-kompatiblen Endgeräten und Bridges gesendet und empfangen werden. Ist ein AVB-kompatibles Endgerät oder eine Bridge mit einem Nicht-AVB-kompatiblen Endgerät oder einer Bridge verbunden, so wird der Ausgangsport des AVB-kompatiblen Geräts oder der Bridge als Boundary Port bezeichnet. Er bildet die Aussengrenze des als AVB-Domain bezeichneten Zusammenschlusses der AVB-kompatiblen Geräte.

Die Mindestanforderungen an AVB-Kompatible Entgeräte sind:

a) 100MBit Full-Duplex Anbindung

b) Talker und Listener müssen mindestens einen Stream anbieten bzw. empfangen können.

c) Talker und Listener müssen in der Lage sein Talker- bzw. Listener-Attribute im Netzwerk zu deklarieren.

d) Talker und LIstener müssen gPTP-kompatibel sein, damit eine Bridge diese als „As Capable“ markieren kann.

e) Listener müssen in der Lage sein VLAN Membership-Attribute im Netzwerk zu deklarieren.

AVB beschreibt zwei Prioritätsklassen für die Kommunikation mittels AVB-Streams im Netzwerk. Diese beiden Klassen werden als Stream Reservation Class A und Stream Reservation Class B bezeichnet, wobei die Stream Class A die hochrangigere Klasse der beiden ist. Da AVB auf 802.1Q aufsetzt, also Ethernet-Header mit VLAN Header existieren, können die beiden AVB-Prioritätsklassen auf zwei der insgesamt acht 802.1Q Prioritäten gemappt werden und der Rest als Best Effort genutzt werden.

SR Class	numerische Darstellung	CMI	max. Ende-zu-Ende Latenz
A	6	125ms	2ms
B	5	250ms	50ms

Tabelle 2.1: Die AVB-Prioritätsklassen

Die Ende-zu-Ende Latenz wird nur über eine maximale Netzwerkstrecke von 7 Knoten garantiert.

Erkennung der AVB-Domain

Die Kommunikation über AVB-Streams ist nur innerhalb einer AVB-Domain möglich, da ausserhalb dieser Domain keine Echtzeitgarantien mehr gemacht werden können. Eine AVB-Domain ist die Schnittmenge der in IEEE 802.1AS standardisierten gPTP-Domain und der in IEEE 802.1Qat standartisierten SRP-Domain.

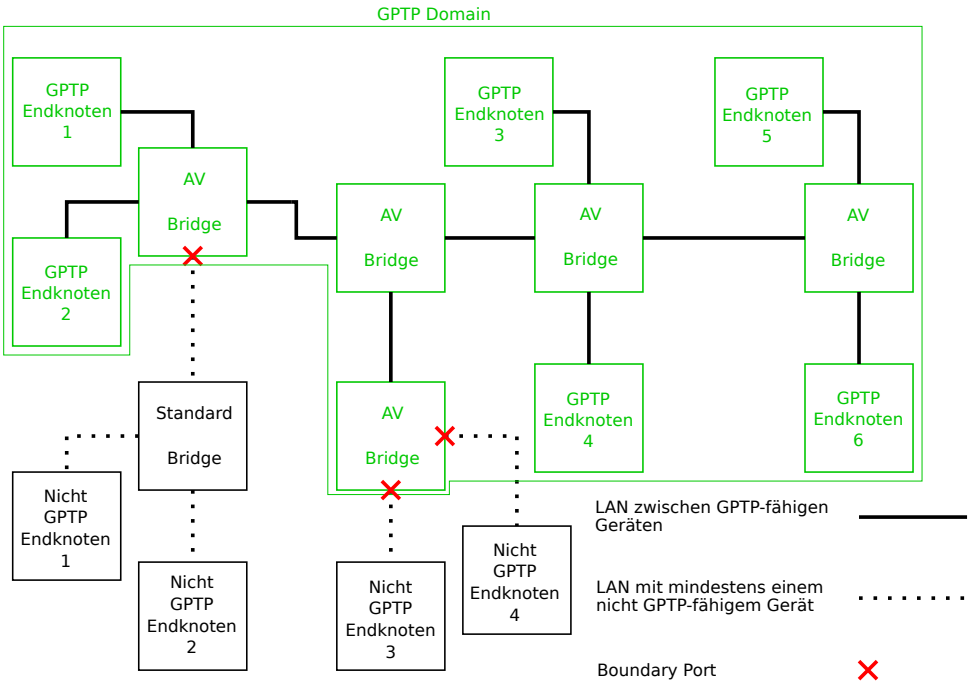

Abbildung 2.9: gPTP-Domain (Darstellungsform nach [10])

Abbildung 2.9 zeigt das aus Abbildung 2.8 bekannte AVB-Beispielnetzwerk. Die grün eingefärbten Geräte sind alle Teil einer gPTP-Domain. Die gPTP -Endknoten wurden von den Bridges als gPTP-fähig erkannt und entsprechend markiert. Die A/V-Bridges untereinander haben sich ebenfalls als gPTP-fähig erkannt, da dies eine Minimumvoraussetzung für AVB ist. Die Standardbridge ist nicht-gPTP-fähig und der ausgehende Port der angeschlossenen A/V-Bridge dementsprechend als Boundary Port markiert. Ebenso verhält es sich mit den A/V-Bridge Ports an denen die Nicht-gPTP-Knoten 3 und 4 angeschlossen sind.

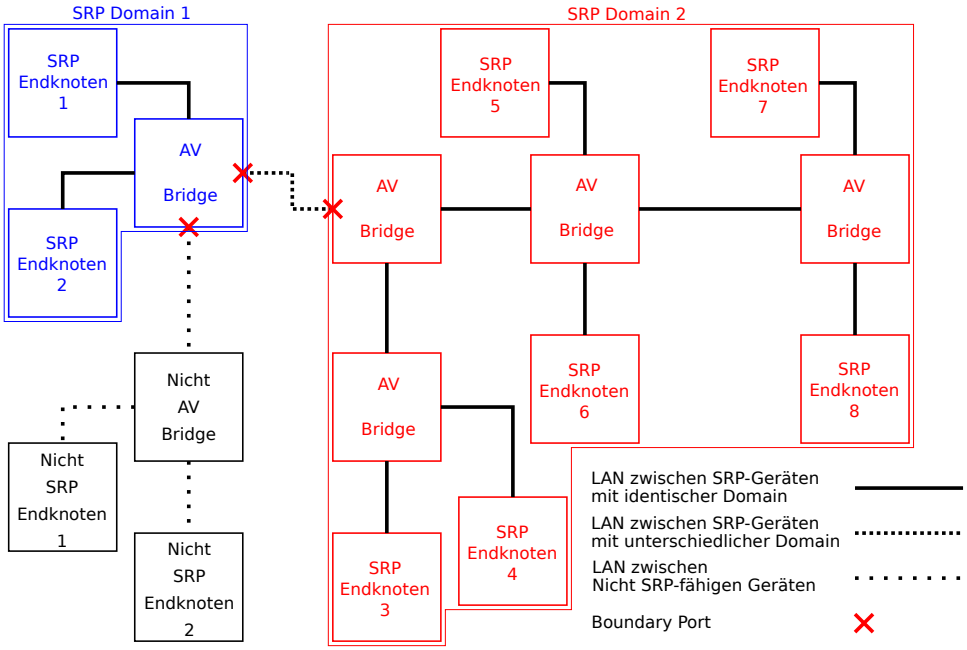

Abbildung 2.10: SRP-Domain (Darstellungsform nach [10])

In Abbildung 2.10 wird ebenfalls Bezug auf das AVB-Beispielnetzwerk genommen. In diesem falle werden unterschiedliche SRP-Domains betrachtet. Wie zu sehen ist sind zwei unterschiedliche SRP-Domains entstanden, hier blau und rot dargestellt. Eine Bridge und die zwei angeschlossenen Endgeräte sind nicht Teil einer SRP-Domain, da diese nicht-SRP-kompatibel sind. Der Grund dafür, dass zwei SRP-Domains existieren besteht darin, das zwar die Geräte in der SRP-Domain 1 als auch in der SRP-Domain 2 passende Domain-Attribute untereinander deklariert und registriert haben, aber die numerische Darstellung dieser Attribute für SR Class A und SR Class B, in beiden Domains unterschiedlich ist. Beispielsweise könnten in SRP-Domain 1 die Werte für SR Class A und SR Class B 5 und 6 sein, in SRP-Domain 2 aber 6 und 5. Daher ist es nicht möglich ein Talker Advertise-Attribut von SRP-Domain 1 in SRP-Domain 2, oder anders herum zu schicken. Ein Talker Advertise-Attribut würde in beiden Fällen in der Bridge zu einem Talker Failed-Attribut gewandelt, da es sich hier über eine Kommunikation über einen Boundary Port hinaus handelt.

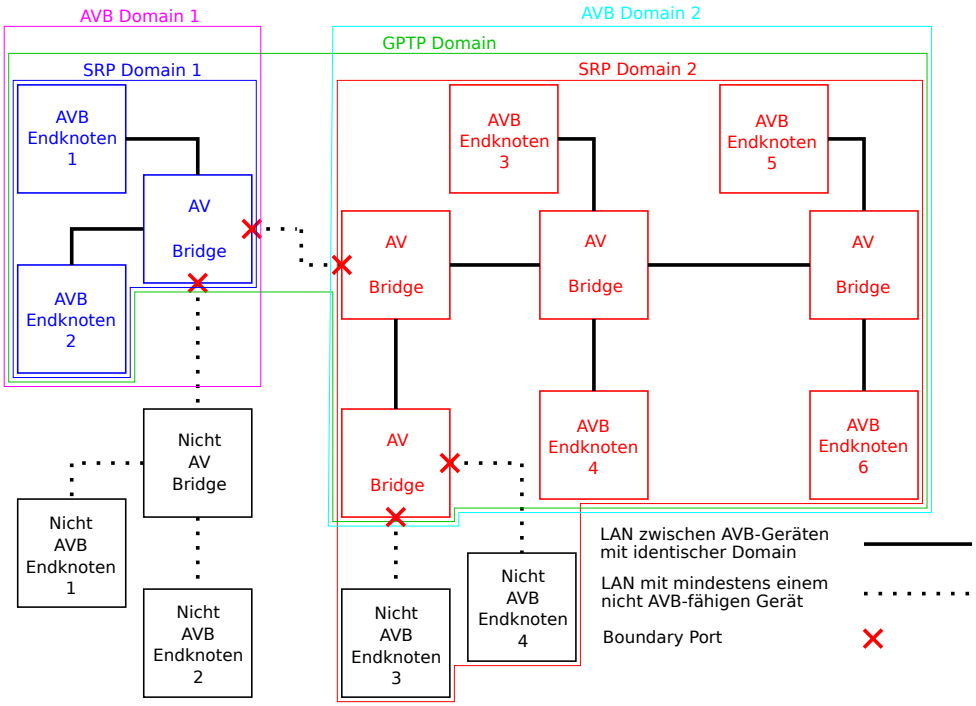

Abbildung 2.11: AVB-Domain (Darstellungsform nach [10])

Abbildung 2.11 zeigt die Schnittmenge der gPTP- und SRP-Domains. Aufgrund der Tatsache, das zwei SRP-Domains entstanden sind, haben sich ebenfalls zwei AVB-Domains gebildet, hier violet und türkis. Zwischen beiden AVB-Domains kann keine AVB-Kommunikation stattfinden, da dies über Boundary Ports hinaus erfolgen würde. Von AVB-Kommunikation ausgeschlossen sind ebenfalls Nicht-AVB-Endknoten 1-4.

3 Hardware

Dieses Kapitel soll einen kurzen Überblick über die Hardware geben, die zur Entwicklung des AVB-Stacks eingesetzt wurde.

3.1 Hilscher NXHX 500-ETM

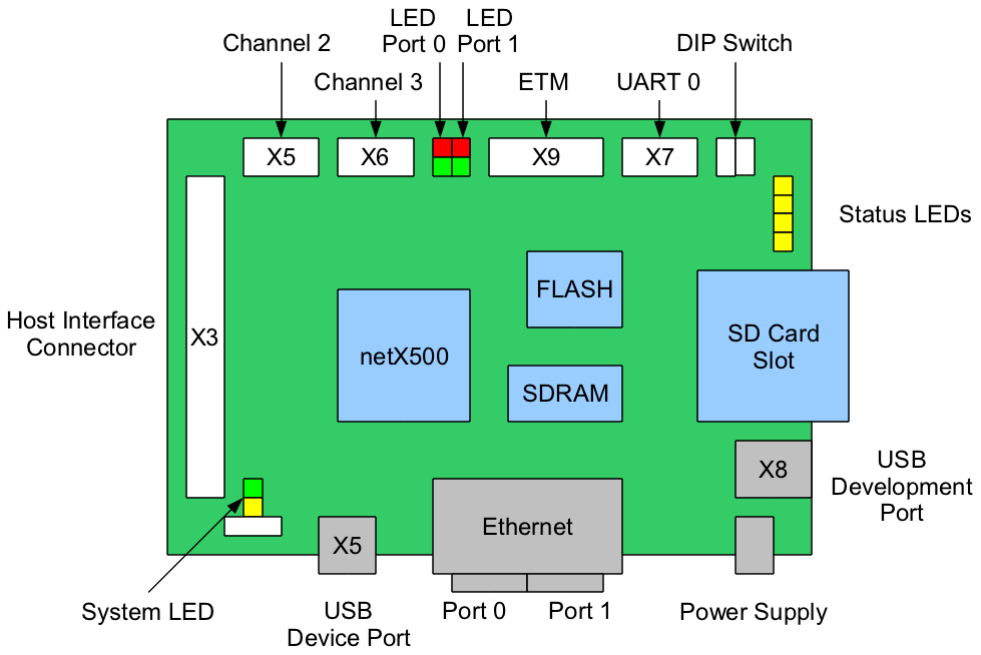

Abbildung 3.1: schematische Ansicht des Hilscher NXHX 500-ETM Boards (Quelle: [16])

Das NetX 500 ETM Board (siehe [13]) der Firma Hilscher basiert auf einer NetX SoC Architektur. Diese ermöglicht die voneinander unabhängige Kommunikation über vier verschiedene Kommunikationskanäle. Die Architektur ist auf maximalen Datendurchsatz bei der Kommunikation optimiert.

In dem SoC befindet sich eine 32Bit, mit 200Mhz getaktete, ARM 926EJ-S CPU. Diese verfügt über 144KByte RAM und 32KByte ROM. Zusätzlich stehen auf dem Board 8MByte SDRAM und 16MByte Flashspeicher zur Verfügung.

Das NetX 500 ETM Board besitzt zwei Ethernet-Schnittstellen. Weitere Schnittstellen wie SPI, UART, können über aufsteckbare Module nachgerüstet werden. Zusätzlich befinden sich diverse Debug-Schnittstellen, eine USB-Schnittstelle und eine SD-Kartenslot auf dem Board.

Im Laufe dieser Arbeit soll das Hilscher NetX 500 ETM Board nur NetX Board genannt werden.

3.2 Extreme Networks Summit X440-8t

Abbildung 3.2: Extreme Networks Summit X440-8t (Quelle: [2])

Abbildung 3.2 zeigt den Summit X440-8t AVB-Switch der Firma Extreme Networks (siehe [2]). Dieser wird im Laufe dieser Arbeit wegen besserer Lesbarkeit AVB-Switch genannt.

Der Switch beinhaltet eine 500Mhz CPU mit 512MB RAM und 512MB Flash. Auf dem Switch läuft das flexible erweiterbare ExtremeXOS. Er verfügt über acht GBit Ethernet Ports, einen Port für eine serielle Konsole und ein 100Mbit Management Port. Der Switch verfügt über vielfälltige Konfigurationsmöglichkeiten, insbesondere QoS und ist in der Lage AVB Traffic abzuwickeln.

3.3 XMOS AVB Audio Endpoint Kit (XK-AVB-LC-SYS)

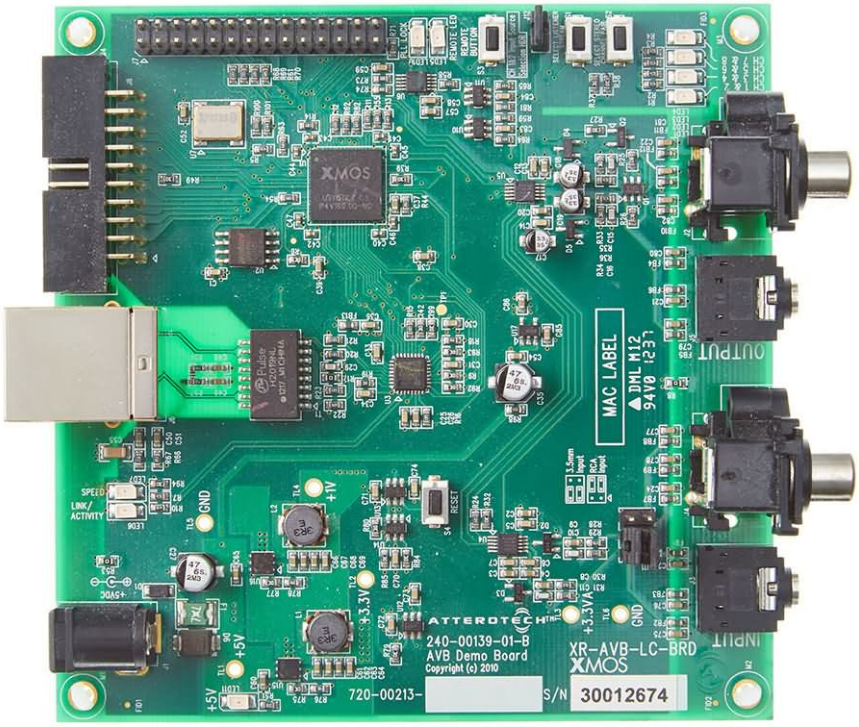

Abbildung 3.3: XMOS AVB Audio Endpoint Kit(Quelle: [14])

Abbildung 3.3 zeigt eines der beiden Boards des Audio Endpoint Kits der Firma XMOS (siehe [14]). Diese bilden die vorhandene AVB-Referenzplattform und verfügen über Cinch und Klinkenbuchsen zur Ein- und Ausgabe von Ton, der über IEEE 802.1 AVB and IEEE 1722 verschickt wird.

Mittels Wireshark kann der Netzwerkverkehr und die Kommunikation mit dem AVB-Switch beobachtet werden und so Fehler in dem sich in Entwicklung befindlichen AVB-Stack leichter gefunden werden.

4 Anforderungen und Konzept

In diesem Kapitel sollen die von Audio/Video-Bridging gestellten Anforderungen an den umzusetzenden Software-Stack analysiert werden. Die in Kapitel 2 vermittelten Grundkenntnisse über die Funktionsweise von Audio/Video-Bridging werden hier weiter vertieft, um konkrete Anforderungen formulieren zu können. Im Anschluss wird ein Konzept zum Aufbau des Stacks und dessen Umsetzung vorgestellt.

4.1 Ausgangssituation

Der von Kai Müller entwickelte echtzeitfähige TTEthernet-Stack (siehe [16]) soll die Ausgangbasis für den hier zu entwickelnden Audio/Video-Bridging-Stack bilden. Dies ist dahingehend begründed, dass der TTE-Stack zusammen mit dem NetX-Board eine ausreichende getestete Hardware- und Software-Basis bildet, als auch den neuesten Entwicklungsstand bezüglich des NetX-Board Hardware Abstraction Layers und weitere Sub-Komponenten bietet.

Der TTE-Stack ist Intterupt-getrieben, d.h. der Programmablauf besteht ausschliesslich aus Interrupt Service Routinen, die von Events ausgelöst werden. Diese Events können Einträge im TTE-Scheduler sein und werden somit zu festen Zeitpunkten ausgelöst oder sind an eingehende Frames gekoppelt.

Der TTE-Stack ist abhängig von der hochgenauen zeitlichen Auflösung des TTE-Schedulers. Wird diese und somit auch dessen Schedule nicht mehr eingehalten ist die korrekte Funktionsweise des TTE-Stacks nicht mehr gegeben.

Die störungsfreie, architektonische Integration des zu erstellenden AVB-Stacks in die vorhandene Software-Basis stellt daher ein Problem dar. Eine Lösung wird im Konzeptteil dieses Kapitels präsentiert.

Ein weiteres Problem bildet die zwischen TTEthernet und Audio/Video-Bridging vorhandene unterschiedliche Sicht auf die im Netzwerk vorhandenen Prioritäten für Frames. TTE kennt Time Triggered Traffic der die höchste Priorität darstellt, Rate Constrained Traffic als zweithöchste Priorität und Best Effort Traffic als niedrigste Priorität. Bei Audio/Video-Bridging sind dies die in Unterabschnitt 2.4.5 beschriebenen AVB SR Class A und B und ebenfalls Best Effort. Time Triggered Traffic und AVB SR Class A Traffic stehen im Konflikt zueinander. Da diese für des jeweilige Protokoll höchste im Netzwerk vorhandene Priorität darstellen sollen. Eine Lösung dazu soll ebenfalls im Konzeptteil geklärt werden.

4.2 Anforderungen

Da Audio/Video-Bridging eine durch das IEEE standartisierte Sammlung von Protokollen dargstellt, ist ein vorrangiges Ziel diese Standards möglichst genau einzuhalten und somit auch zu anderen AVB-kompatiblen Geräten kompatibel zu sein, um mit diesen kommunizieren zu können. Der in dieser Arbeit für den Aufbau eines Beispiel-AVB-Netzwerks verwendete AVB-Switch (siehe Abschnitt 3.2) ist ein solches Gerät.

Ein weiteres Ziel ist nicht nur die Einhaltung des spezifizierten Verhaltens sondern auch den in den IEEE Standards beschriebenen internen Aufbau der verschiedenen Protokolle einzuhalten. Die dargestellten Strukturen sollen sich auch in dem fertig gestellten AVB-Stack wiederfinden und die IEEE-Standards damit als eine Art Wartungshandbuch benutzt werden können. Dritte sollen in die Lage versetzt werden den Stack zu erweitern oder evtl. auftretende Fehler einfacher beseitigen zu können.

Wie in Unterabschnitt 2.4.5 beschrieben ist die Kommunikation mittels AVB nur in einer AVB-Domain möglich. Eine Grundvorraussetzung ist daher die Möglichkeit eine AVB-Domain aufzubauen. Wie ebenfalls beschrieben wurde ist eine solche AVB-Domain die Schnittmenge zwischen einer gPTP-Domain (siehe Unterabschnitt 2.4.2) und einer SRP-Domain (siehe Unterabschnitt 2.4.3). Unterabschnitt 4.2.1 wird sich daher näher mit den Anforderungen an gPTP zum Aufbau dieser gPTP Domain befassen. Entsprechendes gilt für Unterabschnitt 4.2.2 für SRP zum Aufbau einer SRP-Domain.

Eine weitere Anforderung an SRP besteht darin, Endgeräte als Talker und Listener oder ggf. beides innerhalb einer AVB-Domain bekanntzumachen. Unterabschnitt 4.2.2 befasst sich ebenfalls mit diesem Thema.

Die Anforderungen an den Credit Based Shaper aus Unterabschnitt 2.4.4 werden in Unterabschnitt 4.2.3 genauer beschrieben.

4.2.1 Anforderungen an das gPTP

Das generalized Precision Time Protocol dient dazu, die Zeit mindestens zweier Endgeräte innerhalb einer gPTP-Domain zu synchronisieren. Der Aufbau dieser gPTP-Domain und die damit notwendige Erkennung, der als 802.1AS Capable bezeichneten Geräte erfolgt nur von Port zu Port. Die im Rahmen dieser Arbeit konstruierten AVB-Netzwerke schliessen den in Abschnitt 3.2 vorgestellten AVB-Switch immer mit ein, d.h. eine Kommunikation von Endknoten zu Endknoten erfolgt immer über mindestens einen AVB-Switch.

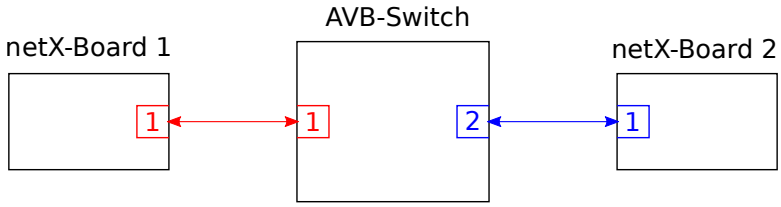

Abbildung 4.1: gPTP Port zum Port Authentifizierung

In Abbildung 4.1 ist ein Beispielnetzwerk mit zwei, der in Abschnitt 3.1 vorgestellten, NetX-Boards und einem AVB-Switch zu sehen. NetX-Board 1 ist an den ersten Port des AVB-Switches angeschlossen, NetX-Board 2 an den zweiten. Beide Boards bilden Peers des AVB-Switches und müssen sich dort durch korrekte gPTP-Kommunikation als 802.1AS Capable ausweisen.

Die im Rahmen dieser Arbeit von AVB benutzen Funktionen betreffen nur Ethernet Layer 2 und schliessen keine höheren Layer mit ein. Eine Uhrensynchronisation z.B. zum Zwecke einer Synchronisation mehrer Streams ist damit unnötig.

Die Anforderung ein NetX-Board gegenüber einem AVB-Switch als 802.1AS Capable erscheinen zu lassen, aber trotzdem nicht das komplette 802.1AS Protokoll umsetzen zu müssen, stellt damit einen Konflikt dar für den das Konzept aus Unterabschnitt 4.3.2 eine Lösung präsentieren soll.

4.2.2 Anforderungen an das SRP

Das Stream Reservation Protocol stellt die Funktionalität bereit Informationen über von Talkern angebotene Streams und deren Listener über das Netzwerk zu verbreiten. Ausserdem werden Bandbreiten für diese Streams auf dem Pfad zwischen Talkern und Listenern auf den dazwischenliegenden AVB-Bridges reserviert.

Um einen Talker zu deklarieren bzw. um diese Deklarierung wieder zu entfernen sind mindestens folgende Funtionen nötig:

- **register_talker(streaminformationen):** deklariert einen Talker bzw. dessen Stream, spezifische Streaminformationen als Übergabeparameter
- **deregister_talker(streaminformationen):** entfernt eine Deklarierung für einen Talker bzw. dessen Stream, spezifische Streaminformationen als Übergabeparameter

Selbiges gilt für Listener:

- **register_listener(streaminformationen):** deklariert einen Listener bzw. den Stream den ein Listener empfangen möchte, spezifische Streaminformationen als Übergabeparameter
- **deregister_talker(streaminformationen):** entfernt eine Deklarierung für einen Listener bzw. den Stream den ein Listener empfangen möchte, spezifische Streaminformationen als Übergabeparameter

Damit dies geschehen kann ist es außerdem notwendig die AVB-Domain bzw. im Falle von SRP die SRP-Domain, aufzubauen, indem die numerischen Werte für die AVB SR Class A bzw. B mittels Domain-Attributen bekannt gemacht werden.

- **register_domain(domaininformationen):** deklariert ein Domain-Attribut mit den passenden Werten.

Wie in Abbildung 2.3 dargestellt, setzt das SRP auf dem MMRP, MVRP und MSRP auf, wobei MMRP optional und in der Standardkonfiguration nicht aktiviert ist. Da diese drei Protokolle Implementationen von MRP darstellen, enthalten diese jeweils eine Menge von State-Machines um die deklarierten bzw. von anderen Knoten deklarierten und lokal registrierten Attribute zu verwalten.

Dabei werden die State-Machines in unterschiedliche Arten und Aufgaben unterteilt:

- **Applicant State Machine:** verwaltet den Zustand eigener, deklarierter Attribute, existiert einmal pro deklarierten Attribut
- **Registrar State Machine:** verwaltet den Zustand fremder, lokal registrierter Attribute, existiert einmal pro registriertem Attribut
- **LeaveAll State Machine:** sorgt mit dem Triggern von allen Applicant und Registrar State Machines des jeweiligen MRP Applicants und dem Versenden entsprechender MRP Nachrichten für das zyklisches Auffrischen der Attribute, existiert einmal pro MRP Applicant
- **PeriodicTransmission State Machine:** ähnlich wie LeaveAll State Machine, triggert aber nur Applicant State Machines, sorgt für regelmässiges Auffrischen von Attributendeklarierungen, exisitiert einmal pro MRP Applicant

Diese State Machines werden von eingehenden MRP Nachrichten getriggert sowie ebenfalls für sie zuständige Timer die regelmässig Events generieren. Die Timerwerte sind in Centisekunden angegeben. Eine Centisekunde entspricht $1*10^{-2}$s.

Für die korrekte MRP Funktion notwendige Timer:

- **Periodic Timer:** generiert Events für die Periodic Transmission State Machine, existiert einmal pro PeriodicTransmission State Machine, Laufzeit ist 100cs
- **Join Timer:** wird gestartet, wenn eine Applicant State Machine eine MRP Nachricht verschicken will, triggert alle anderen Applicant State Machines des selben MRP Applicants, abschicken der Nachricht nach Ablauf des Timers, dient dem

sammeln von MRP Nachrichten von verschiedenen Applicant State Machines innerhalb eins Frames, existiert einmal pro MRP Applicant, Laufzeit ist ein Zufallswert zwischen 0-20cs

- **Leave Timer:** wird gestartet, wenn eine eingehende Attributdeklarierung ausbleibt, Entfernung der lokalen Attributregistrierung nach Ablauf des Timers, existiert einmal pro registriertem Attribut, Laufzeit ist Zufallswert zwischen 60-100cs
- **LeaveAll Timer:** fordert regelmässig entfernte Peers eines MRP Applicants dazu auf, ihre Attribute neu zu deklarieren, dient dem Auffrischen von Attributdeklarierungen, existiert einmal pro LeaveAll State Machine, Laufzeit ist Zufallswert zwischen 1000-1500cs

Der Aufbau des SRP Moduls, als auch das Lösen der Problems der begrenzten Hardware-Ressourcen des Hilscher-Boards, zum Abbilden der hier besprochenen Timer auf echte Hardware, werden im Konzeptteil in Unterabschnitt 4.3.3 geklärt.

4.2.3 Anforderungen an den CBS

Der in Unterabschnitt 2.4.4 vorgestellte Credit Based Shaper soll folgende Funktionen enthalten:

a) Versenden von AVB SR Class A Traffic

b) Versenden von AVB SR Class B Traffic

c) Versenden von Best Effort Traffic

d) Anzahl von AVB SR Class A Streams soll nicht begrenzt sein

e) Anzahl von AVB SR Class B Streams soll nicht begrenzt sein

Desweiteren soll das in IEEE 802.1qav beschriebene zweistufige Queue Modell umgesetzt werden.

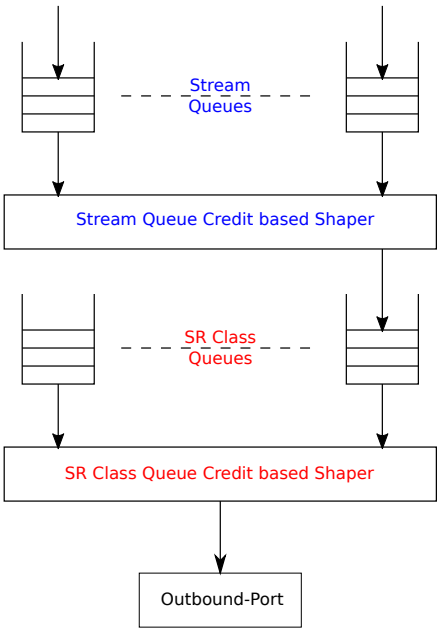

Abbildung 4.2: Zweistufiges Queue Modell (Darstellungsform nach [7])

Abbildung 4.2 zeigt das zweistufige Queue Modell aus Sicht einer einzelnen SR Class. Das zweistufige Queue Modell ist in Stream basierte, hier in blau dargestellt und in SR-klassenbasierte Queues, hier rot dargestellt, Queues unterteilt.

Innerhalb der streambasierten Queues gleicher SR Class, errechnet sich der Credit beinflussenden SendSlope für die jeweilige Queue aus ihren speziefischen Bandbreitenanforderungen. Innerhalb der SR-klassenbasierten Queues errechnet sich der SendSlope aus der Summe der SendSlopes der vorhandenen Stream basierten Queues der jeweiligen Klasse.

Auf diese Weise wird verhindert, das Stream basierte Queues gleicher Priorität sich gegenseitig die Bandbreite wegnehmen und die Reihenfolge des Hinzufügens von Frames in die SR-klassenbasierten Queues gemäss der reservierten Bandbreite erfolgt.

Ein Konzept dafür findet sich in Unterabschnitt 4.3.4.

4.3 Konzept

Die derzeitige Softwarekonfiguration auf dem NetX-Board sieht ein Konfigurieren der Anwendung über eine Konfigurationsdatei namens config.c vor. Diese Konfigurationsdatei wird bereits von Kai Müllers TTE-Stack benutzt und soll von dem zu erstellenden AVB-Stack ebenfalls genutzt werden. Um Konflikte mit TTE zu vermeiden aber trotzdem keine Änderungen an TTE vornehmen zu müssen und diesem im Ursprungszustand belassen zu können, soll TTE mit einer leeren Konfiguration gestartet werden. Auf diese Weise existieren keine konkurierenden Scheduler-Einträge für TTE Tasks, die den AVB-Stack bei der Arbeit unterbrechen könnten. Ein Unterbrechen von TTE durch AVB wird so ebenfalls ausgeschlossen.

4.3.1 Lösungskonzept für Timer

Da dem NetX-Board nur eine begrenzte Anzahl von Hardware-Timern zur Verfügung stehen soll ein Timer-Modul erstellt werden, das es zulässt eine zunächst unbegrenzte Anzahl von logischen Timern auf einen dieses Hardware-Timer abzubilden. Das Timer-Modul soll eine API zum Erstellen, Starten, stoppen und Abfragen von logischen Timern besitzen. Ebenfalls soll die Möglichkeit bereitgestellt werden Callback-Funktionen an logische Timer zu binden und so Funktionen automatisch nach Ablaufen eines Timers ausführen zu können.

4.3.2 Lösungskonzept für das gPTP

Eine komplette Implementation des IEEE 802.1AS Protokolls ist nicht notwendig, da die einzige Anforderung darin besteht vom AVB-Switch als 802.1AS Capable markiert zu werden. Dies kann bereits bezweckt werden indem auf die, vom Switch als gPTP-Master initiierten Laufzeitmessungsnachrichten, die er zu seinen Peers sendet, ordnungsgemäß geantwortet wird und die vom Switch gemessene Laufzeit unterhalb eines einstellbaren Tresholds liegt.

Eine solche Laufzeitmessung ist in Abbildung 4.3 dargestellt.

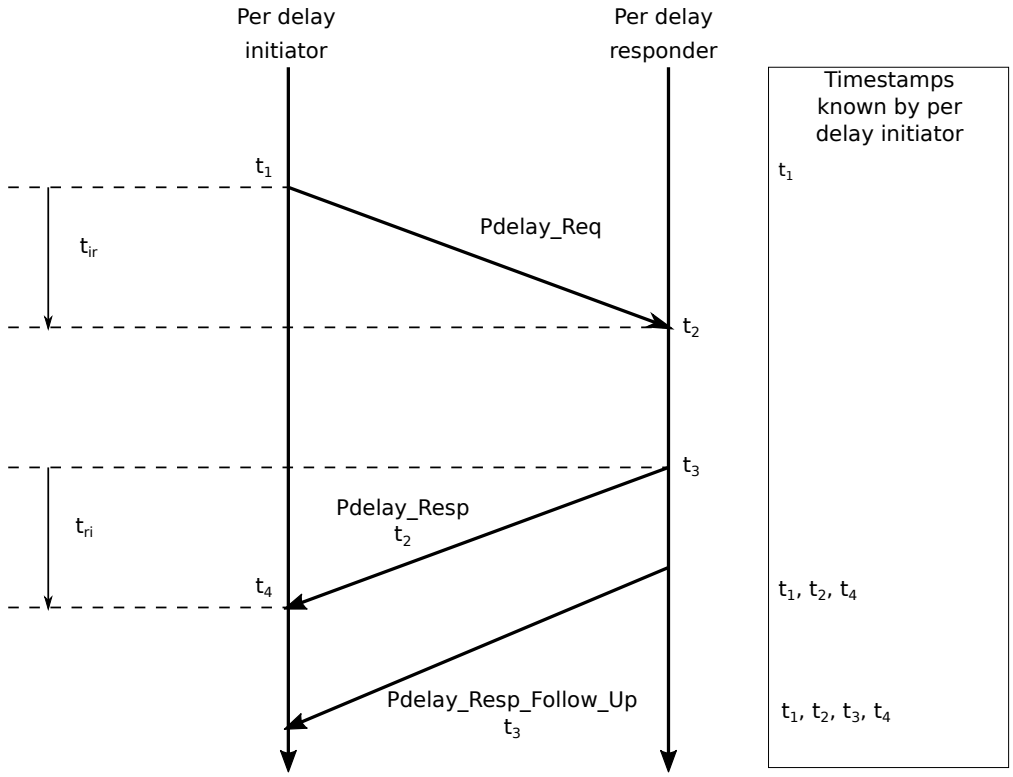

Abbildung 4.3: gPTP Propagation Delay Measurement (Quelle: [9])

Abbildung 4.3 zeigt eine Propagation Delay Messung bei der die Übertragungszeit von einem Endknoten zu einem anderen Endknoten gemessen wird. Bei dieser Messung sendet ein, hier als Per delay initiator bezeichneter Knoten, eine Propagation Delay Request Nachricht an seinen Peer. Mit dieser wird eine Antwort für die Propagation Delay Messung angefordert. Der Zeitpunkt, wann diese Nachricht abgeschickt wird, wird als t_1 festgehalten. Der anwortende Knoten merkt sich den Ankunfszeitpunkt der Propagation Delay Request Nachricht als t_2. Als Antwort wird eine Propagation Delay Response Nachricht verschickt, die den Zeitpunkt t_2 enthält. Der Absendezeitpunkt wird als t_3 vermerkt. Darauf folgt eine weitere Nachricht, die als Propagation Delay Response Follow Up bezeichnet wird und t_3 enthält. Nach Erhalten dieser Nachricht sind dem Per delay initiator t_1-t_4 bekannt und die Zeiträume t_{ir} und T_{ri} können berechnet werden. Diese beiden Zeiträume sind im Idealfall gleichlang und geben die Laufzeit vom Initiator zum Responder bzw. zurück an.

$$t_{\mathrm{ir}} = t_2 - t_1 \tag{4.1}$$

$$t_{\mathrm{ri}} = t_4 - t_3 \tag{4.2}$$

$$D = \frac{t_{\mathrm{ir}} + t_{\mathrm{ri}}}{2} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{4.3}$$

Der in Gleichung 4.3 als D bezeichnete Wert ist die Laufzeit zwischen dem Initiator und dem Responder.

Weitere gPTP-konforme Aktionen wie Teilnahme am Master Clock-Auswahlalgorithmus und eine Uhrensynchronisation werden damit unnnötig.

Das Konzept sieht vor, dass ein Modul geschaffen wird, das einmal gestartet, selbstständig auf Propagation Delay Request Nachrichten wartet und diese mit Propagation Delay Response und Propagation Delay Response Follow Up Nachrichten mit den ensprechenden Zeitpunkten anwortet. Da der Initiator nur Differenzen aus den empfangenen Werten errechnet, ist es auch unerheblich ob die Uhren beim Responder angeglichen also synchronisiert wurden.

4.3.3 Lösungskonzept für das SRP

Das Konzept für SRP sieht die Erstellung eines einzelnen SRP-Moduls vor, daa die in Unterabschnitt 4.2.2 vorgestellte API zum Anlegen/Entfernen von Talkern/Listenern umsetzt. Funktionsaufrufe finden ausschließlich über die SRP API statt. Deweiteren soll das SRP-Modul weitere Sub-Module für MVRP und MSRP beinhalten, die die Kernfunktionen bereitstellen.

vorgesehene API für MVRP:

- **register_vlan(vlan id):** deklariert ein-VLAN Attribut
- **deregister_vlan(vlan id):** entfernt die Deklarierung des VLAN-Attributs

vorgesehene API für MSRP:

- **register_talker(streaminformationen):** deklariert ein Talker-Attribut
- **deregister_talker(streaminformationen):** entfernt die Deklarierung des Talker-Attributs
- **register_listener(streaminformationen):** deklariert ein Listener-Attribut
- **deregister_listener(streaminformationen):** entfernt die Deklarierung des Listener-Attributs
- **register_domain(domaininformationen):** deklariert ein Domain-Attribut

Das SRP-Modul soll selbständig arbeiten. Nach dem Starten des SRP-Moduls sollen automatisch Domain-Attribute mit den dazugehörigen numerischen Werten für AVB SR Class A und B deklariert werden. Beim Deklarieren eines Talkers oder Listeners soll automatisch die dazugehörige VLAN ID mitdeklariert werden. Danach sollen deklarierte und registrierte Attribute selbstständig aktuell gehalten werden, MRP Nachrichten versendet, empfangen und verarbeitet werden. Dies soll transparent für den Anwender erfolgen.

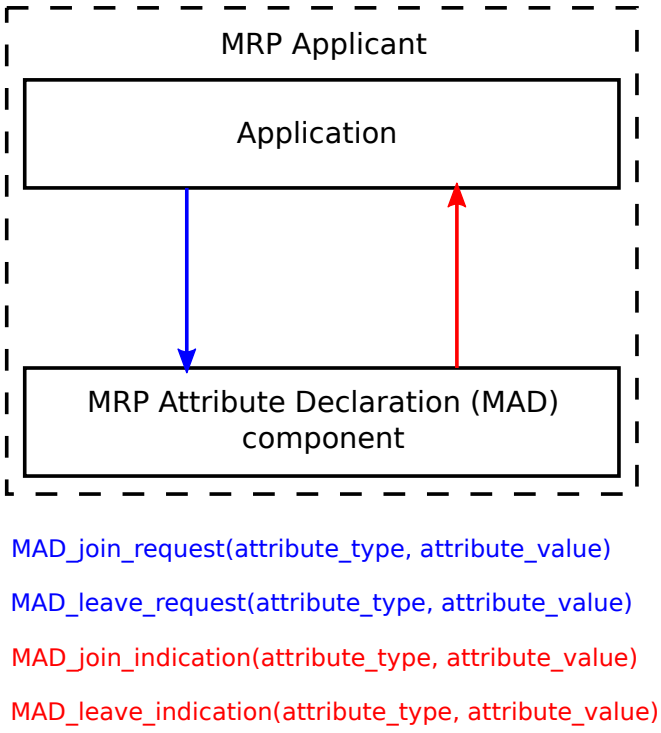

Abbildung 4.4: Der Aufbau eines MRP Applicants wie in IEEE 802.1Q beschrieben

Ein MRP Applicant ist eine konkrete MRP-Anwendung, z.B. die hier besprochenen MVRP, MMRP und MSRP. Diese existiert auf einem unterstützten Gerät einmal pro Port, also z.B. auf einer Bridge mit acht Ports, acht mal. Der MRP Applicant besteht aus zwei Teilen der Application, die die applikationspezifischen Daten und Funktionen enthält und der MRP Attribute Declaration Component kurz MAD Komponente. Die Aufgabe der MAD Komponente besteht darin Attributdeklarierungen und Registrierungen zu verwalten. Hier befinden sich die in Unterabschnitt 4.2.2 beschriebenen State Machines und Timer.

Die Application kommuniziert mit der MAD-Komponente über ein generisches Inter-

face:

- **MAD_join_request(attribute_type, attribute_value):** Aufruf einer MAD Funktion, Deklarierung eines Attributes
- **MAD_leave_request(attribute_type, attribute_value):** Aufruf einer MAD Funktion, entfernen einer Attributedeklarierung
- **MAD_join_indication(attribute_type, attribute_value):** Rückmeldung des MAD Modules bei der Application, ein fremdes Attribut wurde registriert
- **MAD_leave_indication(attribute_type, attribute_value):** Rückmeldung des MAD Modules bei der Application, die Registrierung für ein fremdes Attribut wurde entfernt

Die MAD Komponente kommuniziert mit einem niedrigeren Software Layer (nicht in Abbildung 4.4 abgebildet) und hat so die Möglichkeit eingehende Frames zu verarbeiten und eigene zu verschicken.

Ziel ist es den Aufbau zu übernehmen und eine generische MAD Komponente zu schaffen, um Redundanz zu vermeiden und möglichst viel Code zwischen den verschiedenen MRP Applicants wieder verwenden zu können.

4.3.4 Lösungskonzept für den CBS

Das Konzept für den Credit based Shaper sieht das Umsetzen des in Unterabschnitt 2.4.4 besprochenen Algorithmuses vor. Zudem soll folgender Aufbau der Queues umgesetzt werden:

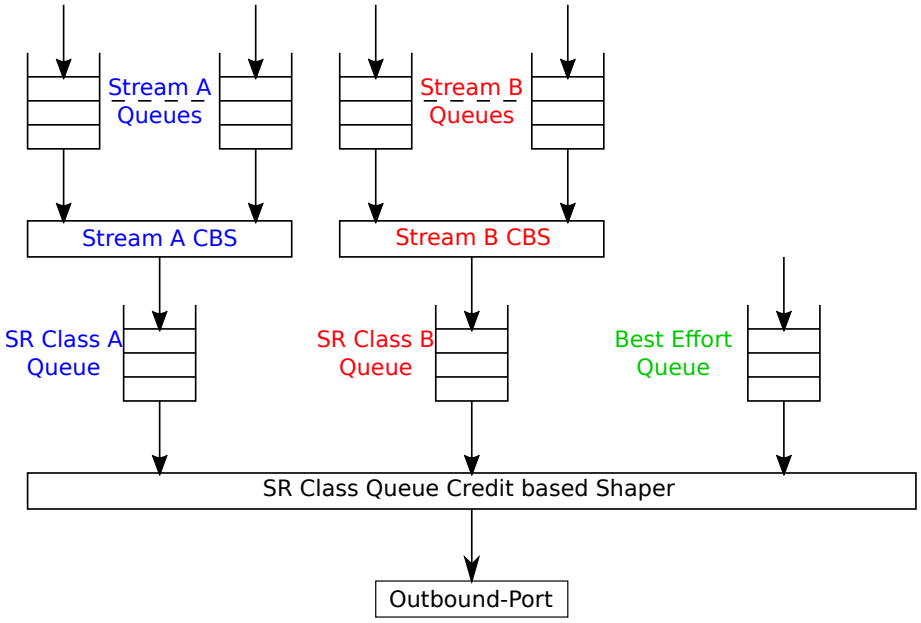

Abbildung 4.5: geplantes Credit based Shaper Modell (Darstellungsform nach [7])

Jeweils ein Credit based Shaper für AVB SR Class A und B basierte Streams. Diese Shaper bestimmen in welcher Reihenfolge die Frames aus diesen Queues in den jeweiligen AVB SR Class A und B Queues einsortiert werden. Die Anzahl der Stream Queues, die verwaltet werden können, soll nicht begrenzt sondern dynamisch sein. In der zweiten Stufe befinden sich jeweils eine Queue für AVB SR Class A und B und eine Queue für Best Effort Traffic. Für Best Effort Traffic exisiert keine erste Stufe in diesem Shaper Modell, da für Best Effort Frames keine unterschiedlichen Queues existieren und diese daher gleichwertig. Der Credit based Shaper wählt aus den drei Queues der zweiten Stufe den Frame, der gesendet werden soll.

5 Implementierung

Dieses Kapitel hat die Beschreibung der Implementierung des im vorherigen Kapitel beschriebenen Konzeptes zum Ziel. Es wird detailiert auf die Funktionsweise der erstellten Module eingegangen. Bei der Implementierung aufgetretene Probleme und deren Lösung werden ebenfalls erläutert.

5.1 Logische Timer

Das in Audio/Video-Bridging enthaltene Stream Reservation Protocol verwendet eine Vielzahl von Timern, um die attributverwaltenen State Machines mit Events zu triggern.

Bei einem deklarierten Talker-Attribut sind das bereits:

Für MVRP und MSRP ohne deklarierte/registrierte Attribute jeweils:

1x Periodic Timer für die MRP-Applikation

1x LeaveAll Timer für die MRP-Applikation

Für das in MVRP deklarierte VLAN-Attribut:

1x Join Timer für deklariertes Attribut

1x Leave Timer für evtl. eingehende Attributregistrierung

Für das in MSRP deklarierte Talker-Attribut:

1x Join Timer für deklariertes Attribut

1x Leave Timer für evtl. eingehende Attributregistrierung

Damit werden schon acht Timer benötigt. Sollte sich ein Listener für den Stream reservieren, so werden weitere zwei Timer von MSRP benötigt. Ein weiterer Timer wird zusätzlich jeweils von dem in Abschnitt 5.3 besprochenen LLC-Modul und dem in Abschnitt 5.4 besprochenen CBS-Modul verwendet. Da das NetX-Board nicht genügend Hardware Timer anbietet und die Menge an möglichen Attributdeklarierungen und Registrierungen nicht durch die Anzahl der verfügbaren Hardware-Timer begrenzt werden soll, ist dieses Timer-Modul erstellt worden, das eine beliebige Menge an logischen Timern auf einen einzelnen Hardware Timer abbildet.

Der auf dem NetX-Board verwendete Hardware-Timer hat eine Auflösung von 10ns und benutzt ein 32Bit breites vorzeichenloses Zählregister. Damit ergibt sich folgende maximale Zählweite:

$$\text{max. Zählwert} = \text{Registerbreite} \times \text{Auflösung} \tag{5.1}$$

$$42949672960ns = 2^{32} \times 10ns \approx 42,9s \tag{5.2}$$

Mit der gegebenen maximalen Zählweite und Präzission lassen sich alle zeitlichen Anforderungen des Stacks einhalten. Zusätzlich lassen sich die logischen Timer für Utility-Funktionen, wie das verzögerte oder periodische Ausführen von Code, benutzen. Dies kann durch Pollen des Status eines Timers geschehen oder automatisch in dem eine auszuführende Callback-Funktion mit maximal einem Parameter an den Timer übergeben wird.

Die API des Timer-Modules bietet folgende Funktionen um Timer zu manipulieren:

- **navb_timer_init_timer():** initialisiert einen Timer, muss für jeden Timer einmal vor Benutzung ausgeführt werden
- **navb_timer_set_timer():** setzt die Zeit bis zum Ablaufen eines Timers, hat die Möglichkeit das automatische Neustarten eines Timers zu setzen, enthält ausserdem eine mögliche Callback-Funktion mit oder ohne Parameter
- **navb_timer_start_timer():** startet einen Timer
- **navb_timer_stop_timer():** stoppt einen Timer
- **navb_timer_is_running():** überprüft ob ein Timer gerade läuft

- **navb_timer_is_expired():** überprüft ob ein Timer abgelaufen ist
- **navb_timer_process_callback():** startet angefallende Callback-Funktionen, wird nicht direkt vom Benutzer aufgerufen, sondern läuft als Funktionsaufruf in der bg_task() von config.c, pollt den Callback-Fifo
- **navb_timer_remove_callback():** löscht angefallene Callback-Funktionen für einen spezifischen Timer

5.1.1 Arbeitsweise des Timer-Moduls

Timer bestehen aus Variablen des Typs navb_timer_t (siehe Listing 5.1). Diese enthalten in der Variable timeout den für den Timer eingestellten Timeout. Das Timer-Modul verwaltet eine verkettete Liste aller gestarteten Timer. Diese sind nach ablaufenden Timeouts sortiert. Der Hardware-Timer ist jeweils auf den Timer am Kopfende dieser Liste eingestellt. Der in Variable runtime gespeicherte Wert enhält die Zeit, die der Hardware-Timer für diesen Timer, bis zum Neustarten für den nächsten Timer laufen wird. Ist ein Timer abgelaufen, so wird er aus der Liste entfernt oder ggf. neugestartet und wieder in die Liste einsortiert. Ist die Callback-Funktion an den abgelaufenen Timer gebunden, so wird dieser in einer Fifo-Datenstruktur gespeichert. Die Funktion navb_timer_process_callback() arbeitet diese Callback-Funktionen in der config.c ab.

Listing 5.1: Datenstruktur für einen logischen Timer

```
typedef struct {
  uint32_t timeout;
  volatile uint32_t runtime;
  volatile bool expired;
  bool oneShot;
  void (*ptFuncPtr)(void*);
  void* ptFuncParam;
  volatile struct navb_timer_t* nextTimer;
} navb_timer_t;
```

Starten und Stopppen eines logischen Timers

Abbildung 5.1 zeigt exemplarisch das Starten eines weiteren logischen Timers. Zur einfacheren Darstellung ist die Timerauflösung in Sekunden angegeben und die schon aktiven logischen Timer sind alle zur gleichen Zeit gestartet worden. Die derzeitig laufenden Timer sind: Timer A mit 5s Laufzeit, Timer B mit 10s Laufzeit und Timer C mit 15s Laufzeit. Für jeden dieser Timer soll der Hardware-Timer 5s laufen. Nachdem eine Sekunde vergangen ist respektive der Hardware-Timer 1s gelaufen ist, soll ein neuer logischer Timer gestartet werden. Timer D hat eine Laufzeit von 7s. Es wird nun die Stelle in der Liste gesucht, bei der die angebene Runtime in Timer D grösser ist als die aufsummierten Runtimes der vorhergehenden Timer abzüglich des aktuellen Hardware-Timer-Standes, aber auch kleiner als das Timeout eines nachfolgenden Timer. Diese Stelle findet sich zwischen Timer A und Timer B. Die Runtime von Timer D ist, Timeout von Timer D minus Runtime von Timer A minus des aktuellen Hardware-Timer Stands. Das gleiche Muster gilt für Timer B. Für die neue Runtime von Timer B muss die Runtime von Timer D und der aktuelle Hardware-Timer Stand abgezogen werden.

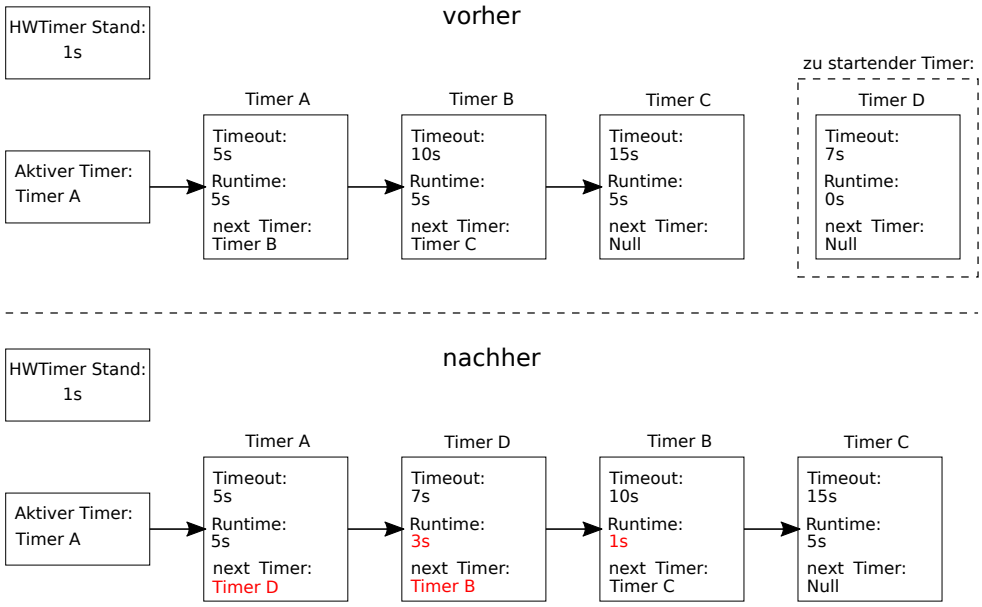

Abbildung 5.1: Starten eines logischen Timers

Ein Löschen eines logischen Timers respektive löschen eines Timers aus der Liste der aktiven Timer erfolgt auf ähnliche Weise. Die Runtime des zu löschenden Timers wird auf die Runtime des nachfolgenden Timers addiert.

5.1.2 Präzission der logischen Timer

Ein Starten oder Stoppen eines logischen Timers verändert die Liste der aktiven Timer. Beim Entwickeln des Timer Moduls kam es zum Auftreten von Race-Conditions, die dadurch verursacht worden sind, dass ausgeführter Code der Start- und Stop-Funktionen von der Interrupt Service Routine des Hardware-Timers unterbrochen wurde, der ebenfalls die Liste der aktiven Timer manipuliert hat. Auf diese Weise kam es zu inkonsistenten Zuständen. Um dieses Problem zu umgehen sind Code-Abschnitte innerhalb der Start- und Stop-Funktionen als kritische Abschnitte definiert worden, die es vor Unterbrechung zu schützen galt. Ein Verändern der Hardware-Timer Konfiguration führte nur zum Verlorengehen von Events und somit nicht zur Lösung. Letztendlich wurde der Weg des Anhaltens des Hardware-Timer bei Eintritt und ein Aktiveren des Hardware-Timers bei Austritt aus dem kritischen Abschnitts gewählt.

Dies hat zur Folge, dass sich die tatsächliche Latenz eines logischen Timers zur im Timer eingestellten Zeit unterscheidet, wobei erstere die längere ist. Dies wurde in Anbetracht, das dieses Verhalten nur bei massivem Starten und Stoppen sehr vieler Timer grössere Auswirkungen zeigt und dieser Anwendungsfall nicht Teil des normalen Betriebs dieses AVB-Stacks ist in Kauf genommen.

Da die ISR des Hardware-Timers keinen an die Timer gebundenen Code ausführt, kann zwischen Ausführung dieses Codes und dem Auslaufen eines Timers Zeit vergehen. Dies macht sich in der Form von Jitter bei den Code-Ausführungszeiten bemerkbar, da Event-basierter Code, z.B. Callback-Funktonen in der Regel in der Funktion bg_task() ausgeführt wird und diese durch Code der mit höherer Priorität läuft, unterbrochen bzw. verzögert werden kann. Ausserdem kann gerade anderer Event-basierter Code ausgeführt werden, während ein neues Event gerade auftritt. Das Ausführen des Codes des neuen Events würde damit ebenfalls verzögert.

5.2 Erweiterung des Bufferpools

Der von Kai Müller, zur effizienten Verwaltung eingehender und ausgehender Frames, entworfene Bufferpool wurde dahingehend erweiter,t dass AVB-Frames ebenfalls verwaltet werden können. Dazu sind die in Abbildung 5.2 rot gefärbten Buffer hinzugefügt worden.

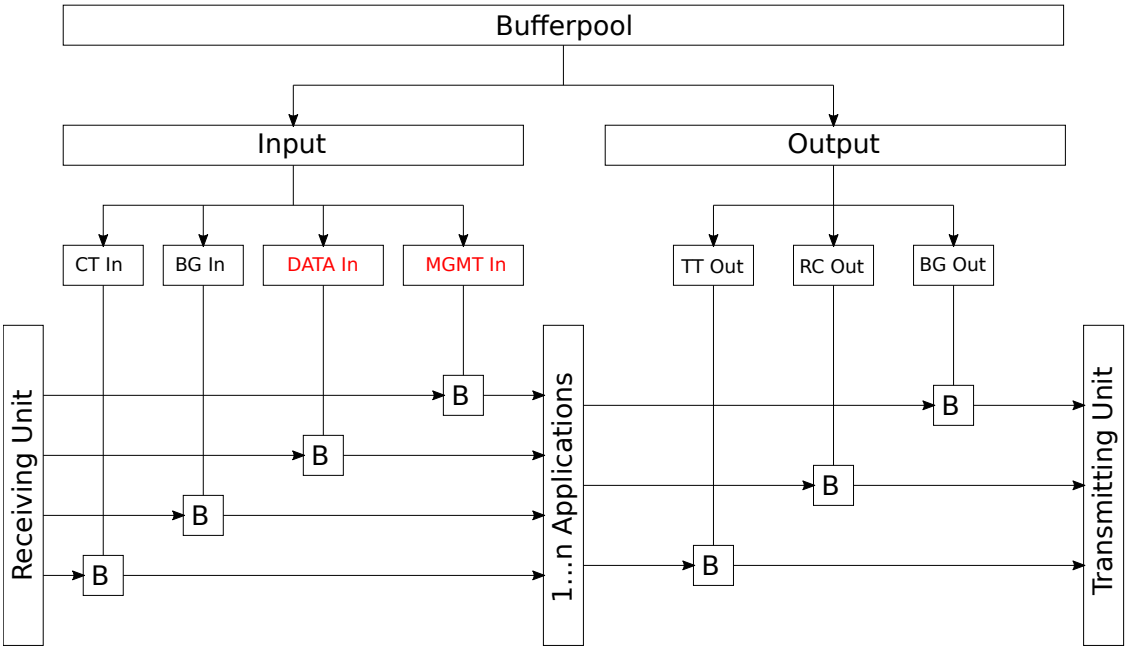

Abbildung 5.2: Die Erweiterung des Bufferpools (Darstellungsform nach [16])

DATA In enthält jeweils einen Buffer für eingehenden AVB SR Class A und B Traffic. Die Zugehörigkeit zu einer bestimmten AVB SR Class ist das einzige Unterscheidungsmerkmal für eingehende Frames. Frames, die zu unterschiedlichen Streams der gleichen AVB SR Class gehören, werden damit auch in dem selben Buffer, nach dem FIFO-Prinzip einsortiert. Diese Lösung wurde gewählt, da zum Zeitpunkt der Konfiguration der Buffer, im Vergleich zu TTE, die Streams IDs noch nicht bekannt sind und diese nicht einem bestimmten Buffer zugeordnet werden können. Ein nachträgliches Mappen einer Stream ID auf einen generischen AVB SR Class A oder B Eingangsbuffer führt zu erhöhtem Auf-

wand bei der Verarbeitung bzw. Einsortierung in den Bufferpool, was sich ein erhöhter maximaler Latenz der Verarbeitungszeit der einsortierenden ISR niederschlägt.

MGMT In ist ein Bufferpool mit vier Buffern, für die für AVB notwendigen Sub-Protokolle MVRP, MMRP, MSRP und gPTP, hier einfach als Management bezeichnet. Diese Buffer enthalten Frames der Priorität BE, sind aber zum einfacheren Zugriff in eigene Buffer ausgelagert worden. Auf diese Weise wird garantiert, dass z.B. die Funktion zum Verarbeiten von gPTP-Frames bei Zugriff auf den Buffer ausschliesslich gPTP-Frames bekommt.

5.2.1 TTE Background Buffer als Ausgangsbuffer

Wie in Abbildung 5.2 zu sehen ist, wurden keine AVB spezifischen Ausgangsbuffer angelegt. Dies war unnötig, da das unterschiedlich priorisierte Versenden von Frames in AVB nicht von einem spezifischen Speicherort abhängig ist. Frames unterschiedlicher Priorität können im selben Buffer gespeichert werden. Eine Priorisierung findet innerhalb des Credit based Shapers (siehe Abschnitt 5.4) statt. Eine Vorauskonfiguration der Buffer, wie dies bei den TT oder RC Buffern von TTE ist, bei dem BG Buffer nicht notwendig. Es existiert ein einzelner BG Buffer mit einer festen maximalen Anzahl Frames.

5.3 Link Layer Controller

Das Link Layer Controller Modul oder kurz LLC bildet ein Interface zum Senden von Frames. Es ist aus der Idee heraus entstanden den Zugriff auf die Hardware und den Bufferpool zu vereinfachen und so weniger fehleranfällig zu machen. Das Anfordern und Freigeben von Speicher für Frames, das Öffenen und Schliessen des Bufferpools und letztendlich das Senden des Frames wird auf wenige Funktionen abstrahiert. Der Name des Moduls stammt aus IEEE 802.1Qat. Dort kommuniziert die MAD-Komponente eines MRP-Applicants über das LLC-Interface mit der Hardware. Dort ist die LLC-Schicht des OSI-Modells gemeint, welches hier nicht umgesetzt ist.

Das LLC stellt folgende Datenstrukturen bereit:

Listing 5.2: Datenstruktur für einen LLC Frame Header

```
typedef struct {
  uint8_t dstAddr[6];
  uint8_t srcAddr[6];
```

```
  uint16_t etherType;
} navb_llc_frame_header_t;
```

Ein navb_llc_frame_header_t repräsentiert einen Standard-Ethernet Header der die notwendige Ziel- und Absenderadresse und den zum Frame gehörigen Ethertype enthält.

Listing 5.3: Datenstruktur für einen LLC Frame Handler

```
typedef struct {
  struct sys_frame_t* pptSysFrame;
  unsigned char* frameData;
  uint32_t* frameLength;
  bool open;
} navb_llc_frame_handler_t;
```

Ein navb_llc_frame_handler_t ist ein Handler für einen einzelnen Frame. Er beinhaltet einen Pointer auf den angeforderten innerhalb des Bufferpools geöffneten Frame. frameData ist ein Pointer, der direkt auf das Datenfeld des Frames zeigt. frameLength wiederum zeigt auf die Variable, die die Länge des Frames angiebt und dient gleichzeitig als Offset für frameData. Diese muss spätestens vor dem Versenden korrekt gesetzt werden.

Folgende API-Funktionen werden von dem LLC-Modul angeboten:

- **navb_llc_init():** initialisiert das LLC-Modul
- **navb_llc_init_handler():** initialisert einen Handler, muss für jeden Handler einmal vor Benutzung ausgeführt werden
- **navb_llc_getFrame():** dient zum Holen und Bereitmachen für das Beschreiben eines Frames, öffnet einen Frame im Background Traffic Buffer, fordert Speicher für einen Frame von der HAL an, bindet diesen an den Background Traffic Buffer, kopiert die Headerinformationen in den Frame, setzt Längen- und Datenfeldpointer, markiert Handler als geöffnet
- **navb_llc_sendFrame():** dient zum Senden eines Frames, überpruft Frameläge und erhöht diese ggf. auf die Minimallänge eines Standard Ethernetframes, schliest den Buffer, setzt den Sende-Timer auf die übergebene Zeit, startet die

HAL-Sendefunktion, startet den Sende-Timer, gibt Handler für nächsten Frame frei

- **navb_llc_releaseFrame():** dient zum Freigeben eines Frames bei nicht gewolltem Sendevorgang, gibt Speicher mittels HAL-Funktion wieder frei, schliesst Buffer, gibt Handler für nächsten Frame frei

Beim Versenden eines Frames wird ein Timer des Typs navb_timer_t gestartet. Läuft dieser Timer aus wird eine Callback-Funktion aufgerufen, die den Speicher, der für den Frame verwendet wurde wieder freigibt. Ebenfalls kann bei Bedarf eine in der Initialisierungsphase des LLC-Moduls übergebene Callback-Funktion aufgerufen werden und so eine Rückmeldung an den Aufrufer der Sendefunktion gegeben werden.

Auf diese Weise ist mit dem LLC-Modul, der aus vielen Einzelschritten bestehende Vorgang des Holens eines Frames und Senden des Selbigen auf die beiden Funktionen navb_llc_getFrame() und navb_llc_sendFrame() reduziert worden.

5.4 Implementierung des Credit based Shapers

Um die Komplexität bei der Umsetzung des Credit based Shapers zu verringern, ist im Laufe der Entwicklung eine Vorversion des Shaper Moduls entstanden. Die Aufgabe dieser Vorversion ist es den Shaper Algorithmus in einem kleineren Rahmen zu implementieren und so schneller zu lauffähigem Code zu gelangen. Die Vorversion des Shapers wird FQTSS Version 0 genannt. FQTSS ist die im IEEE 802.1Qav verwendete Bezeichnung und steht für „Forwarding and Queuing Enhancements for Time-Sensitive Streams“. Im Verlauf dieser Arbeit wird das FQTTS Version 0 Modul auf Grund besserer Lesbarkeit als CBS-Modul bezeichnet.

Das CBS-Modul kann eine einzelne Queue von AVB SR Class A Frames und eine weitere Queue mit der Priorität Best Effort verwalten. Die AVB SR Class A Queue kann als Queue für einen einzelnen AVB SR Class A Stream oder als Queue für mehrere AVB SR Class A Streams gesehen werden. In diesem Falle ist die Summe der AVB SR Class A Stream-Bandbreiten die für die Queue konfigurierte Bandbreite. Die AVB SR Class A Queue ist statisch, kann also nicht während der Laufzeit entfernt oder hinzugefügt werden. Die zur Verfügung stehende Bandbreite der AVB SR Class A Queue wird ebenfalls statisch über Defines konfiguriert.

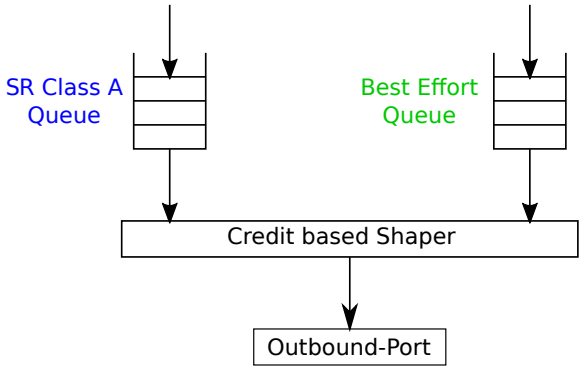

Abbildung 5.3: Credit based Shaper Version 0 (Darstellungsform aus [7])

Abbildung 5.3 zeigt das im CBS Version 0 Modul umgesetzte Queue Modell. Im Vergleich zur ursprünglich geplanten Version 1 (siehe Abbildung 4.2) besteht das CBS Version 0 Modul nur aus der zweiten Stufe und hat ebenfalls keine Unterstützung für AVB SR Class B Streams.

Aus Zeitgründen wurde eine Version 1 des CBS Moduls nicht mehr umgesetzt und die vorhandene Version 0 stellt vorerst die einzig verfügbare Version dar. Eine einzelne AVB SR Class A Queue stellt zwar eine Limitierung dar, jedoch hat bereits ein einzelner AVB SR Class A Stream konfiguriert mit minimaler Framesize von 46 Bytes und einem einzigen Frame pro Class Measurement Interval, mit in etwa 5.3Mbit ausreichend Bandbreite für Steuerungsaufgaben zur Verfügung.

5.4.1 Interface des CBS Version 0

Aus Anwendersicht stellt sich die API des CBS als eine Art Wrapper für das LLC Modul dar, da eine grosse Ähnlichkeit zwischen beiden APIs besteht.

Folgende Datenstrukturen werden angeboten:

Listing 5.4: Datenstruktur für einen CBS Frame Handler

```
typedef struct {
  navb_llc_frame_handler_t frame;
  struct navb_fqtss_ver0_frame_handler_t* nextEntry;
  unsigned char* frameData;
  uint32_t* frameLength;
  bool queued;
} navb_fqtss_ver0_frame_handler_t;
```

Der CBS Handler enthält einen LLC Handler und zusätzlich die Möglichkeit eine verkettete Liste von CBS Handlern zu erstellen.

Folgende Funktionen werden angeboten:

- **navb_fqtss_ver0_init():** initialisert das CBS-Modul
- **navb_fqtss_ver0_getFrame():** wrapped LLC getFrame() Aufruf, fügt CBS-Funktionalität hinzu, ansonsten logisch identisch
- **navb_fqtss_ver0_sendFrame():** wrapped LLC sendFrame() Aufruf, fügt CBS-Funktionalität hinzu, ansonsten logisch identisch, Möglichkeit einen Frame als AVB Class A oder BE abschicken zu können
- **navb_fqtss_ver0_releaseFrame():** wrapped LLC releaseFrame() Aufruf, fügt CBS-Funktionalität hinzu, ansonsten logisch identisch

Der Parameter zum Absenden von Frames als AVB SR Class A oder BE, hat nur Einfluss auf das Verhalten des CBS Algorithmuses und nicht auf den Inhalt eines Frames. Zum Beispiel ist es möglich einen Frame mit BE Inhalt mit AVB SR Class A Priorität zu verschicken und das gleiche gilt ebenfalls für Frames mit BE Inhalt, die mit AVB SR Class A Priorität verschickt werden können.

5.4.2 Funktioneller Aufbau des CBS Version 0 Moduls

Der CBS-Algorithmus ist innerhalb einer State Machine implementiert und ist somit über Events getrieben. Das CBS-Modul verwaltet zwei verkettete Listen von im vorherigen Abschnitt vorgestellten navb_fqtss_ver0_frame_handler_t Elementen. Eine Liste repräsentiert die AVB SR Class A Queue, die andere BE.

Die Queues haben den folgenden Aufbau:

Listing 5.5: Datenstruktur für die im CBS verwendeten Queues

```
typedef struct {
  int maxFrameSize;
  int maxIntervalFrames;
  int credit;
  int idleSlope;
  int sendSlope;
  navb_fqtss_ver0_frame_handler_t* head;
  navb_fqtss_ver0_frame_handler_t* tail;
  unsigned int frameCount;
} navb_fqtss_ver0_queue_t;
```

maxFrameSize und maxIntervalFrames sind die beiden Parameter aus denen die zur Verfügung stehende Bandbreite für diese Queue berechnet wird. credit stellt den aktuellen Credit der Queue dar. idleSlope und sendSlope sind die beiden Parameter, die den Credit verändern. Elemente werden am tail in die Queue eingefügt und am head entnommen. frameCount gibt die Menge in der Queue befindlichen navb_fqtss_ver0_frame_handler_t Elemente an. Es wird eine identische Queue Datenstruktur für die AVB SR Class A Queue als auch für die BE Queue verwendet. Nicht alle Attribute finden bei der BE Queue Verwendung. Es werden dort lediglich die Variablen head, tail und frameCount verwendet.

weitere wichtige Veriablen des CBS-Moduls:

Listing 5.6: Variablen des CBS

```
unsigned int portTransmitRateByteS;
navb_fqtss_ver0_queue_t AQueue;
navb_fqtss_ver0_queue_t BeQueue;
navb_fqtss_ver0_queue_t* currentSendingQueue;
navb_fqtss_ver0_frame_handler_t fqtssFrames[];
uint64_t sysTimeT1;
uint64_t sysTimeT2;
navb_timer_t idleTimer;
navb_fqtss_ver0_state_t cbsState;
```

portTransmitRateByteS gibt die Transferrate des Ethernet-Ports in Byte/S an. Dieser Wert ist notwendig, um Sendezeiten für Frames ausrechnen zu können. AQueue und BeQueue sind die beiden Queues. currentSendingQueue ist ein Pointer auf eine von Beiden, falls gerade gesendet wird. fqtssFrames ist ein Array von Frame Handlern von denen jeweils einer mit navb_fqtss_ver0_getFrame() angefordert werden kann, falls noch ein freier Handler vorhanden ist. Die Länge des Array kann mit dem Define NAVB_FQTSS_VER0_MAX_QUEUED_FRAMES konfiguriert werden und gibt damit ebenfalls die maximale Anzahl von offenen Frames an, die das System für ausgehenden Verkehr zulässt. sysTimeT1 und sysTimeT2 bezeichnen aufeinander folgende Zeitpunkte an dem jeweils ein Event aufgetreten ist. cbsState hält den aktuellen State der State Machine.

Die State Machine des CBS-Moduls besitzt folgende States:

- **EMPTY:** es wird kein Frame gesendet und beide Queues sind leer
- **SENDING:** es wird ein Frame aus einer der beiden Queues gesendet, die Information aus welche Queue gerade gesendet wird ist nicht im Zustand gespeichert, daher zeigt der Zeiger currentSendingQueue auf die sendende Queue
- **IDLE:** es wird nicht gesendet, die BE Queue enthält keinen weiteren Frame, die AVB SR Class A Queue hat einen Credit < 0 und muss daher warten bis credit wieder $>= 0$, es kann ein weiterer Frame in der AVB SR Class A Queue vorliegen

Und folgende Events:

- **FRAME_A_QUEUED:** navb_fqtss_ver0_sendFrame() wurde mit dem entsprechenden Parameter aufgerufen und ein neuer Frame ist in der AVB SR Class A Queue aufgenommen worden
- **FRAME_BE_QUEUED:** navb_fqtss_ver0_sendFrame() wurde mit dem entsprechenden Parameter aufgerufen und ein neuer Frame ist in der BE Queue aufgenommen worden
- **FRAME_A_FINISHED_TRANSFERING:** ein AVB SR Class A Frame ist fertig transferiert worden
- **FRAME_BE_FINISHED_TRANSFERING:** ein BE Frame ist fertig transferiert worden

- **QUEUE_A_CREDIT_REACHED_0:** der idleTimer für die AVB SR Class A Queue ist ausgelaufen, was bedeutet, dass der Credit jetzt wieder >= 0 ist, falls ein weiterer Frame in AVB SR Class A vorliegt und die State Machine nicht schon im SENDING State ist, geht diese in diesen Zustand über, und sendet den Frame wenn möglich, ansonsten geht die State Machine in den EMPTY State über

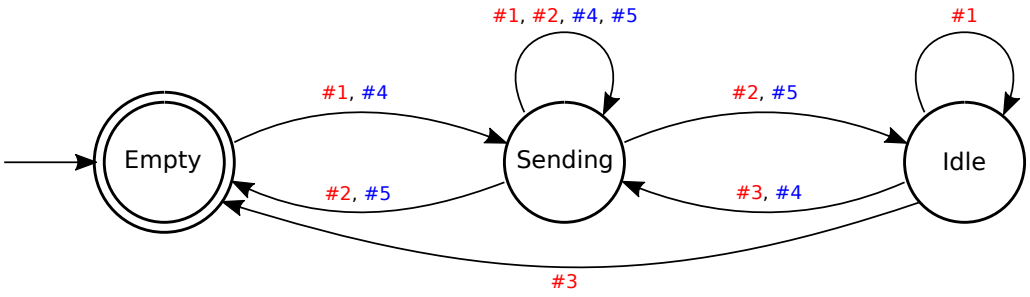

Abbildung 5.4: Die grafische Darstellung der implementierten CBS State Machine

Für einen reibungslosen Funktionsablauf benötigt das CBS-Modul eine Reihe lokaler Funktionen.

Diese sind:

- **navb_fqtss_ver0_calc_transfer_time():** berechnet die Transferzeit einer angegebenen Anzahl von Bytes
- **navb_fqtss_ver0_calc_idle_time():** berechnet die Zeit bis ein Credit wird den Wert 0 erreicht, dies geschiet auf Basis des aktuellen Creditwerts und des Idle Slopes

- **navb_fqtss_ver0_calc_credit_change():** berechnet das Delta einer Creditwertveränderung unter Angabe eines Idle- oder Sendslopes und Angabe einer Zeit
- **navb_fqtss_ver0_calc_actual_bandwidth():** berechnet die tatsächllich verbrauchte Bandbreite unter Angabe von maxFrameSize und maxIntervalFrames
- **navb_fqtss_ver0_add_frame_to_queue():** Utility-Funktion die einen Handler zu einer Queue hinzufügt
- **navb_fqtss_ver0_remove_frame_from_queue():** Utility-Funktion die einen Handler von einer Queue entfernt
- **navb_fqtss_ver0_cbs_credit_zero_callback():** Callback-Funktion, die an den Idle-Timer gebunden ist und das Event QUEUE_A_CREDIT_REACHED_0 und die State Machine sendet, wenn dieser ausläuft
- **navb_fqtss_ver0_cbs_transfer_finished_callback():** wird in der Initialisierungsphase an die Init-Funktion des LLC Moduls übergeben, wird aufgerufen, wenn das LLC Modul einen Transfer abgeschlossen hat, sendet das Event FRAME_A_FINISHED_TRANSFERING oder FRAME_BE_FINISHED_TRANSFERING an die State Machine in Abhängigkeit auf welche Queue die Variable currentSendingQueue zeigt
- **navb_fqtss_ver0_cbs():** die Funktion, die das Verhalten der State Machine umsetzt

Arbeitsweise des CBS-Moduls anhand eines Beispiels

Das folgende einfache Beispiel soll die Arbeitsweise des CBS-Moduls und dessen State Machine verdeutlichen. Zur Anschauung dient ein vereinfachtes Anwendungsbeispiel aus Abbildung 2.7. In diesem Falle wird ein einzelner AVB SR Class A Frame gesendet.

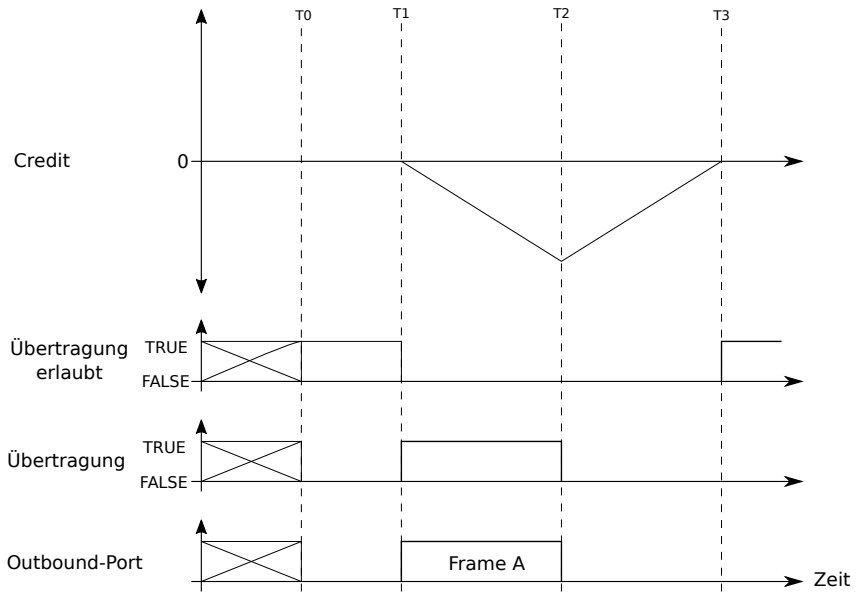

Abbildung 5.5: Anwendungsbeispiel für die Arbeistweise des CBS-Moduls (Darstellungsart nach [7])

T0 Die Funktion navb_fqtss_ver0_init() wird aufgerufen und initialisiert das Modul. Zusätzlich wird die Initialisierungsfunktion des darunterliegenden LLC-Moduls aufgerufen und auf diese Weise die Funktion navb_fqtss_ver0_cbs_transfer_finished_callback() als Callback Funktion eingetragen. maxFrameSize und maxIntervalFrames werden mit den vordefinierten Werten aus den jeweiligen Defines gefüllt. Daraus wird die tatsächlich verbrauchte Bandbreite mittels navb_fqtss_ver0_calc_actual_bandwidth() berechnet und der Wert in die Variable idleSlope der AVB SR Class A Queue geschrieben. sendSlope berechnet sich aus idleSlope - NAVB_FQTSS_VER0_PORT_TRANSMIT_RATE. Diverse Variablen werden ebenfalls initialisiert und die State Machine in des Zustand EMPTY versetzt.

T1 Die Funktion navb_fqtss_ver0_sendFrame() wurde ausgeführt. Die Funktion navb_fqtss_ver0_add_frame_to_queue() fügt den übergebenen Frame Handler am tail der Queue ein. Der CBS Algorithmus in der Funktion

navb_fqtss_ver0_cbs(), wird mit dem Event FRAME_A_QUEUED getriggert. Die Zeit für des auftretende Event wird gespeichert. Die Funktion navb_fqtss_ver0_calc_transfer_time() errechnet aus der Grösse des Frames die Transferzeit. Die AVB SR Class A Queue wird als sendende Queue markiert. Die Funktion navb_llc_sendFrame() des darunterliegenden LLC Moduls wird mit der ausgerechneten Transferzeit als Übergabeparameter aufgerufen. Der Zustand der State Machine wechselt zu SENDING.

T2 Der Send Timer des LLC-Moduls ist ausgelaufen und die Callback-Funktion navb_fqtss_ver0_cbs_transfer_finished_callback() wird aufgerufen. Da die AVB Class A Queue sendende Queue ist, wird das Event FRAME_A_FINISHED_TRANSFERING and die State Machine gesendet. Die aktuelle Systemzeit wird gespeichert. Der sich verändernde Credit wird mit der Funktion navb_fqtss_ver0_calc_credit_change() und der beiden gespeicherten Zeitpunkte errechnet und der aktuelle Credit angepasst. Da der Credit der AVB SR Class A Queue negativ ist, und deshalb nicht mehr senden kann, wird der Idle Timer mit dem errechneten Wert der Funktion navb_fqtss_ver0_calc_idle_time() gestartet. Die State Machine wechselt in den Zustand IDLE, da ebenfalls kein BE Frame zm senden vorhanden ist.

T3 Der Idle Timer ist ausgelaufen und die Callback-Funktion navb_fqtss_ver0_cbs_credit_zero_callback() sendet das Event QUEUE_A_CREDIT_REACHED_0 and die State Machine. Die aktuelle Zeit wird gespeichert. Der Credit wird auf 0 gesetzt. Da keine weiterer Frame zum Versenden vorhanden ist begiebt sich die State Machine in den Zustand EMPTY.

5.5 Implementierung des gPTP

Das gPTP-Modul dient dazu, wie in Unterabschnitt 2.4.2 und Unterabschnitt 4.3.2 beschrieben, von dem AVB-Switch als gPTP-Capable markiert und so Teil der gPTP-Domain zu werden. Um gPTP-Frames empfangen zu können wurde wie in Abbildung 5.2 dargestellt, innerhalb des MGMT In Bufferpools ein Buffer für eingehende gPTP-Frames geschaffen. Da nur auf gPTP-Frames vom Typ Path Delay Request geantwortet werden soll, werden nur diese innerhalb des Buffers gespeichert und die restlichen gPTP-Frames direkt nach Empfang, innerhalb der Emfpangs-ISR des NetX-Boards, verworfen.

Die zum gPTP-Modul gehörende Funktion navb_gptp_process_incoming_gptp_frame() überprüft den Buffer auf vorliegende Frames und sendet dementsprechend zuerst einen

Path Delay Response Frame und danach einen Path Delay Response Follow Up Frame als Antwort. Da die Funktion in der Background Task der config.c aufgerufen wird, ergibt sich dadurch ein Pollen des Buffers.

Der zu beantwortende Path Delay Request Frame und der folgende Path Delay Response und Path Delay Response Follow Up Frame haben die folgende Form:

Header:

Listing 5.7: Header eines gPTP-Frames (1)

```
typedef struct {
  uint8_t clockId[8];
  uint16_t port;
} navb_gptp_message_sourcePortIdentity_t;
```

Listing 5.8: Header eines gPTP-Frames (2)

```
typedef struct {
  uint8_t transportSpecificMessageType;
  uint8_t reservedVersionPTP;
  uint16_t messageLength;
  uint8_t domainNumber;
  uint8_t reserved1;
  uint16_t flags;
  uint64_t correctionField;
  uint32_t reserved2;
  navb_gptp_message_sourcePortIdentity_t sourcePortIdentity;
  uint16_t sequenceId;
  uint8_t control;
  uint8_t logMessageInterval;
} navb_gptp_message_header_t;
```

Der Header ist für alle drei Frametypen identisch im Aufbau, während die Nutzlast sich für alle drei Typen unterscheidet.

Listing 5.9: Body eines Path Delay Request Frames

```
typedef struct {
  navb_gptp_message_header_t header;
```

```
  uint8_t reserved1[10];
  uint8_t reserved2[10];
} navb_gptp_path_delay_request_message_t;
```

Listing 5.10: Body eines Path Delay Response Frames

```
typedef struct {
  navb_gptp_message_header_t header;
  uint8_t requestReceiptTimestamp[10];
  navb_gptp_message_sourcePortIdentity_t requestingPortIdentity;
} navb_gptp_path_delay_response_message_t;
```

Listing 5.11: Body eines Path Delay Response Follow Up Frames

```
typedef struct {
  navb_gptp_message_header_t header;
  uint8_t responseOriginTimestamp[10];
  navb_gptp_message_sourcePortIdentity_t requestingPortIdentity;
} navb_ptp_path_delay_response_follow_up_message_t;
```

Auf jedes einzelne Element innerhalb der Datenstrukturen soll hier nicht eingegangen werden. Dazu dient der entsprechende IEEE Standard. Der grundsätzliche Ablauf der Funktion navb_gptp_process_incoming_gptp_frame() sieht aber wie folgt aus:

1. Öffnen des Buffers ist erfolgreich, da ein Path Delay Request Frame vorliegt. Die aktuelle Systemzeit wird als Empfangszeit festgehalten.

2. Da sich die drei Frametypen nicht in Ihrer Länge und nicht essentiell vom Aufbau her unterscheiden, wird der Path Delay Request Frame Frame in Path Delay Response Frame kopiert und der Bufferpool mit dem Path Delay Request Frame daraufhin geschlossen.

3. sourcePortIdentity, requestingPortIdentity und weitere Werte werden angepasst. Die festgehaltene Empfangszeit wird in requestReceiptTimestamp geschrieben.

4. Ein Frame zum Versenden wird vom CBS angefordert und der fertig konfigurierte Path Delay Response Frame wird hinein kopiert.

5. Die aktuelle Systemzeit wird als Absendezeit festgehalten und der Frame abgeschickt.

6. Der Path Delay Response Frame wird in den Path Delay Response Follow Up Frame kopiert und dort die Absendezeit des vorherigen Frames in responseOrigin-Timestamp geschrieben.

7. Ein weiterer Frame wird vom CBS angefordert und der Path Delay Response Follow Up Frame hinein kopiert.

8. Der Path Delay Response Follow Up Frame wird abgeschickt.

Unter der Annahme, das ein einfaches Beantworten der Path Delay Request Frames mit den korrekten Werten zu einer gPTP-Capability gegenüber des AVB-Switches folgen würde, wurde in der seriellen Konsole des AVB-Switches der Wert „gPTP Capable“ beobachtet. Aber auch nach wiederholtem Beantworten der Path Delay Request Frames hat sich dieser Wert nnicht geändert und stand dauerhaft auf „No“. Ein Erhöhen des maximal zulässigen Tresholds für die Antwortzeit innerhalb der AVB-Switch Konfiguration brachte ebenfalls keine Abhilfe.

5.5.1 gPTP-Capable mit Hilfe des XMOS Audio Boards

Um die Ursache des Problems des nicht vorhandenen gPTP-Capables zu finden ist zuerst die Annahme geprüft worden, ob ein einfaches Beantworten von Path Delay Request Frames ausreichend ist. Dazu wurden die vorhandene Referenzplattform untersucht.

Der AVB-Switch als auch das XMOS Audio Boards sind laut Spezifikationen vollständig gPTP-kompatibel. Wird der Netzwerkverkehr zwischen beiden Geräten mit Wireshark beobachtet, so ist feststellbar, dass eine Vielzahl unterschiedlicher gPTP-Pakete untereinander verschickt werden. Zusätzlich zu den oben genannten Frame-Typen sind ebenfalls Sync- und Follow-Up Frames zu sehen. Der ablaufende Master-Clock Auswahlalgorithmus als auch die Uhresynchronisation sind ebenfalls sichtbar. Um die Kommunikation der beiden Geräte auf die drei oben genannten Frames einzugrenzen und die These zu belegen das ein alleiniges Beantworteten der Path Delay Request Frames ausreichend ist, um vom AVB-Switch als gPTP-Capable erkannt zu werden, mussten alle weiteren gPTP-Frametypen aus dem Netzwerkverkehr herausgefiltert werden.

Dafür ist folgender Aufbau gemacht worden:

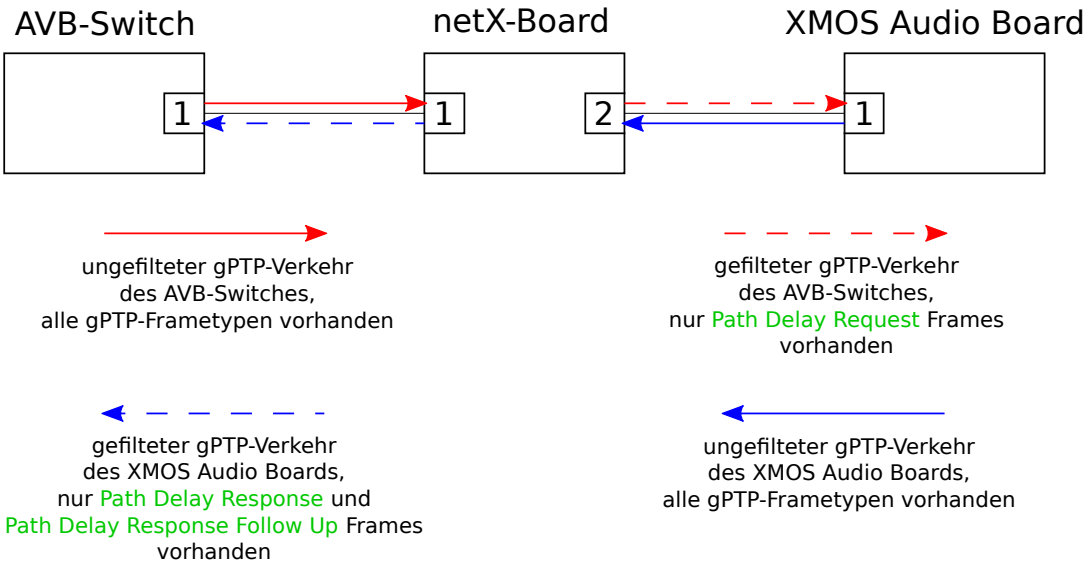

Abbildung 5.6: Ein NetX-Board im Einsatz als Ethernet-Filter

Für das verwendete NetX-Board ist im Gegensatz zur Default-Konfiguration die zweite Ethernet-Schnittstelle ebenfalls aktiv. Wie in Abbildung 5.6 erkennbar ist der AVB-Switch an der ersten Ethernet-Schnittstelle des NetX-Boards angeschlossen und das XMOS Audio Board an der zweiten. Der sich in der Entwicklung befindliche AVB-Stack wurde so modifiziert, dass eingehende Frames anstatt gespeichert und verarbeitet, nur auf ihren Ethertype und ggf. einzelne Bytes im Datenfeld untersucht und ggf. weitergesendet werden.

Treffen nun gPTP-Pakete des Switches an der ersten Ethernetschnittstelle des NetX-Boards ein, so werden diese nur auf der zweiten Ethernet-Schnittstelle zum XMOS Audio Board weitergeschickt, falls es sich um Path Relay Request Frames handelt. Andere Frames werden verworfen. Analog dazu passiert dies an der zweiten Ethernet-Schnittstelle eingehenden Frames des XMOS Audio Boards, sollte es sich nicht um Path Delay Response or Path Delay Response Follow Up Frames handeln.

Der Master-Clock Auswahlalgorithmus als auch die Uhrensynchronisation sind auf diese

Art zwar ausser Funktion, jedoch konnte in der seriellen Konsole des AVB-Switches beobachtet werden, dass das XMOS Audio Board dennoch als gPTP-Capable erkannt wird und die Annahme sich deswegen bestätigt hat.

Anpassen der versendeten Zeiten

Nach Prüfen des vorhandenen Codes und mit Abgleich zum Netzwerkverkeh,r der in Wireshark zwischen dem AVB-Switch und dem XMOS Audio Board beobachtet werden konnte, wurde die Programmlogik für korrekt befunden und der Fehler im Inhalt der versendeten Zeiten vermutet. Höchstwahrscheinlich ist, dass der Fehler im Umrechnen der Systemzeit auf dem NetX-Board zu dem in den gPTP-Frames verwendeten Fomat liegt.

Auf dem XMOS Audio Board mit vorgeschaltetem Filter findet keine Zeitsynchronisation statt. Die versendeten Zeiten müssen also nicht irgendeiner Relation zur internen Zeit des AVB-Switches stehen, da dieser nur Deltas aus den empfangenen Zeiten errechnet, um die Dauer der Übertragung zu bestimmen.

Daraus entstand die Annahme, dass es möglich ist, Zeiten des XMOS Audio Boards als eigene Zeiten des NetX-Boards auszugeben. Zuerst wurden 26 Pärchen von Zeiten aus den Antwort Path Delay Response und Path Delay Response Follow Up Frames des XMOS Audio Boards aufgezeichnet und hardcoded in das gPTP-Modul übernommen. Anstatt die eigene Systemzeit zu verschicken, wird jetzt bei jedem empfangen eines Path Delay Request Frames jeweils ein aufgezeichneter Wert zurückgeschickt. Bei dem 27. Frame findet ein Wrap-Around statt und es wird wieder der erste Wert geschickt.

Eine Kontrolle in der seriellen Konsole des AVB-Switches führte zu einem positiven Ergebnis. Das angeschlossene NetX-Board mit dem laufenden gPTP-Modul wird jetzt als gPTP-Capable erkannt. Die Path Delay Request werden sekündlich verschickt. Im Schnitt dauert es zwei Perioden bis das NetX-Board als gPTP-Capable erkannt ist. Durch weiteres experimentieren konnte festgestellt werden, dass insgesamt drei verschiedene Pärchen von Antwortzeiten ausreichend sind. Dies verringert den Speicherverbrauch der hardgecodeten Werte von 520 Bytes auf 60 Bytes.

5.6 Implementierung des Stream Reservation Protocols

Die Implementierung des Stream Reservation Protocols besteht wie in Abbildung 2.3 aus dem Grundlagenteil dargstellt aus mehreren logischen Komponeten.

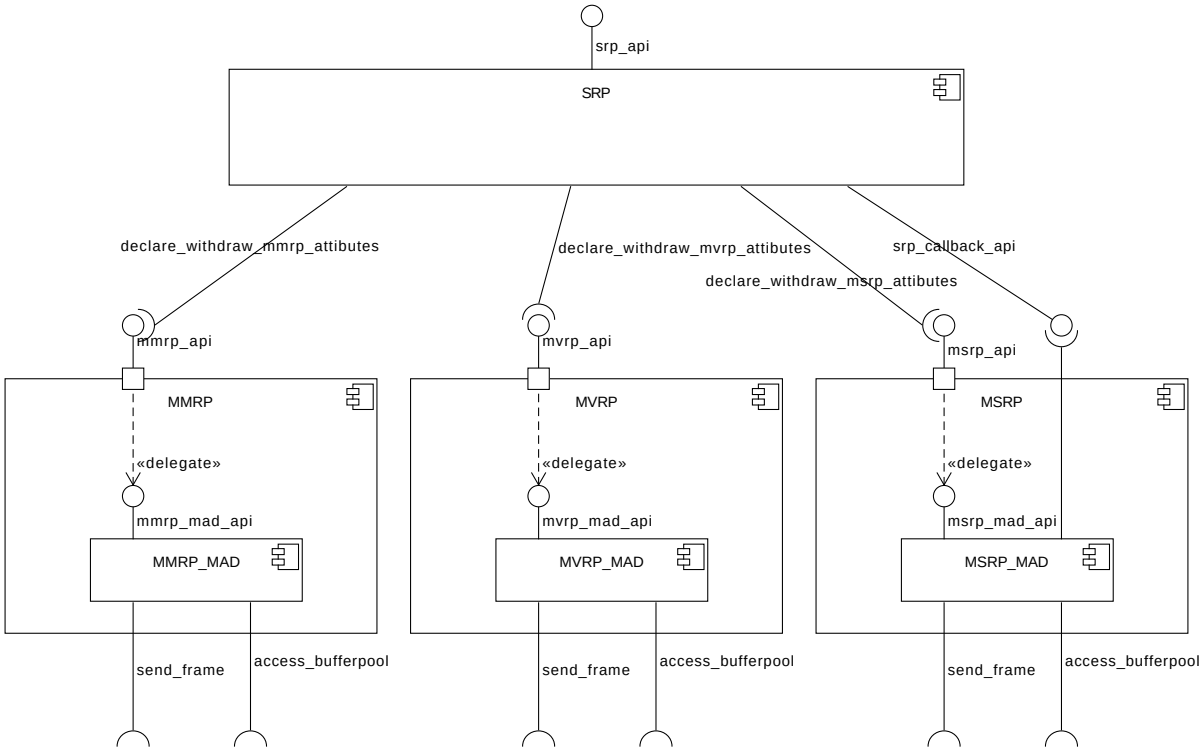

Abbildung 5.7: logische Kompontenten der SRP Implementierung

Abbildung 5.7 zeigt die logischen Komponenten der SRP-Implementation. Für jede der drei MRP-Anwendungen existiert eine Komponente mit eigener API. Wie zu sehen ist, ist entgegen der ursprünglichen Konzeption ein MMRP-Modul entstanden, obwohl per Default nicht benötigt, da es nur bei aktivierten „Talker Pruning“ eine Funktion hat. Da aber bei der Implementierung darauf geachtet wurde, dass möglichst viel Code generisch und daher nicht MRP-anwendungsspezifisch ist, ist eine Implementierung des MMRP-Moduls mit wenig Aufwand verbunden gewesen und wurde daher trotzdem umgesetzt. Eine zukünftige Möglichkeit Talker Pruning zu nutzen ist daher vorhanden, aber im Laufe dieser Arbeit nicht ausreichend getestet worden.

Die API der einzelnen MRP-Anwendungsmodule ist gemäss der IEEE-Standards umgesetzt worden. MMRP ermöglicht es Attribute für Mac-Adressen und sogenannte „Service Requirements“ zu deklarieren und ggf. die Deklarieren wieder zurückzuziehen. Bei MVRP sind dies VLAN IDs und bei MSRP Domain, Talker und Listener Attribute.

Auf diese drei Module setzt das SRP-Modul auf. Dieses Modul bietet eine einfache API zum Anmelden von Talkern und Listenern.

Wie ebenfalls in Abbildung 5.7 zu sehen ist, besitzt jedes MRP-Modul ein zusätzliches MAD-Modul, das ebenfalls gröstenteils generisch ist.

5.6.1 MSRP als Beispiel einer Implementierung einer konkreten MRP-Applikation

Im Laufe dieses Kapitels soll MSRP als Beispiel einer Implementierung einer konkreten MRP-Anwendung dienen. Der Grund dafür ist, dass die Mechanismen mit Hinsicht auf die Verarbeitung von Attributinformation innerhalb der drei MRP-Protokolle zwar identisch sind, MSRP aber zusätzlich Besonderheiten beim Entpacken und Packen von PDUs, dem Nachrichtenformat für MRP innerhalb von Frames, bietet, die nicht in MMRP oder MVRP enthalten sind.

	generisches Modul	anwendungsspezifisches Modul
allgemeine Funktionen	navb_mrp	navb_msrp
entpacken/packen von PDUs	navb_mrp_pdu	navb_msrp_pdu
MAD	navb_mrp_mad	navb_msrp_mad

Tabelle 5.1: Modulübersicht innerhalb des MSRP-Moduls

Während Abbildung 5.7 eher einen Blick aus logischer Sicht auf die Komponenten des MSRP bietet, listet die Tabelle 5.1 alle in die Implementation involvierten Module auf. Es existieren generische Module, die nur Bibliotheksfunktionen und notwendige Datenstrukturen bereitstellen und von allen drei MRP-Implementationen benutzt werden. Ausserdem existieren zusätzlich anwendungsspezifische Module, die jeweils pro MRP-Implementation anwendungsspezifische Details zu den generischen Modulen hinzufügen. Der Grossteil für die Funktion einer MRP-Implementation notwendigen Codes ist in den generischen Modulen enthalten.

Die Aufgaben der einzelnen Module aus Tabelle 5.1:

- **navb_mrp:** bietet Basisfunktionen und Datenstrukturen für das Anlegen und Verwalten von Attributen innerhalb der MRP-Anwendungskomponente
- **navb_msrp:** stellt die grundlegende API bereit, die notwendig ist um in MSRP Attribute zu deklarieren oder Deklarierungen zurückzunehmen, diese Funktionen sind z.B. navb_msrp_register_stream() und navb_msrp_deregister_stream(), stellt ebenfalls Speicher für MSRP-Attribute bereit
- **navb_mrp_pdu:** stellt Funktionen zum Packen und Entpacken von MRP Nachrichten und die in diesen Nachrichten enthaltenen Events bereit
- **navb_msrp_pdu:** bietet eine Funktion um die zu dieser MRP-Implementation gehörenden anwendungsspezifischen Attribute aus Nachrichten entpacken zu können
- **navb_mrp_mad:** stellt grundlegende Datenstrukturen und Funktionen bereit, um die in MAD-Komponente enthaltenen State Machines und zugehörige Timer benutzen zu können, bietet die in IEEE 802.1Qat beschriebene und in Abbildung 4.4 dargestellte API
- **navb_msrp_mad:** bietet eine Initialisierungsfunktion um die in diesem Modul gespeicherten und von den Bibliotheksfunktionen aus navb_mrp_mad verwalteten Datenstrukturen zu initialiseren, besitzt Wrapper-Funktionen die, die in Abbildung 4.4 gezeigten Funktionen auf anwendungsspezifische Attribute dieser MRP-Implementierung abzubilden

Datenhaltung innerhalb der MSRP-Anwendung

Wie gefordert sollen die Attributinformationen in der MRP-Anwendungskomponente, in diesem Fall MSRP, gespeichert werden. Die MAD-Komponente soll der Anwendungskomponente die Request/Indication-API anbieten und intern die State Machines verwalten aber selber keine, mit Ausnahme der Deklarations- und Registrierungsinformationen, Attributinformationen halten. Jedoch ist es notwendig, dass die MAD-Komponente zum Zwecke des Abschickens von Frames als auch bei Eingang von Frames Zugriff zu diesen Attributinformationen hat.

Die folgenden Datenstrukturen sollen die Verteilung der verschiedenen Attributinformation innerhalb von MSRP verdeutlichen. Zum Zwecke der Vereinfachung sind diese Datenstrukturen zum Teil im Umfang reduziert dargestellt.

Listing 5.12: Datenstruktur für ein Talker Advertise-Attribut

```
typedef struct {
  navb_msrp_pdu_streamId_t streamId;
  ...  // and more
} navb_msrp_pdu_talker_advertise_t;
```

Ein navb_msrp_pdu_talker_advertise_t Datenyp, der ein Talker Advertise-Attribut darstellt, enthält alle Information die notwendig sin um einen Talker bzw. dessen angebotenen Stream zu beschreiben. Diese Datenstruktur stammt aus dem anwendungsspezifischen MRP PDU-Modul (navb_msrp_pdu) und wird daher ebenfalls innerhalb von PDUs versendet und empfangen. Ein Array dieser Datenstruktur, mit der Länge der maximal möglichen Talker Advertise-Attribute befindet sich als Variable innerhalb des MSRP-Moduls. Die restlichen in MSRP existierenden Attribute wie Domain etc. haben eigene Datenstrukturen.

Listing 5.13: Datenstruktur der die Eigenschaften eines Attributs beschreibt

```
typedef struct {
  bool* inUse;
  unsigned char* value;
  unsigned char type;
  unsigned int length;
  unsigned int offset;
  unsigned char* additionalValue;
  unsigned int additionalValueLength;
} navb_mrp_attribute_descriptor_t;
```

Die navb_mrp_attribute_descriptor_t Datenstruktur stammt aus dem generischen MRP-Anwedungskomponentenmodul (navb_mrp) und hat die Aufgabe Eigenschaften eines Attributtyps zu beschreiben. Diese Informationen sind notwendig, da diese wie z.B. type und length innerhalb einer PDU verschickt werden. value ist ein unsigned char-Zeiger der auf ein Array von angelegten Variablen eines bestimmten Attributyps zeigt, in diesem Falle der oben genannte Array aus Talker Advertise-Attributen. inUse wiederum ist ein

Zeiger, der auf ein Array mit der Belegung des Attributvariablen-Arrays zeigt und dessen Belegung angibt. Jeder Attributtyp hat ein Offset. Dieser wird in der Variable offset gespeichert. Ein Array der navb_mrp_attribute_descriptor_t Datenstruktur wird in dem Modul navb_msrp gespeichert und enthält einen Eintrag pro Attributtyp.

Listing 5.14: Datenstruktur die die Eigenschaften eines Attributs innerhalb des MAD-Moduls beschreibt

```
typedef struct {
  bool* inUse;
  unsigned char* value;
  unsigned char* type;
  unsigned int* length;
  unsigned int* offset;
  unsigned char* additionalValue;
  unsigned int* additionalValueLength;
  navb_mrp_mad_applicant_state_t applicantStateMachine;
  navb_mrp_mad_registrar_state_t registrarStateMachine;
  navb_timer_t leaveTimer;
  nav_mrp_leave_timer_param_t leaveTimerParam;
} navb_mrp_mad_attribute_t;
```

navb_mrp_mad_attribute_t ist eine Datenstruktur der von dem generischen MAD-Modul (navb_mrp_mad) zur Verfügung gestellt wird. Diese Datenstruktur wird für jeden Attributtyp der drei MRP-Implementationen benutzt, um diese in ihren Eigenschaften zu beschreiben. Ein Array diesen Typs, mit der Länge der maximalen Anzahl von Attributtypen pro MRP-Implementation, existiert innerhalb des anwendungsspezifischen MAD-Moduls, in diesem Falle navb_msrp_mad. Der Inhalt der Zeiger inUse und value ist identisch mit Zeigern einer Variablen vom Typ navb_mrp_attribute_descriptor_t. Auf diese Weise bekommt die MAD-Komponente Zugriff auf die Attributinformation die in der Anwendungskomponete gespeichert sind. Die Zeiger Value bis additionalValue-Length zeigen auf die entsprechenden Felder dieser Variablen. Wie ebenfalls zu sehen ist, befinden sich auch die in Unterabschnitt 4.2.2 beschriebenen Applicant und Registrar State Machines, ebenso der Leave Timer innerhalb dieser Datenstruktur.

Listing 5.15: Diese Datenstruktur hält alle Informationen die zum Betrieb des MAD-Moduls notwendig sind

```
typedef struct {
```

```
  unsigned int attributesMax;
  navb_mrp_mad_attribute_t* attributes;
  navb_mrp_mad_leave_all_state_t* leaveAllStateMachine;
  navb_mrp_mad_periodic_transmission_state_t*
    periodicTransmissionStateMachine;
  navb_timer_t* joinTimer;
  navb_timer_t* leaveAllTimer;
  navb_timer_t* periodicTimer;
  ... // and more
} navb_mrp_mad_t;
```

Diese Datenstruktur stammt ebenfalls aus dem generischen MAD-Modul (navb_mrp_mad). Eine Variable dieser Datenstruktur befindet sich innerhalb des anwendungsspezifischen MAD-Moduls (navb_msrp_mad). In dieser Datenstruktur sind alle zum Betrieb notwendigen Information. Es existiert ein Zeiger auf Array des vorher besprochenen navb_mrp_mad_attribute_t Datenstruktur. Mit der Länge der maximalen Anzahl von Attributen, die für diese MRP-Implementation verfügbar sind. Die restlichen Elemente der Datenstruktur sind Zeiger auf Variablen, die sich ebenfalls in dem anwendungsspezifischen MAD-Modul befinden. Diese Zeiger zeigen auf jeweils nur ein Element des jeweiligen Typs, da diese nur einmal pro MRP-Applicant und nicht einmal pro Attribut dieses MRP-Applicants existieren.

Abbildung 5.8 verdeutlicht wie die MAD-Komponente auf die Daten der Anwendungskomponente zugreift ohne die Daten selber halten zu müssen.

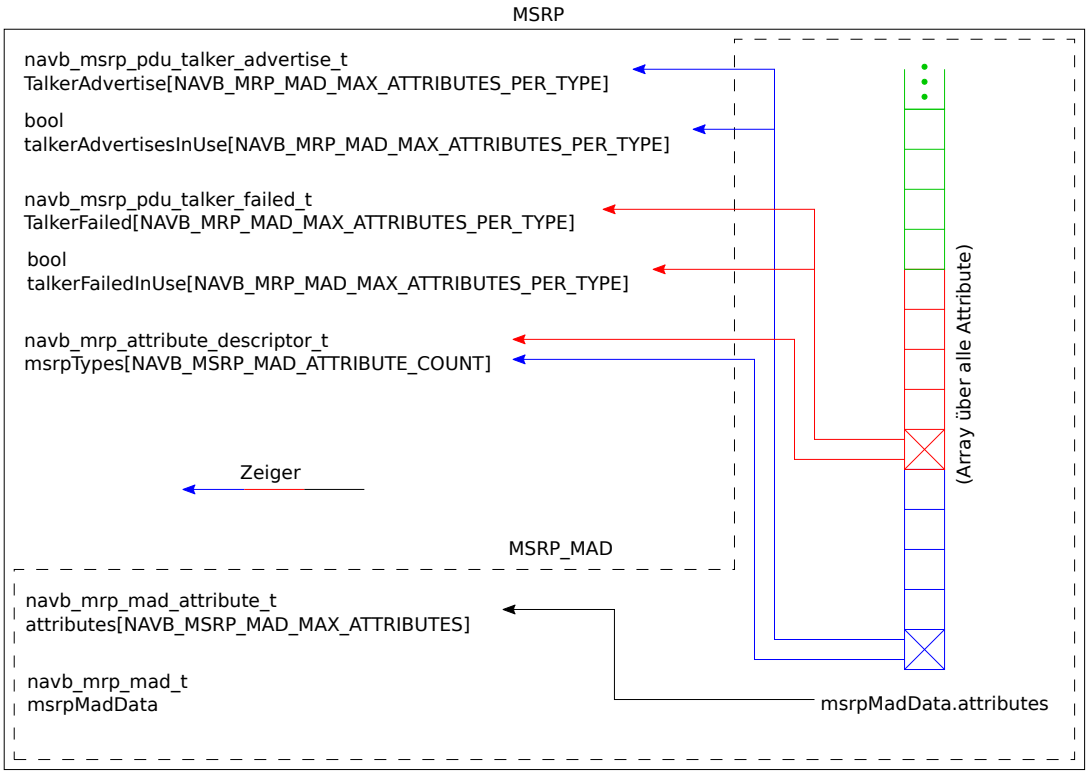

Abbildung 5.8: Zugriff auf Attributdaten innerhalb der Anwendungskomponente

Das Array attributes enthält Einträge über alle in dieser MRP-Implementierung möglichen Attribute. Die Anzahl der möglichen Attribute ist im voraus über Defines begrenzt und errechnet sich durch Anzahl der Attribute pro Attributtyp multipliziert mit der Anzahl der unterschiedlichen Attributtypen. Ebenso ist vorgegeben an welcher Stelle im Array welche Art von Attributen liegt. Der Zugriff auf einen bestimmten Typ erfolgt über den vorher geannten Offset. Die hier blau gefärbten Zellen sind Attribute des Typs Talker Advertise, die roten sind Attribute des Typs Talker Failed. Grün soll den Rest der in MSRP vorhandenen Attributtypen darstellen. Die mit „X“ markierten Zellen enthalten ein vorhandenes Attribut. Bei ihnen ist der Wert *msrpMadData.attributes->inUse == TRUE. Auf die gleiche Art kann die MAD-Komponente auf den Attributwert zugeifen. Dieser ist in msrpMadData.attributes->value zu finden. Die MAD-Komponte interpretiert keine Attributwerte sondern empfängt und verschickt diese lediglich. Mit einem generischen char Zeiger und der Angabe der Länge des Attributs in *msrpMadData.attributes->length ist dies für alle MRP-Implementierungen und Attributtypen möglich.

Modellierung der State Machines innerhalb des MRP MAD-Moduls

Die vier von MRP geforderten State Machines sind unterteilt in attributspezifische, d.h. es ist jeweils eine State Machine für genau ein Attribut vorhanden und anwendungsspezifische State Machines, bei der genau eine State Machine pro MRP-Applicant vorhanden ist, unterteilt. Die Applicant- und Registrar- State Machine sind attributspezifisch, während die Join- und LeaveAll State Machine anwendungsspezifisch sind.

Jede dieser State Machines besteht aus einer Datenstruktur deren angelegte Variable den derzeitigen Zustand hält und einer für jeden State Machine-Type individuelle Trigger-Funktion. Die Trigger-Funktion ist mittels eines geschachtelten Switch-Konstrukts modelliert die den derzeitigen Zustand und das übergebene Event in eine Action und ggf. Zustandsänderung übersetzt. Die Trigger-Funktion bekommt ebenfalls die Variable, im Falle von MSRP, msrpMadData übergeben. Mit dieser Datenstruktur erhält die Trigger-Funktion Zugriff auf alle in MSRP vorhandenen Daten inklusive Attributwerte und der anderen State Machines die bei Bedarf ebenfalls getriggert werden können. Im Falle der Applicant und Resgistar State Machine wird ebenfalls der Index der zu triggernden State Machine übergeben. Dieser Index zeigt auf einen Eintrag im attributes Array.

Desweiteren befinden sich Funktionen die bestimmte Aktionen für State Machines ausführen, z.B. senden von Frames, in dem Modul. Callback-Funktionen für die, in MRP vorhandenen Timer, befinden sich ebenfalls in dem Modul. Diese Callback-Funktionen werden beim Konfigurieren einers Timers übergeben und gestartet, wenn ein Timer ausgelaufen ist. Diese Callback Funktionen rufen dui Trigger-Funktionen der verschiedenen State Machines auf senden Events an diese.

MSRP-Anwendungsbeispiel: Initialisierung des MSRP-Moduls

Wird die Initialisierungsfunktion des MSRP-Moduls aufgerufen, so werden die inUse Bool Arrays, von denen ein Array für jeden Attributtyp existiert mit dem Wert FALSE initialisiert. Die Attributspezifischen Timer und State Machines werden in einen Default-Zustand versetzt. Als nächstes wird der Array msrpTypes, der die Beschreibungen für jeden Attributtyp enthält, mit den für den jeweiligen Attributtyp passenden Werten gefüllt. Als letztes wird die Initialisierungsfunktion des MSRP MAD-Moduls aufgerufen und dieser der Array mit dem Attributbeschreibungen übergeben. Zusätzliche Parameter sind Zeiger auf Callback-Funktionen für ein Zuordnen von Attributmerkmalen zu einem

bestimmten Atributtyp. Eine Funktion zum Hinzufügen eines neues Attributes in die Anwendungskomponente aus der MAD-Komponente heraus und jeweils eine Callback-Funktion für ein Leave- und Join Indication. Diese vier Callback-Funktionen sind in der Anwendunskomponente implementiert und werden in Abschnitt 5.6.1 näher erläutert.

Innerhalb der Initialisierungsfunktion des MSRP MAD-Moduls wird der attributes Array mit den Werten des übergebenen Arrays konfiguriert. Ist dies beendet, so sind die Attributeigenschaften jedes Element so konfiguriert, dass sie genau einen Attributtyp repräsentieren und die Zeiger mit den Werten value und inUse auf jeweils ein Attribut im entsprechenden Array in der MSRP-Anwendungskomponente zeigen.

Danach werden die anwendungsspezifischen Timer konfiguriert, dessen Zeiten zuvor mittels eines Zufallszahlengenerators ausgerechnet worden sind und auf die bei Bedarf zurückgegriffen werden kann. Zeiger für Callback-Funktionen werden gespeichert und die Startzustände der anwendungsspezifischen LeaveAll und Periodic State Machines gesetzt. Mit dem Senden eines Begin Events an beide State Machines ist das MSRP MAD-Modul und damit auch das MSRP-Modul „lebendig“ und damit einsatzbereit.

MSRP-Anwendungsbeispiel: Deklarieren eines neuen Talker-Attributs

Soll in neuer Stream innerhalb des Netzwerks bekanntgegeben werden, so muss die Funktion navb_msrp_register_stream() des MSRP-Moduls, mit den entsprechenden Stream-Parametern aufgerufen werden. Auf diese Weise wird ein neues Talker Advertise-Attribut deklariert und Frames mit dessen Streaminformation verschickt.

Dieser Vorgang ist in zwei Teilabschnitte gegliedert die in Abbildung 5.9 und Abbildung 5.10 erläutert werden.

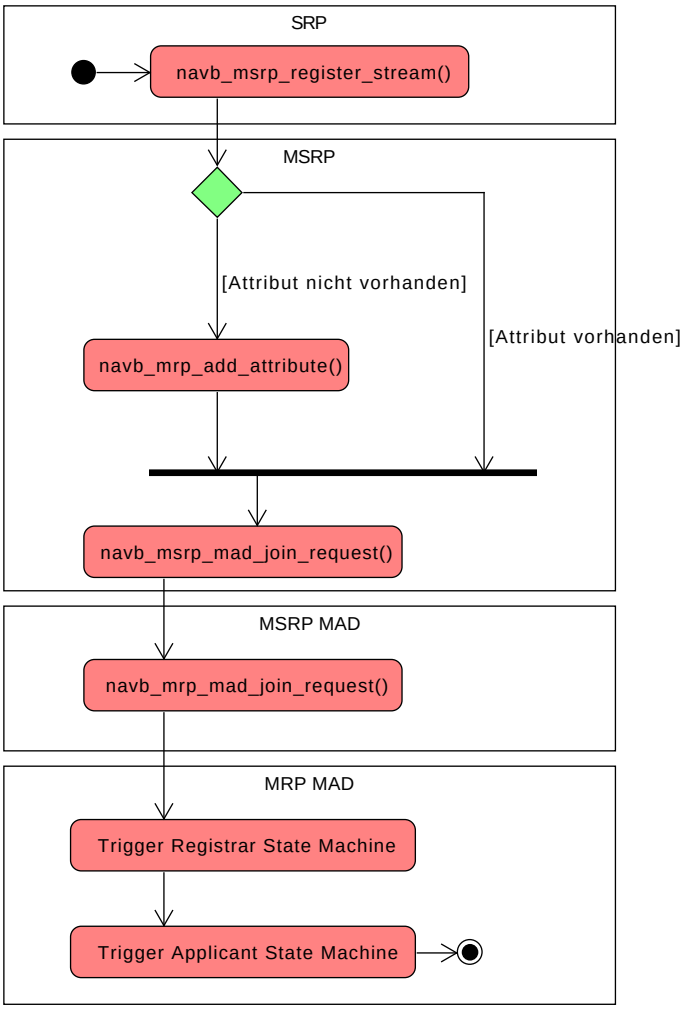

Abbildung 5.9: Deklarieren eines Streams bzw. Talker Advertise-Attributs

Abbildung 5.9 zeigt wie aus SRP heraus die MSRP-Funktion navb_msrp_register_stream() aufgerufen wird. In diesem Funktionsaufruf sind alle Informationen um ein Talker Advertise-Attribut und damit auch einen Stream zu beschreiben enthalten. Zunächst wird überprüft, ob dieses Attribut bereits bekannt ist. Ist dies nicht der Fall, so wird mit der Funktion navb_mrp_add_attribute() aus dem generischen MRP-Modul aufgerufen. Das neue Attritbut wird in den Talker Advertise-Attribute Array innerhalb des MSRP-Moduls gespeichert und die Variable inUse dort auf TRUE gesetzt. Der Index dieses Elements ist ein Return-Wert.

Danach wird die Funktion navb_msrp_mad_join_request() aufgerufen. Diese Funktion stammt aus dem MAD-Modul von MSRP. Es bildet eine Wrapper-Funktion für die Funktion navb_mrp_mad_join_request(). Übergabeparameter sind der Index des gespeicherten Talker Advertise-Attributs zusammen mit einem attributtypspezifischen Offset. Diese beiden Parameter sind notwendig, damit das generische MAD-Modul das entsprechende Attribut in der Liste aller Attribute wiederfinden kann.

Letztendlich werden die Applicant- und Registrar State Machine, mit einem NEW oder JOIN Event getriggert. Welches von beidem der Fall ist ist durch die MRP-Implementation vorgegeben. Bei MSRP ist es das NEW Event.

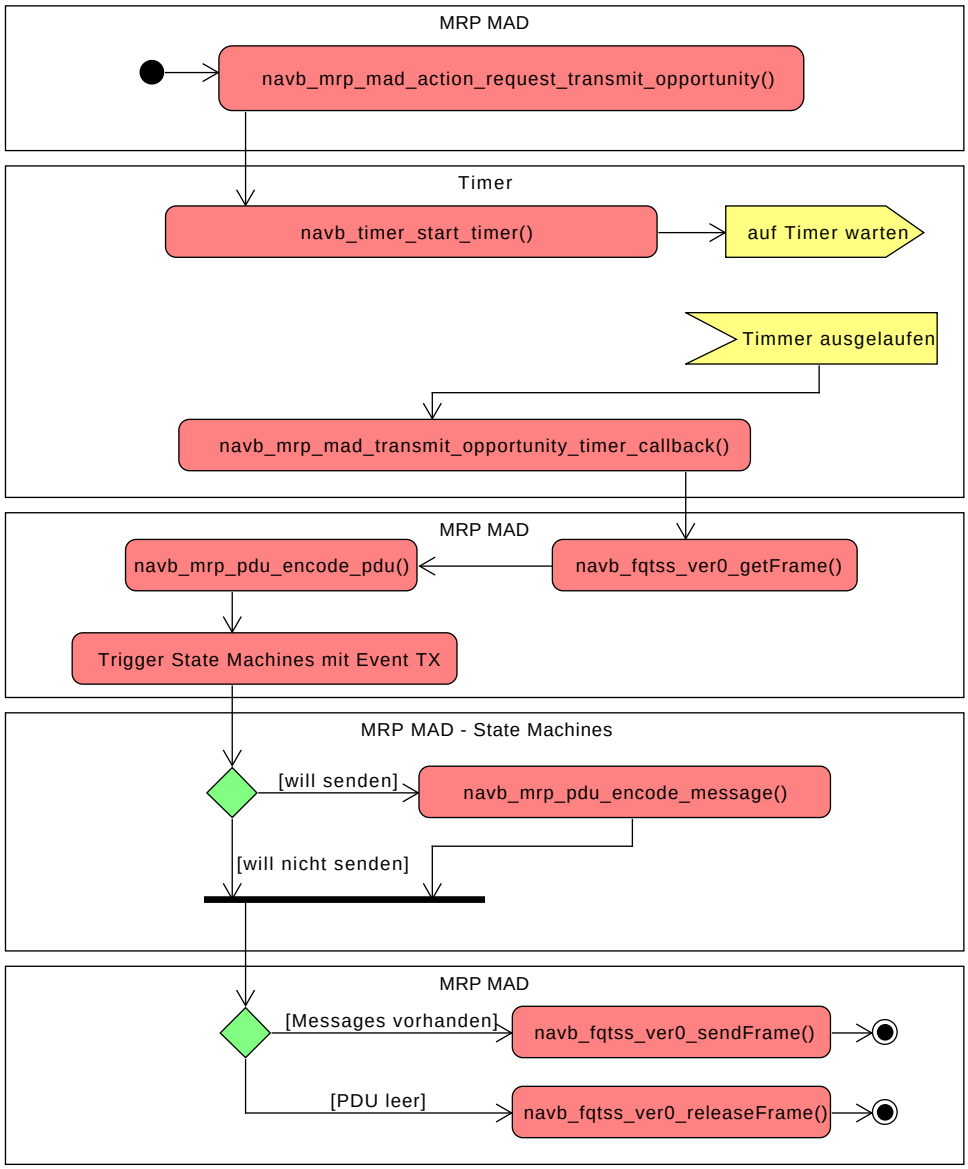

Abbildung 5.10: Senden eines MRP Frames

Das Deklarieren eines Attributs und das Senden eines Frames mit Informationen zu diesen Attribut sind zwei unterschiedliche Vorgänge, da sie zeitlich nicht direkt aufeinander folgen und damit voneinander entkoppelt sind. Abbildung 5.10 beschreibt letzteren Vorgang. Will eine State Machine einen Frame versenden, so muss sie erst eine sogenannte „Transmission Oppotunity“ anfordern, d.h. es wird der Wunsch geäußert einen Frame zu

verschicken, aber es muss vorher gewartet werden bis dies soweit ist. Der Grund dafür besteht darin, dass anderen State Machines in der Wartezeit die Möglichkeit gegeben wird ebenfalls Attributinformationen zu senden und so ein Frame Informationen zu mehr als einem Attribut enthalten kann und somit Bandbreite gespart wird.

Wird die Transmission Opportunity angefordert so wird der Join Timer des MRP-Applicants gestartet. Läuft der Timer aus, so startet er seine Callback-Funktion. Die Callback-Funktion fordert einen Frame vom CBS-Modul an. Darin wird eine Datenstruktur, die PDU, die das Format für MRP Frames erstellt.

Als nächstes werden alle aktiven State Machines mit dem Event TX getriggert. Dieses Event sagt aus, das jetzt die Möglichkeit besteht Attributinformationen zu senden. Attributinformationen werden als Messages kodiert und State Machines die im entprechenden Zustand sind fügen Messages in die PDU ein.

Sind Messages in die PDU eingefügt worden, so wird die Sendefunktion an des CBS-Moduls aufgerufen. Falls dies nicht der Fall ist, wird der angeforderte Frame wieder freigegeben.

MSRP-Anwendungsbeispiel: Registrieren eines neuen Listener Attributs

Das folgende Beispiel soll zeigen wie fremde Attribute registriert und auf MRP aufbauende Software-Layer darüber benachrichtigt werden. Wegen der besseren Übersicht ist die folgende Grafik in zwei logisch aufeinander folgende Abschnitte unterteilt.

Für das, im vorherigen Abschnitt deklarierte Talker-Attribut, soll nun ein Listener-Attribut registriert werden.

Der eingehende Frame ist bereits durch die Empfangs-ISR in den korrekten Bufferpool einsortiert worden und wartet nun auf die Bearbeitung.

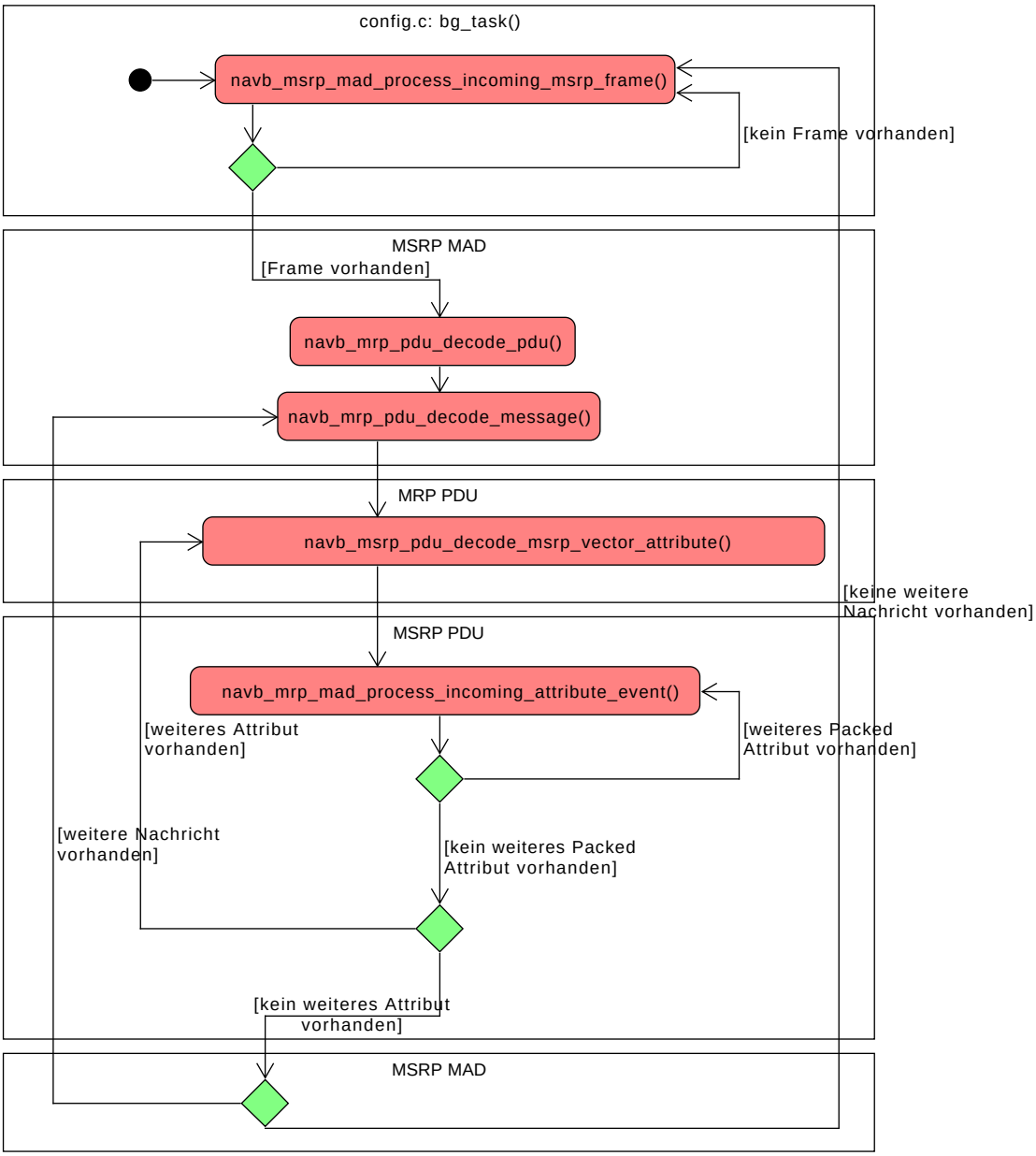

Abbildung 5.11: Registrieren eines Stream-Konsumenten bzw. Listener-Attributs

Wie in Abbildung 5.11 zu sehen, wird in der Background Task-Funktion innerhalb des Config-Moduls zyklisch die Funktion navb_msrp_mad_process_incoming_msrp_frame() aufgerufen. Die Funktion prüft bei einem Aufruf, ob ein Frame im zugehörigen MSRP-Buffer vorliegt. Ist dies nicht der Fall terminiert die Funktion sofort und wird beim

nächsten Aufruf der Background-Task wieder aufgerufen.

Falls ein Frame vorliegt wird dieser geöffnet und über die im Moment noch ungeordnete Menge an Bytes innerhalb des Datenfeldes wird eine PDU-Datenstruktur gelegt. Diese Struktur hat einen Zeiger auf eine nächste Nachricht. Liefert der Zeiger den Wert einer EndMark ist keine Nachricht vorhanden, ansonsten wird versucht diese Nachricht zu parsen. Dieser Vorgang wird wiederholt bis die EndMark erreicht wurde.

Innerhalb einer Nachricht liegt eine Attributliste. Diese kann ein oder mehrere Attribute enthalten. Es wird versucht ein Attribut nach dem anderen zu parsen bis das Ende der Attributliste, ebenfalls gekennzeichnet durch eine EndMark, erreicht ist.

Jedes Attribut besteht aus dem Klartext aller seiner Attributwerte und einem Event. Der Wert des Events bildet den Zustand der State Machines im Absender des Frames ab. Das Event soll wiederum auf die eigenen, für dieses Attribut zuständigen, State Machines angewendet werden. Attribute und ihre Events können zusätzlich in einem gepackten Format gespeichert werden, dass in IEEE 802.1Qat unter dem Begriff „FirstValue+1“ zu finden ist. Mehr dazu in Abbildung 5.12. Falls dieser Fall vorliegt werden ebenfalls alle gepackten Attribute eines nach dem anderen geparst.

Ist ein Attribut mit seinem Event aus einem Frame extrahiert, so wird die Funktion navb_mrp_process_incoming_atribute_event() ausgeführt. Diese ist die zentrale generische Verarbeitungsfunktion für eingehende Attribute.

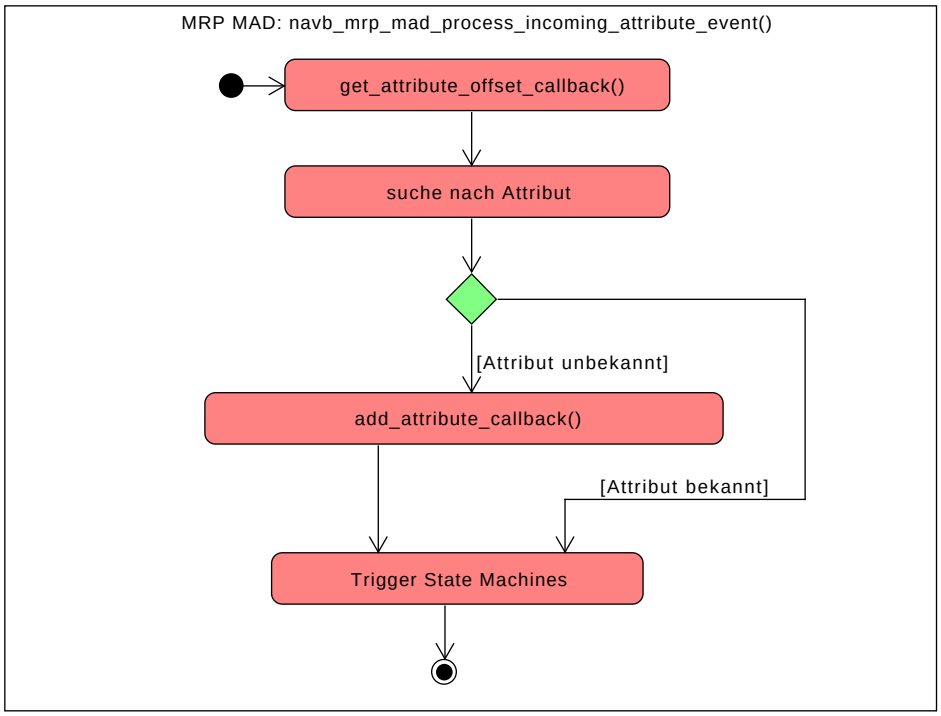

Abbildung 5.12: Verarbeiten von eingehenden Attributinformationen

Abbildung 5.12 zeigt den Ablauf innerhalb der generischen Verarbeitungsfunktion navb_mrp_mad_process_incoming_attribute_event(). Die Funktion ist Teil des MAD-Moduls und ist dahingehend auch nur für das Verwalten der Zustände von Attributen und das Verschicken bzw. Verarbeiten von eingehenden Frames, jedoch nicht der Interpretation von Attributinformation zuständig. Aus diesem Grunde ist in diesen Layer auch kein Wissen über die Struktur oder die Werte einzelner Attribute vorhanden. Die Funktion besitzt lediglich Informationen die direkt aus dem eingehenden Frame extrahiert worden. Diese Informationen enthalten den Typ des Attributs, den Wert als nicht interpretierbarer „Blob“ aus Bytes, dessen Länge und ein Event.

Um bestimmen zu können, ob das Attribut bereits lokal existiert. Ist es notwendig genauer zu bestimmen, um was für einen Attributtypen es sich handelt, da der im Frame gespeicherte Attributtyp, in dieser Stack-Implementation, nicht einzigartig ist. Mehr dazu in Abschnitt 5.6.1.

Es wird die Callback-Funktion get_attribute_offset_callback() aufgerufen. Diese Funk-

tion ist Teil des MSRP-Anwendungsmoduls und wurde der Intialisierungsfunktion des MAD-Moduls als Parameter übergeben. Ihr Zweck ist es, den Offset für einen spezifischen Attributtyp zu bestimmen.

Nun wird der attributes Array mit dem Offset als Startelement und der Länge definiert, in NAVB_MRP_MAD_MAX_ATTRIBUTES_PER_TYPE, bei dehnen der Wert inUse auf TRUE gesetzt ist, byteweise verglichen. Um die Geschwindigkeit nicht durch unnötiges Swappen von Bytes beim Vergleichen zu verlangsamen sind die Werte der Attribute lokal in Networkbyteworder gespeichert.

Findet sich kein Attribut so wird die Funktion add_attribut_callback() aufgerufen. Diese Funktion stammt ebenfalls aus dem MSRP-Anwendungsmodul und hat die Aufgabe dem MAD-Modul die Möglichkeit zu geben ein neues Attribut in einen der dafür zuständigen Arrays in dem MSRP-Anwendungsmodul schreiben zu können.

Ist dies getan bzw. das Attribut ist schon vorhanden, so wird das Event auf die State Machines, die für dieses Modul zuständig sind angewendet. Sollte das Attribut neu sein oder das Event kündigt ein Löschen einer lokalen Registrierung an, so wird eine Join bzw, Leave Indication ausgelöst. Dabei wird jeweils eine Callback-Funktion ausgeführt die eine Rückmeldung inklusive der genauen Attributinformation an das MSRP-Anwendungsmodul gibt.

Inkonsistenzen im PDU-Format für Listener-Attribute und die daraus folgende Änderung der Speicherung von Attributwerten innerhalb des MSRP-Moduls

Abbildung 5.13 zeigt das von MRP verwendete Format für die Protocol Data Unit, die den Aufbau von MRP Frames bestimmt. Innerhalb einer PDU werden Informationen zu Attributen und deren Status in der Form von Events verschickt. Das Format ist bis auf die hier beschriebene Ausnahme generisch und wird von allen drei MRP-Implementation benutzt. Eine PDU enthält ausschliesslich Attributinformation zu einer spezifischen MRP-Implementation. Sollen Attributinformation für zwei verschiedene MRP-Implementation, z.B. MVRP und MSRP versendet werden, so geschiet dies in zwei PDUs, in zwei unterschiedlichen Frames. Attribute innerhalb dieser Frames werden durch ihren Typ, ihre Länge und einen konkreten Wert unterschieden. Zu jedem Attribut gehört mindestens ein Event.

Das grundsätzliche Format einer PDU lautet wie folgt.

1. eine PDU enthält mindestens eine Nachricht und ist mit einer Endmarke abgeschlossen
2. in einer Nachricht befindet sich ein Attributtypenfeld und ein Attributlängenfeld, optional ist ein Feld für die Länge der Attributliste die sich am Ende der Nachricht befindet
3. in der Attributliste befindet sich mindestens ein Attribut, das als VectorAttribute bezeichnet ist, die Attributliste ist ebenfalls mit einer Endmarke abgeschlossen
4. ein VectorAttribute besitzt einen Header, dass das Attribut als konkreten Wert und als FirstValue bezeichnet, eine Liste von Events als Vector bezeichnet und als Besonderheit die nur für MSRP und Listener gilt einen extra Vector, der die Art des Listeners angibt
5. der Header des VectorAttributes enthält Informationen ob ein spezielles LeaveAll Event aufgetreten ist, und eine Angabe wieviel Werte die Vectoren enthalten

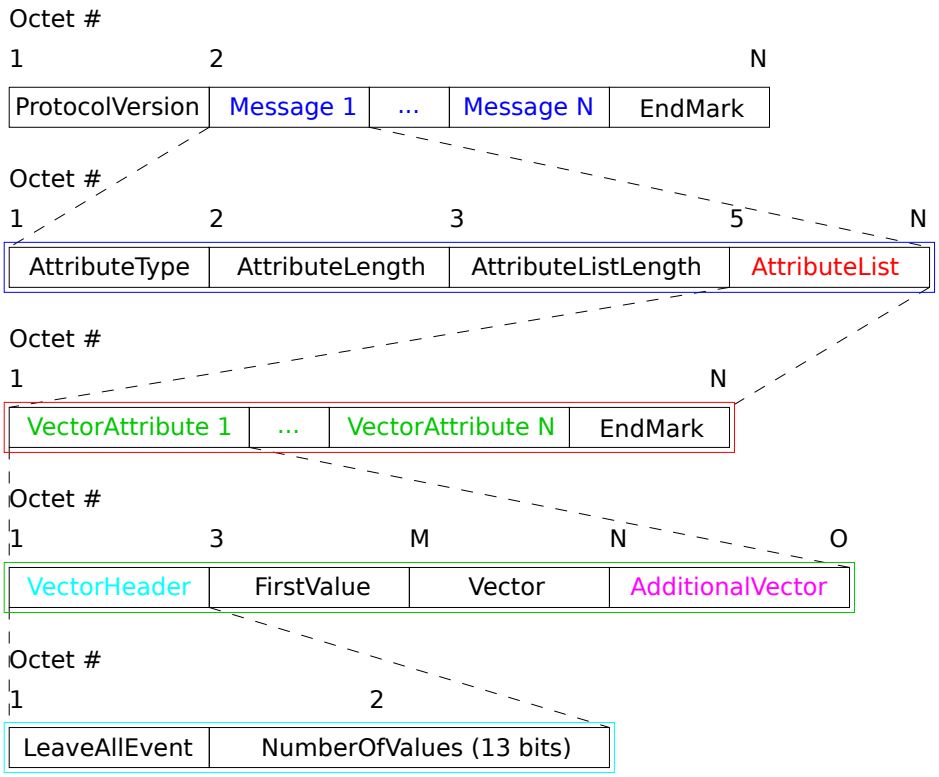

Abbildung 5.13: PDU als Format für MRP Frames (Darstellungsform nach [8])

Soll eine PDU mit Attributen gleichartigen Typs und mit fortlaufenden Werten, z.B. Talker-Attribute mit fortlaufenden Stream-IDs, verschickt werden, so können diese innerhalb eines einzigen VectorAttributes zusammen gepackt werden. Das Talker-Attribut mit der niedrigsten Stream-ID bildet den „FirstValue“. Die folgenden Stream-IDs bilden sich in dem FirstValue um eins inkrementiert wird. Wie oft dies geschehen muss um alle Attribute zu entpacken steht in „NumberOfValues“.

Wie in Abbildung 5.13 zu sehen exisitieren mehrere Möglichkeiten Attributinformationen innerhalb einer PDU zu verpacken. Unter der Vorraussetzung das Informationen zu mehr als einem Attribut vorliegen sind dies:

1. ein Attribut pro Nachricht
2. mehr als ein Attribut innerhalb der Attributliste
3. falls die Attributwerte fortlaufend sind, mehr als ein Attribut innerhalb eines VectorAttributes

Jeder dieser drei Möglichkeiten wird von dem implementierten AVB-Stack für das Entpacken und Verarbeiten unterstützt. Jedoch wird beim Versenden von PDUs immer nur ein Attribut pro Nachricht versendet. Zur effizienten Verwaltung der eingehenden und ausgehenden Attribute sind diese lokal in Network Byteorder gespeichert und in Arrays ihres jeweiligen Attributtyps gruppiert. Eine Identifikation der Attributtypen innerhalb der Anwendung erfolgt über den, in dem Feld „AttributeType“ enthaltenen Wert. Für MSRP sind dies die Attributtypen, Talker Advertise, Talker Failed, Listener und Domain.

Die verschiedenen Listener-Typen, Listern Ready, Listener Ready Failed und Listener Failed werden nicht über das Feld AttributeType unterschieden. Es befindet sich, nur bei Listener-Attributen vorhanden, ein zusätzlicher Vector mit dem konkreten Listener-Typ hinter dem Event Vector. Dieses zusätzliche Feld ist aber nicht in jeder PDU, die Listener-Attribute enthält, vorhanden. Wenn dieses Feld fehlt, ist es nicht möglich zuzuordnen, um welchen Typ von Listenern es sich handelt.

Der AVB-Switch sendet im Ruhezustand ein Listener-Attribut bei dem alle Attributwerte mit Nullen gefüllt sind. Dazu wird ein LeaveAll Event gesendet. Das zusätzliche Feld für den Listener-Typ fehlt. Dieses Verhalten konnte nicht in IEEE 802.1Qat wiedergefunden werden. Der Stack ist so konfiguriert, dass dieser Listener als Listener Ready korrekt registriert wird. Jedoch wird die Registrierung auf Grund des LeaveAll Events danach wieder entfernt.

Wird ein Talker Failed-Attribute registriert und der Grund für die fehlgeschlagene Talker-Registrierung verschwindet, so wird die Talker Failed-Registrierung entfernt und ein Talker Advertise-Attribut wird registriert. Ändert sich eine Listener-Attributregistrierung in seinem Typ, so bleibt die bestehende Listener-Attributregistrierung bestehen.

Um die Gefahr zu vermeiden, dass der AVB-Stack einen Wechsel des Listener-Typs übersieht und dahingehend eine Rückmeldung an einen höheren Layer, z.B. dass das Senden jetzt möglich ist, fehlt, wurde die lokale Verwaltung für Listener-Attribute geändert.

Die drei verschiedenen Listener-Typen besitzen jetzt jeweils einen eigenen Speicher-Array. Das hat zu Folge, dass der Wert in AttributeType jetzt im Falle von Listener-Attributen nicht mehr genau einen Array bezeichnet sondern drei. Daher ist das, in dieser Anwendung, attributidentifizierende Merkmal der Offset mit dem diese innerhalb des attributes Arrays innerhalb der MAD-Komponente angesprochen werden.

Ein Wechsel des Typs für eine Listener-Attributregistrierung ist jetzt immer mit einem Deregistrierung und einer Registrierung eines neuen Attributs verbunden und daher auch einfacher in höhere Layer wie SRP weiterzumelden.

5.6.2 Implementation des SRP-Layers

Mit den von den MVRP- und MSRP-Modulen angebotenen APIs ist es bereits möglich Streams und Streamkonsumenten im Netzwerk anzumelden.

Das in diesem Anschnitt besprochene SRP-Modul soll die Handhabung dieser beiden Protokolle für den Anwender vereinfachen. Es setzt auf MMRP, MVRP und MSRP auf. Beim Initialisieren des Moduls werden in den darunterliegenden Modulen Callback-Funktionen angemeldet. Diese werden ausgeführt, wenn in MMRP, MVRP oder MSRP Indication Callback-Funktion ausgeführt werden (siehe Abbildung 4.4 und Abbildung 5.7). Die dort übergebenen Parameter werden so in das SRP-Modul weitergereicht und das SRP-Modul wird über Änderung der Attribute benachrichtigt

Für Talker und Listener wurde jeweils eine Datenstruktur geschaffen. Diese Datenstrukturen heissen navb_srp_talker_t und navb_srp_listener_t. In diesen Datenstrukturen sind die Werte für Talker und Listener in Host Byteorder gespeichert. Diese müssen mit ihrer Initialierungsfunktion vor der Erstbenutzung mit Werten initialisert werden. Das SRP-Modul enthält jeweils einen Array aus Zeigern auf beide Datenstrukturen. Beim Erstellen eines Talkers oder Listeners wird dort ein Eintrag erstellt.

In den Datenstrukturen befindet sich ebenfalls ein State für eine State Machine, die den Zustand des jeweiligen Talkers bzw. Listeners wiederspiegelt. Der State sagt aus, ob es einem Talker erlaubt ist zu senden bzw. ob es einen verfügbaren Listener gibt. Das Gleiche gilt für einen Listener. Der State sagt aus, ob es einen Talker gibt der bereit ist den Stream zu senden.

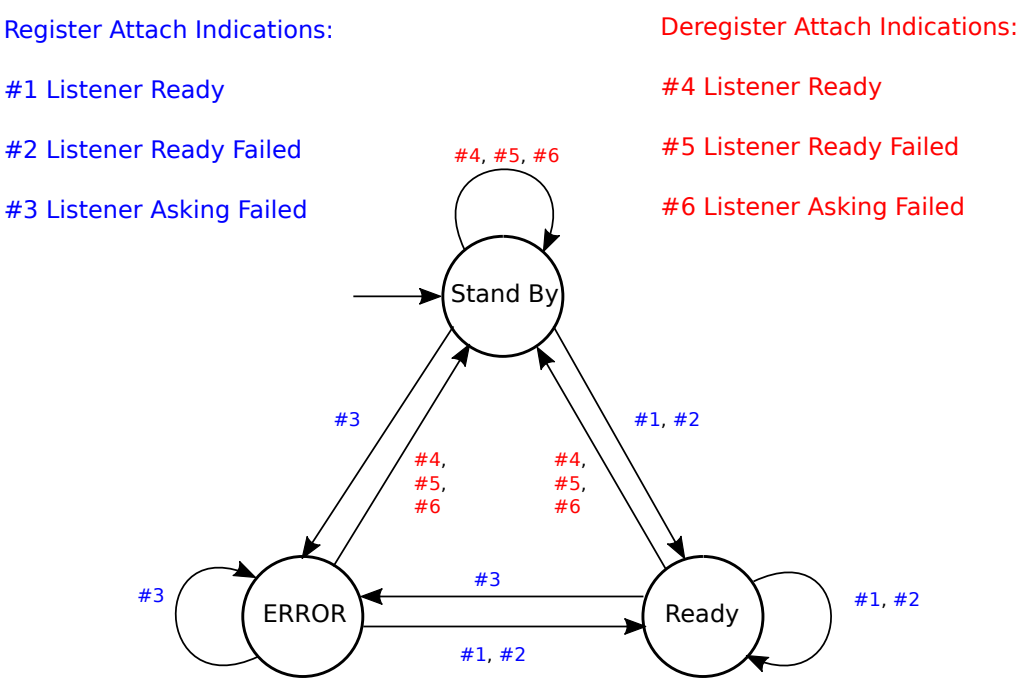

Abbildung 5.14: SRP Talker State Machine

Abbildung 5.14 zeigt die State Machine eines Talkers. Register Attach zeigt an, dass ein neuer Listener registriert wurde. Im Zustand Stand By wartet der Talker. Ready zeigt an, dass mindestens ein Listener bereit ist den Stream zu empfangen. Der ERROR State deutet an, dass zwar ein Listener vorhanden ist aber Mangels Bandbreite oder eines anderen Grunds, nicht gesendet werden darf.

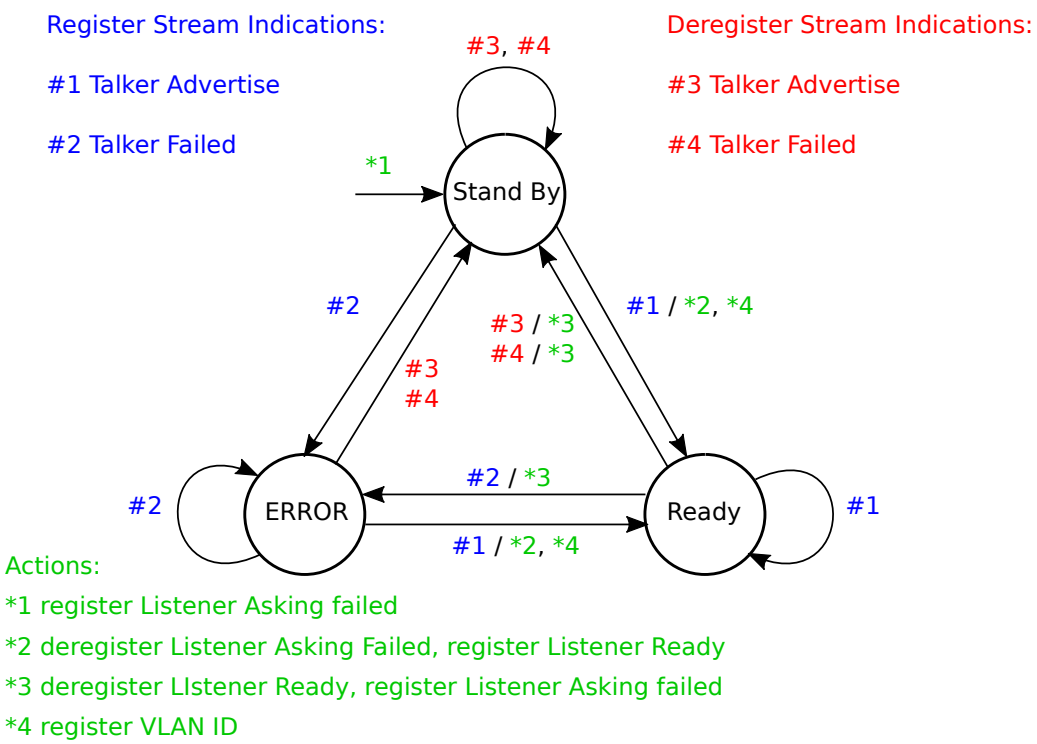

Abbildung 5.15: SRP Listener State Machine

Abbildung 5.15 zeigt die Listener State Machine. Wird sie gestartet wird ein Listener Asking Failed registriert, da in diesem Moment noch nicht bekannt, ist ob ein dazugehöriger Talker existiert. Wird eine Talker Advertise Indication ausgelöst wird ein Listener Ready registriert und der schon vorhandene Listener Asking Failed deregistriert. Ebenfalls wird die im Talker vorhandene VLAN ID registriert um Stream empfangen zu können. Diese war beim Start dieser State Machine ebenfalls unbekannt.

5.7 Architektur des AVB-Stacks

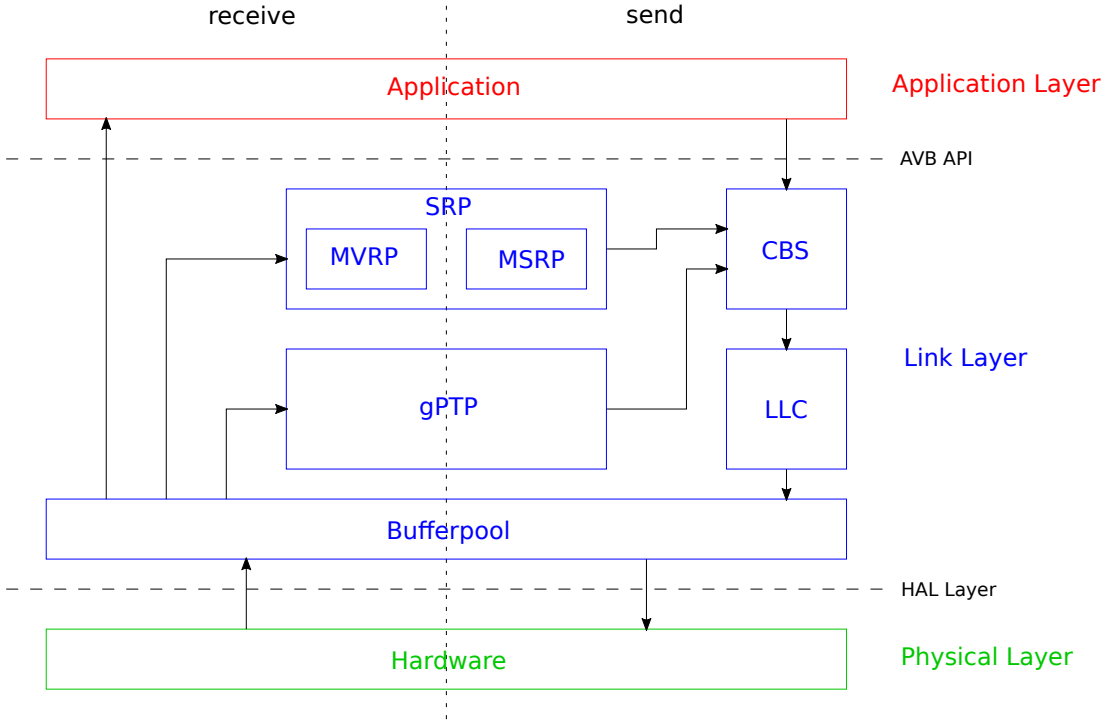

Abbildung 5.16: Architektur des AVB-Stacks

Abbildung 5.16 zeigt die Architektur des AVB-Stacks. Das SRP-Modul als auch das gPTP-Modul empfangen und senden Frames. Das Empfangen geschieht per Direktzugriff auf den Bufferpool. Zum Senden wird der Umweg über das CBS-Modul genommen. Anwendungen verhalten sich auf die gleiche Art.

6 Evaluierung und Qualitätssicherung

Dieses Kapitel beschäftigt sich mit der Sicherstellung der allgemeinen Funktionalität des AVB-Stacks. Anhand von Testfällen soll die Korrektheit und das zu AVB-konforme Verhalten des Stacks überprüft werden. Dies erfolgt in Modultests, um die korrekte Funktion der Teilkomponenten zu überprüfen. Ein abschliessender Gesamttest mittels eines praxisnahen Versuchsaufbau ist ebenfalls geplant gewesen aber aus Zeitgründen nicht mehr umgesetzt worden. Dennoch wird dieser Versuchsaufbau und dessen Zweck nachfolgend besprochen. Eine Testübersicht erfolgt in tabellenform, die die Nummer des Tests, eine kurze Beschreibung, die erwartete Reaktion und das Ergebis des Tests beinhaltet.

6.1 Modultests

Für jedes des diesem Stack befindlichen Moduls wurde ein passendes Testmodul erstellt. Der Sinn dieser Tests besteht darin, die korrekte Funktion von bereits hinreichend komplexen Teilkomponten zu gewährleisten und so überhaupt erst die Entwicklung eines noch komplexeren Gesamtstücks zu ermöglichen. Nur wenn Teilkomponenten ordnungsgemäss funktionieren, besteht die Möglichkeit eines ordnungsgemässen Betriebs des Stacks. Die nachfolgenden Abschnitte befassen sich mit den Tests der jeweiligen Module.

6.1.1 Test des Timer-Moduls

Das Timer-Modul ist ein zentraler Bestandteil des AVB-Stacks. Es wird innerhalb des SRP-Moduls aber auch als allgemeines Utility-Modul von anderen Modulen, wie z.B. dem LLC-Modul verwendet. Es wurde getestet, ob Timer sauber gestartet und zur eingstellten Zeit auslaufen. Ob die Timer korrekt in die Liste der laufenden Timer eingefügt wurden, falls schon Timer aktiv gewesen sind. Ausserdem wurde getestet ob Timerwerte korrekt berechnet, optionale Callback-Funktionen korrekt ausgeführt und Timer ggf. neugestartet wurden. Die in der Tabelle als „One Shot Timer“ bezeichneten Timer, sind Timer die

nach dem Auslaufen nicht automatisch neugestartet werden. „Continuous Timer“ sind so konfiguriert, dass sie von der Timer-ISR automatisch wieder neugestartet werden. Die Tests erfolgten mit an dem NetX-Boards angeschlossener RS-232 Schnittstelle und die Testergebnisse wurden auf der seriellen Konsole beobachtet. Eine dem Timer-Modul zugehörige Debug-Ausgabe gibt modulinterne Informationen aus und macht es so möglich alle berechneten Zeiten extern nachzurechnen und zu vergleichen.

#	Testinhalt	Erwartung	Ergebnis
1	startet einen One Shot Timer, 5s, keine Callback	Timer wird gestartet und läuft nach 5s aus	✓
2	startet einen One Shot Timer, 5s, mit Callback	Timer wird gestartet und läuft nach 5s aus, Callback wird ausgeführt	✓
3	startet einen Continuous Timer, 5s, mit Callback und Parameter für Callback	Timer wird gestartet und läuft nach 5s aus, Callback wird ausgeführt	✓
4	startet einen Continuous Timer, 5s, mit Callback und Parameter für Callback, stoppt Timer sofort wieder	Timer wird gestartet und ordnungsgemäss gestoppt	✓
5	startet drei Timer mit 5s, 10s und 15s, Timer mit 5s wird gestoppt	Timer mit 5s soll korrekt aus der Liste aktiver Timer entfernt werden, Zeiten für nachfolgende Timer sauber neuberechnet	✓
6	startet drei Timer mit 5s, 10s und 15s, Timer mit 10s wird gestoppt	Timer mit 10s soll korrekt aus der Liste aktiver Timer entfernt werden, Zeiten für nachfolgende Timer sauber neuberechnet	✓
7	startet drei Timer mit 5s, 10s und 15s, Timer mit 15s wird gestoppt	Timer mit 15s soll korrekt aus der Liste aktiver Timer entfernt werden	✓
8	startet zwei Timer mit 10s, und 15s, danach 5s	Timer mit 5s soll als erster in der Liste aktiver Timer platziert und HWTimer auf diesen neueingestellt werden	✓
9	startet zwei Timer mit 5s, und 15s, danach 10s	Timer mit 10s soll in der Liste aktiver Timer zwischen den beiden anderen Timern platziert werden	✓
10	startet zwei Timer mit 5s, und 10s, danach 15s	Timer mit 15s soll in der Liste aktiver Timer hinter den beiden anderen Timern platziert werden	✓

Tabelle 6.1: Testübersicht des Timer-Moduls Teil 1

#	Testinhalt	Erwartung	Ergebnis
11	startet vier Continuous Timer, alle haben die gleiche Laufzeit und Callbacks, die jeweils eine der vier Status-LEDs blinken lassen	Timer laufen korrekt aus und werden neugestartet, LEDs blinken	✓
12	startet zwei Continuous Timer und zwei One Shot Timer, alle haben die gleiche Laufzeit und Callbacks, die jeweils eine der vier Status-LEDs blinken lassen, Callbacks der One Shot Timer starten diese manuell neu	Timer laufen korrekt aus und werden neugestartet, LEDs blinken	✓

Tabelle 6.2: Testübersicht des Timer-Moduls Teil 2

Zusätzlich zu den hier aufgelisteten Tests war eine Zeitmessung geplant, die die Zeit vom Starts eines Timers bis zur tatsächlichen Ausführung seiner Callback-Funktion angibt. Die ISR des Hardware-Timers wird zeitnah zur eingestellten Zeit gestartet. Jedoch wird eine mögliche vorhandene Callback-Funktion nur in einen dafür vorgesehenen FIFO gespeichert. Dieser FIFO wird in der Background-Task, die mit niedrigster Priorität läuft und daher unterbrechbar ist abgearbeitet. Daher kann es zu Jitter kommen, der die Zeit vom Starten eines Timers bis zum Bearbeiten einer Callback-Funktion verlängert. Dieser Jitter ist im Verhältnis zu den in den MRP eingestellten Timern sehr gering. Jedoch wird auch ein Timer im LLC-Modul benutzt. Dort wird ein Timer gestartet, wenn ein Frame-Versand beginnt und der ausläuft wenn der Frame versendet wurde. Dabei wird das CBS-Modul über eine Callback-Funktion benachrichtigt um, ggf. einen nächsten Frame zu versenden. In diesem Fall kann die Timerlaufzeit sehr kurz werden.

$$FrameOverhead = Preamble + SFD + CRC + InterFrameGap \tag{6.1}$$

$$FrameOverhead = 7Bytes + 1Bytes + 4Bytes + 12Bytes = 24Bytes \tag{6.2}$$

$$FrameHeader = SourceAddress + DestinationAddress + Ethertype \tag{6.3}$$

$$FrameHeader = 6Bytes + 6Bytes + 2Bytes = 14Bytes \tag{6.4}$$

$$minFrameSize = 46Bytes \tag{6.5}$$

$$minBytesToSend = FrameOverhead + FrameHeader + minFrameSize \tag{6.6}$$

$$minBytesToSend = 24Bytes + 14Bytes + 46Bytes = 84Bytes \tag{6.7}$$

$$minTransferTime = \frac{minBytesToSend}{PortSpeedByte/s} \tag{6.8}$$

$$minTransferTime = \frac{84Bytes}{100Mbit/s} = 0,00000672s = 6,72\mu s \tag{6.9}$$

Gleichung 6.1 bis Gleichung 6.9 zeigt die Berechnung der kürzesten möglichen Sendezeit für einen Frame innerhalb des LLC-Moduls. Diese ist mit 6,72µs sehr kurz im Vergleich zur nächstmöglichen kurzen Timer-Laufzeit, des Join Timers mit 0,01s innerhalb von MRP.

Bei minimaler Framegrösse sollte im Idealfall alle 6,72µs ein neuer Frame abgeschickt werden können. Ist der Jitter aber zu gross verzögert sich die Bearbeitung der Callback-Funktion in der Background Task und der Framedurchsatz sinkt.

Eine Messung des Timer-Jitters ist daher von Interesse, um Aussagen über die Performance machen zu können.

6.1.2 Test des LLC-Moduls

Das LLC-Modul hat die Aufgabe Frames vom Bufferpool zu holen und diese Frames zu öffnen. Nach dem Schreiben in die Frames, diese zu schliessen. Danach die HAL-Sendefunktion aufzurufen. Dabei soll ein Timer gestartet werden. Nach Ablauf dieses

Timers sollen die Frames freigegeben werden und die CBS Callback-Funktion aufgerufen werden.

Der in Abbildung 6.1 gezeigte Aufbau wurde für diesen Modultest benutzt. Das NetX-Board ist direkt über Ethernet mit dem PC, auf dem Wireshark läuft, verbunden. Eingehende Frames werden in Wireshark auf ihre Korrektheit untersucht. Zusätzlich wird die Konsolenausgabe des NetX-Boards überwacht.

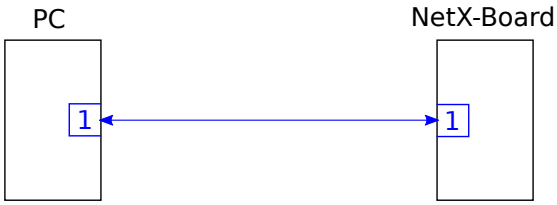

Abbildung 6.1: Die Kommunikation zwischen NetX-Board und PC

#	Testinhalt	Erwartung	Ergebnis
1	Frame von Bufferpool holen, Frame wieder freigeben, neuen Frame holen, hineinschreiben, Frame schliessen, absenden	Funktionen zum Holen, Freigeben, Schliessen und absenden sollen ohne Fehlercode ausgeführt werden, Frame soll korrekt abgesendet werden, Callback soll nach dem Absenden ausgeführt werden	✓

Tabelle 6.3: Testübersicht des LLC-Moduls

6.1.3 Test des Cedit Based Shaper-Moduls

Die Aufgabe des CBS-Moduls ist es Frames mit verschiedenen Prioritäten zu verschicken. Diese werden intern in Queues gesammelt und entsprechend des CBS-Algorithmuses versendet. Die zwei unterstützten Prioritäten sind AVB SR Class A und BE. Aus diesem Grund sind nur Tests mit diesen beiden Prioritäten ausgeführt wurden. Tests mit Priorität AVB SR Class B existieren nicht.

Die Testbedingungen und der Testaufbau sind identisch mit dem in Abbildung 6.1 dargestellten. Der Inhalt der Frames enthält entsprechend ihren Prioritäten 0xaa bzw. 0xbe als Bitmuster, um diese besser in Wireshark voneinander unterscheiden zu können. Zusätzlich ist mit einer, in das CBS-integrierten Debug-Ausgabe, nach dem Versandt des letzten Frames überprüft worden, ob Werte, z.B. Credit einer Queue, korrekt sind. Dies konnte erst nach dem Versandt überprüft werden, da eine Konsolenausgabe ein blockierender Funktionsaufruf und er daher den CBS-Algorithmus behindert würde.

#	Testinhalt	Erwartung	Ergebnis
1	Frame holen, als AVB SR Class A preparieren, Frame abschicken	Frame wird ohne Fehlermeldung verschickt, Frame wird in Wireshark korrekt angezeigt	✓
2	ein AVB SR Class A und ein BE Frame wird prepariert, zuerst wird der AVB SR Class A Frame abgeschickt, danach BE	beide Frames werden korrekt in Wireshark angezeigt, der AVB SR Class A Frame wird zuerst verschickt	✓
3	ein AVB SR Class A und ein BE Frame wird prepariert, zuerst wird der BE Frame abgeschickt, danach der AVB SR Class A	beide Frames werden korrekt in Wireshark angezeigt, der BE Frame wird zuerst verschickt	✓
4	zwei Frames holen und als BE preparieren, beide abschicken	beide Frames werden ohne Fehlermeldung verschickt, beide Frames werden in Wireshark korrekt angezeigt	✓
5	zwei Frames holen und als AVB SR Class A preparieren, beide abschicken	beide Frames werden ohne Fehlermeldung versandt, beide Frames werden in Wireshark korrekt angezeigt	✓
6	sechs Frames holen und fünf AVB SR Class A preparieren und einen als BE, zuerst alle AVB SR Class A Frames absenden dann BE	alle Frames werden ohne Fehlermeldung versandt, alle Frames werden in Wireshark korrekt angezeigt, jedoch wird erst ein AVB SR Class A Frame versendet, dann wegen des negativen Credits der BE Frame und dann die restlichen AVB SR Class A Frames	✓

Tabelle 6.4: Testübersicht des CBS-Moduls

6.1.4 Test des gPTP-Moduls

Die Aufgabe des gPTP-Moduls ist es das NetX-Board Teil der gPTP-Domain werden zu lassen. Dazu ist eine Freigabe des AVB-Switches notwendig.

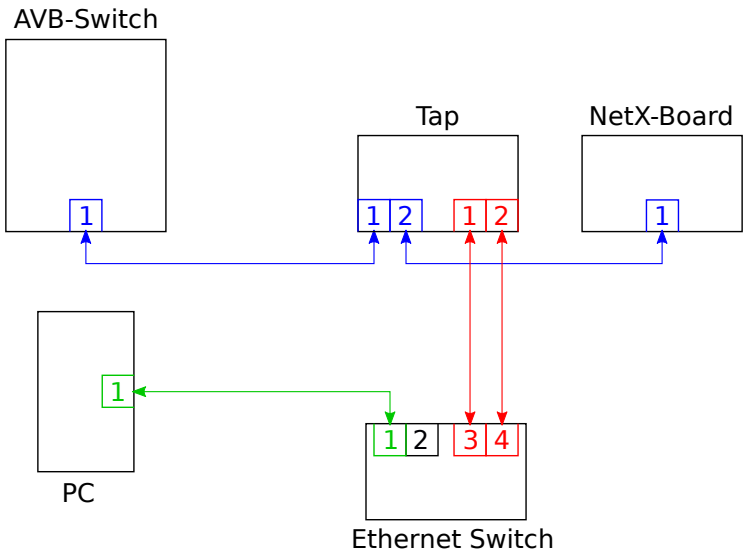

Abbildung 6.2: Die Kommunikation zwischen NetX-Board und AVB-Switch

Die Abbildung 6.2 verdeutlicht den Testaufbau für diesen Modultest. Eine Kommunikation erfolgt zwischen dem AVB-Switch und einem NetX-Board (hier in der Farbe Blau dargestellt). Die Kommunikation wird mittels einer Tap abgriffen und jeweils für jede Kommunikationsrichtung getrennt auf einem extra Port ausgegeben (hier in der Farbe Rot dargestellt). Da der überwachende PC mit Wireshark nur über einen Port verfügt, wurden die getrennten Übertragungsrichtungen mittels eines Ethernet Switches wieder zusammengeführt.

Während des Test wurde die Konsolenausgabe des AVB-Switches regelmässig überprüft und dort der Wert „802.1AS Capable“ für den entsprechenden Port beobachtet.

#	Testinhalt	Erwartung	Ergebnis
1	auf Path Delay Request Frame warten, Path Delay Response und Path Delay Resonse Follow Up Frame senden	Wert von „802.1AS Capable“ wird als TRUE angezeigt	✓

Tabelle 6.5: Testübersicht des gPTP-Moduls

Zusätzlich zum erwarteten Testergebnis konnte ebenfalls die Kommunikation zwischen NetX-Board und AVB-Switch überwacht werden und die gesendeten Nachrichten einer Kommunikation von einem XMOS AVB Audio Endpoint gegenübergestellt werden. Das Verhalten war wie erwartet identisch.

6.1.5 Test des MVRP-Moduls

AVB-Kommunikation ist nur innerhalb des gleichen VLANs möglich. Das MVRP-Modul hat die Aufgabe VLAN-Attribute zu deklarieren.

Im Laufe der Entwicklung wurde das korrekte Deklarieren und Registrieren von VLAN-Attributen, das korrekte Entpacken von MVRP PDUs mit auf verschiedenste Arten gepackten VLAN-Attributen und schliesslich das Anmelden eines VLANs bei einem AVB Switch getestet.

Abbildung 6.3 zeigt den Testaufbau für des Deklarieren und Registrieren von VLAN Attributen. Ein NetX-Board deklariert dabei ein Attribut. Während das andere NetX-Board das neue Attribut bei sich registriert. Die Kommunikation beider Boards kann über die Tap abgegriffen und auf dem PC in Wireshark betrachtet werden. Zusätzlich existiert die Möglichkeit detailiert Debug-Ausgaben für die Konsole zu beobachten.

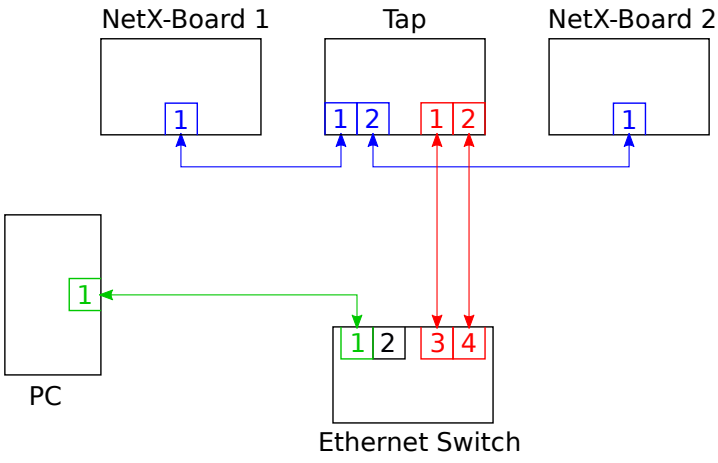

Abbildung 6.3: Die Kommunikation zwischen zwei NetX-Boards

Tabelle 6.6 zeigt die Deklarierungs- und Registrierungstests zwischen zwei NetX-Boards.

#	Testinhalt	Erwartung	Ergebnis
1	starten des MVRP-Moduls, es erfolgt keine Deklarierung oder Registrierung eines Attributs	das Modul startet ohne Fehlermeldung, es ist keine Netzwerkkommuniktion vorhanden	✓
2	starten des MVRP-Moduls, Deklarierung eines VLAN-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
3	starten des MVRP-Moduls, zwei Deklarierungen von VLAN-Attributen	das Modul startet ohne Fehlermeldung, ein Board deklariert zwei Attribute, das andere registriert diese ordnungsgemäß	✓
4	starten des MVRP-Moduls, Deklarierung eines VLAN-Attributs und sofortige Entfernung der Deklarierung	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut und entfernt dieses wieder, evtl. kurzzeitger Traffic vorhanden, danach wieder Ruhe	✓
5	starten des MVRP-Moduls, Deklarierung eines VLAN-Attributs und verzögerte Entfernung der Deklarierung	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut und entfernt dieses nach einiger Zeit wieder, Traffic zwischen beiden Boards vorhanden, danach wieder Ruhe	✓

Tabelle 6.6: Testübersicht des MVRP-Moduls Teil 1

Um das korrekte Entpacken von MVRP-Attributen aus PDUs sicherstellen zu können, wurde eine Reihe von, in Abschnitt 5.6.1 genannten verschiedenen Möglichkeiten Attribute zu packen, getestet. Da der AVB-Stack standardmässig immer nur Nachrichten mit genau einem Attribut in PDUs verpackt, mussten die Nachrichten manuell über das Zusammensetzen der einzelnen Datenstrukturen und das Befüllen dieser mit Werten gebaut werden. Daher wird immer nur genau ein Frame verschickt und es findet nur eine sehr kurzfristige Kommunikation zwischen beiden Boards statt. Dies ist aber unerheblich da der Test das korrekte Entpacken und interpretieren der verschiedenen Packvarianten

hat und nicht die anschliessende Verarbeitung die schon mit den vorhergehenden Tests sichergestellt wurde. Der Testaufbau ist identisch mit Abbildung 6.3.

Tabelle 6.7 zeigt die zugehörigen Tests.

#	Testinhalt	Erwartung	Ergebnis
6	Nachricht enthält ein VLAN-Attribut, leaveAll Event ist gesetzt, sonst kein weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
7	Nachricht enthält ein VLAN-Attribut, kein leaveAll Event ist gesetzt, dafür aber weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
8	Nachricht enthält eine Attributliste mit zwei VLAN-Attributen, erstes Attribut hat leaveAll Event und kein weiteres Attribut gesetzt, zweites Attribut hat kein leaveAll Event dafür aber Attribut	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
9	Nachricht enthält zwei gepackte VLAN-Attribute, beide Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
10	Nachricht enthält vier gepackte VLAN-Attribute, die Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓

Tabelle 6.7: Testübersicht des MVRP-Moduls Teil 2

Der Unterschied zwischen Test #4 und #5 besteht darin, dass die in Abbildung 5.13 gezeigten Vectoren zur Aufnahme von Events, aus einem Byte bestehen, die jeweils maximal drei Events aufnehmen können und bei #4 ein Vector ausreicht, während in #5 zwei Vectoren benötigt werden.

Ein abschliessender Test erfolgt mit dem in Abbildung 6.2 dargestellten Testaufbau. In diesem Fall meldet ein NetX-Board ein VLAN direkt bei dem AVB-Switch an.

#	Testinhalt	Erwartung	Ergebnis
2a	starten des MVRP-Moduls, Deklarierung eines VLAN-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete VLAN für den jeweiligen Port ausgewiesen	✓

Tabelle 6.8: Testübersicht des MVRP-Moduls Teil 3

6.1.6 Test des MSRP-Moduls

Das MSRP-Modul hat die Aufgabe das NetX-Board durch die Deklaration von Domain-Attributen, Teil einer AVB-Domain werden zu lassen. In dieser AVB-Domain können Talker und Listener Attribute deklariert werden. Talker existieren als Talker Advertise und Talker Failed Attribute. Für Listener sind dies Listener Ready, Listener Ready Failed und Listener Asking Failed Attribute.

Wie in Unterabschnitt 6.1.5 für das MVRP-Modul beschrieben, wurde für das MSRP-Modul das korrekte Deklarieren und Registrieren von MSRP-Attributen getestet. Ebenfalls getestet wurde das korrekte Entpacken aller MSRP-Attributtypen in jeder möglichen Packform.

Der Testaufbau ist auf Grund der Ähnlichkeiten mit den MVRP-Modultests nach Abbildung 6.3 identisch.

Tabelle 6.9 enthält Tests, die das korrekte Deklarieren und Registrieren von MSRP-Attributen beinhalten.

#	Testinhalt	Erwartung	Ergebnis
1	starten des MSRP-Moduls, es erfolgt keine Deklarierung oder Registrierung eines Attributs	das Modul startet ohne Fehlermeldung, es ist keine Netzwerkkommunikation vorhanden	✓
2	starten des MSRP-Moduls, Deklarierung eines Domain-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
3	starten des MSRP-Moduls, Deklarierung eines Talker Advertise-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
4	starten des MSRP-Moduls, Deklarierung eines Talker Failed-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
5	starten des MSRP-Moduls, Deklarierung eines Listener Asking Failed-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
6	starten des MSRP-Moduls, Deklarierung eines Listener Ready-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓
7	starten des MSRP-Moduls, Deklarierung eines Listener Ready Failed-Attributs	das Modul startet ohne Fehlermeldung, ein Board deklariert das Attribut, das andere registriert dieses ordnungsgemäß	✓

Tabelle 6.9: Testübersicht des MSRP-Moduls Teil 1

Für das MSRP-Modul wurden ebenfalls eine Reihe von Tests entworfen, die das Ziel haben das korrekte Entpacken von Attributen in jeder möglichen Packvariante sicherzustellen. Diese Tests unterteilen sich nach verwendetem Attributtyp. Der Testaufbau ist identisch mit Abbildung 6.3.

#	Testinhalt	Erwartung	Ergebnis
8	Nachricht enthält ein Domain-Attribut, leaveAll Event ist gesetzt, sonst kein weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
9	Nachricht enthält ein Domain-Attribut, kein leaveAll Event ist gesetzt, dafür aber weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
10	Nachricht enthält eine Attributliste mit zwei Domain-Attributen, erstes Attribut hat leaveAll Event und kein weiteres Attribut gesetzt, zweites Attribut hat kein leaveAll Event dafür aber Attribut	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
11	Nachricht enthält zwei gepackte Domain-Attribute, beide Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
12	Nachricht enthält vier gepackte Domain-Attribute, die Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓

Tabelle 6.10: Testübersicht des MSRP-Moduls Teil 2

#	Testinhalt	Erwartung	Ergebnis
13	Nachricht enthält ein Talker Advertise-Attribut, leaveAll Event ist gesetzt, sonst kein weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
14	Nachricht enthält ein Talker Advertise-Attribut, kein leaveAll Event ist gesetzt, dafür aber weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
15	Nachricht enthält eine Attributliste mit zwei Talker Advertise-Attributen, erstes Attribut hat leaveAll Event und kein weiteres Attribut gesetzt, zweites Attribut hat kein leaveAll Event dafür aber Attribut	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
16	Nachricht enthält zwei gepackte Talker Advertise-Attribute, beide Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
17	Nachricht enthält vier gepackte Talker Advertise-Attribute, die Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓

Tabelle 6.11: Testübersicht des MSRP-Moduls Teil 3

#	Testinhalt	Erwartung	Ergebnis
18	Nachricht enthält ein Talker Failed-Attribut, leaveAll Event ist gesetzt, sonst kein weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
19	Nachricht enthält ein Talker Failed-Attribut, kein leaveAll Event ist gesetzt, dafür aber weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
20	Nachricht enthält eine Attributliste mit zwei Talker Failed-Attribute, erstes Attribut hat leaveAll Event und kein weiteres Attribut gesetzt, zweites Attribut hat kein leaveAll Event dafür aber Attribut	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
21	Nachricht enthält zwei gepackte Talker Failed-Attribute, beide Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
22	Nachricht enthält vier gepackte Talker Failed-Attribute, die Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓

Tabelle 6.12: Testübersicht des MSRP-Moduls Teil 4

#	Testinhalt	Erwartung	Ergebnis
23	Nachricht enthält ein Listener-Attribut, genauer Typ nicht angegeben, leaveAll Event ist gesetzt, sonst kein weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
24	Nachricht enthält ein Listener-Attribut, genauer Typ nicht angegeben, kein leaveAll Event ist gesetzt, dafür aber weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
25	Nachricht enthält eine Attributliste mit zwei Listener-Attributen, erster Listener hat keinen Typ und leaveAll gesetzt, dafür aber kein Event, zweites Attribut ist ein Listener Ready, kein LeaveAll aber dafür ein gesetztes Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
26	Nachricht enthält zwei gepackte Listener-Attribute, erster ist Listener Ready, zweiter Ready failed, beide Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓
27	Nachricht enthält fünf gepackte Listener-Attribute verschiedenen Typs, die Attribute haben leaveAll Event und weiteres Event	das zweite Board entpackt die Nachricht korrekt, Struktur und Werte stimmen mit Test überein	✓

Tabelle 6.13: Testübersicht des MSRP-Moduls Teil 5

Desweiteren erfolge ebenfalls ein Testaufbau wie in Abbildung 6.2 dargestellt. Hier kommuniziert das NetX-Board ebenfalls direkt mit dem AVB-Switch und registriert dort seine Attribute.

#	Testinhalt	Erwartung	Ergebnis
2a	starten des MVRP-Moduls, Deklarierung eines Domain-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Domain für den jeweiligen Port ausgewiesen	✓
3a	starten des MVRP-Moduls, Deklarierung eines Talker Advertise-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Talker Advertise für den jeweiligen Port ausgewiesen	✓
4a	starten des MVRP-Moduls, Deklarierung eines Talker Failed-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Talker Failed für den jeweiligen Port ausgewiesen	✓
5a	starten des MVRP-Moduls, Deklarierung eines Listener Asking Failed-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Listener Asking Failed für den jeweiligen Port ausgewiesen	✓
6a	starten des MVRP-Moduls, Deklarierung eines Listener Ready-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Listener Ready für den jeweiligen Port ausgewiesen	✓
7a	starten des MVRP-Moduls, Deklarierung eines Listener Ready Failed-Attributs	in der Ausgabe des seriellen Terminals des AVB-Switches ist das angemeldete Listener Ready Failed für den jeweiligen Port ausgewiesen	✓

Tabelle 6.14: Testübersicht des MSRP-Moduls Teil 6

6.1.7 Test des SRP-Moduls

Das SRP-Modul setzt auf dem MVRP- und MSRP-Modul auf und benutzt deren API um VLAN-, Talker- und Listener Attribute zu deklarieren.

Dieser Modultest besteht ebenfalls aus mehreren Teilen. Zuerst wird die korrekte Funk-

tion zwischen zwei NetX-Boards getestet. Der Testaufbau entspricht ebenfalls dem von Abbildung 6.3. Talker und Listener müssen korrekt im Peer registriert werden und die passenden Callback-Funktionen müssen die im SRP-Modul enthaltenen State Machines triggern.

#	Testinhalt	Erwartung	Ergebnis
1	starten des SRP-Moduls auf beiden NetX-Boards, es erfolgt keine Deklarierung oder Registrierung eines Attributs	das Modul startet ohne Fehlermeldung, es ist keine Netzwerkkommunikation vorhanden	✓
2	starten des SRP-Moduls auf beiden NetX-Boards, ein NetX-Board deklariert einen Talker	das Modul startet ohne Fehlermeldung, es werden Deklarationen für ein VLAN-Attribut und ein Talker Advertise-Attribut erstellt, der Peer registriert diese	✓
3	starten des SRP-Moduls auf beiden NetX-Boards, ein NetX-Board deklariert einen Talker das endere einen Listener	das Modul startet ohne Fehlermeldung, es werden Deklarationen für ein VLAN-Attribut und ein Talker Advertise-Attribut erstellt, der Peer registriert diese, Peer deklariert VLAN- und Listener-Attribut, State Machines auf beiden Seiten nehmen Ready State ein	✓

Tabelle 6.15: Testübersicht des SRP-Moduls Teil 1

Abbildung 6.4 zeigt einen Testaufbau bei dem die Kommunikation über den AVB-Switch läuft. Ziel ist es ein Stream auf einem NetX-Board zu deklarieren und diese Deklarierung auf dem zweiten NetX-Board zu registrieren.

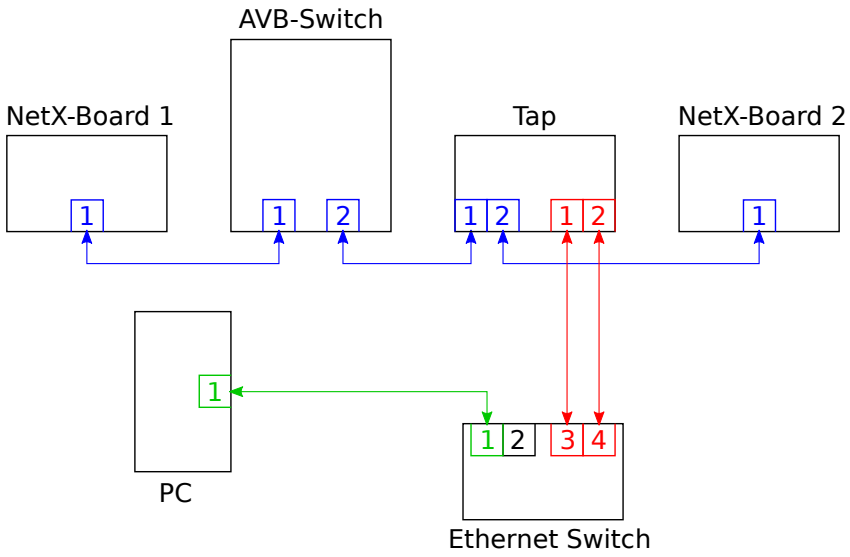

Abbildung 6.4: Die Kommunikation zwischen zwei NetX-Boards über den AVB-Switch

#	Testinhalt	Erwartung	Ergebnis
2a	starten des SRP-Moduls auf beiden NetX-Boards, NetX-Board 1 meldet Talker an, ein Talker Advertise-Attribut und ein VLAN-Attribut werden deklariert	an Port 2 des AVB-Switches erscheint eine Talker Advertise-Deklaration, NetX-Board 2 registriert diese	✗

Tabelle 6.16: Testübersicht des SRP-Moduls Teil 2

Das Ergebis dieses Tests ist negativ. An Port zwei des AVB-Switches erscheint eine Talker Failed-Deklaration. Dieses ist im Laufe der Entwicklung des Stacks mehrfach geprüft worden und erfolgreich gewesen, Der entstandene Fehlercode ist „PORT IS NOT AVB CAPABLE“. Ein Grund für das jetzige scheitern der Talker Advertise-Propagierung konnte auf Grund von zeitlichen Beschränkungen in der Entwicklung nicht gefunden werden (siehe Abschnitt 6.3). Solange dieses Problem nicht behoben ist, ist es nicht möglich den AVB-Switch mit dem AVB-Stack zu benutzen.

6.2 Sicherstellung des Prorisierten Versendens von AVB-Frames

Um Kernfunktionalität von echtzeitpriorisiertem Netzwerkverkehr mittel AVB sicherzustellen ist es notwendig diese Eigenschaft in einem praxisnahen Aufbau zu testen. Dies wurde aus Zeitgründen nicht mehr realisiert.

6.2.1 Testaufbau

Der in Abbildung 6.5 dargestellte Testaufbau fungiert als Stress- und Abnahmetest für diesen AVB-Stack. Ob eine Priorisierung von AVB Traffic gegenüber BE Traffic stattfindet ist bei diesem Test von Interesse und soll untersucht werden.

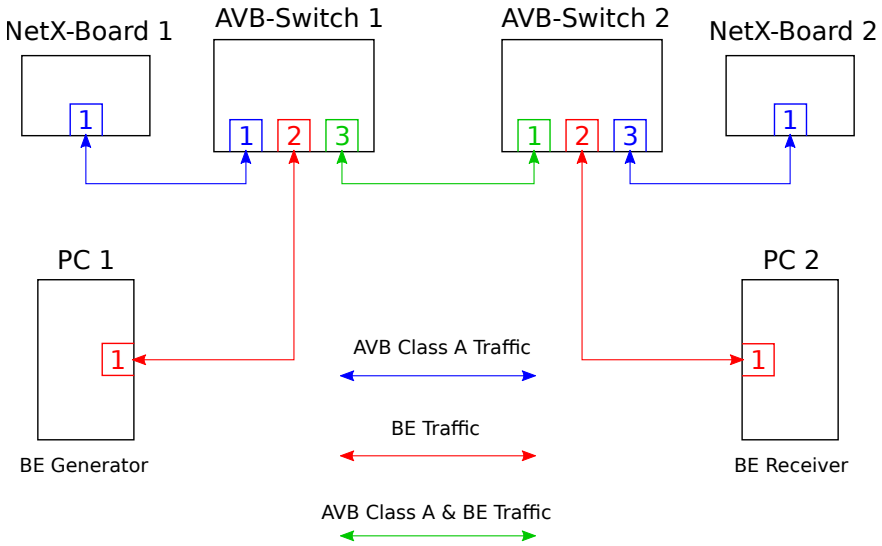

Abbildung 6.5: gemischte AVB SR Class A und BE Kommunikation über zwei AVB-Switches als Stresstest

Es wird eine Messstrecke zwischen zwei AVB-Switchen aufgebaut. An jeden dieser AVB-Switche wird ein NetX-Board und ein PC angeschlossen. NetX-Board 1 meldet einen Talker bzw. Stream im Netzwerk an. Dieser Stream soll die Priorität von AVB SR Class

A haben. NetX-Board 2 meldet einen Listener für diesen Stream an. Nach dem Verbindungsaufbau wird der Stream über die Messstrecke zwischen den beiden AVB-Switche gesendet. PC 1 dient als BE Traffic-Generator der soviel BE Traffic wie möglich generiert und diesen zu dem Empfänger PC 2 sendet. Da die Bandbreite zwischen den beiden AVB-Switchen limitiert ist, entsteht eine Konkurenzsituation zwischen AVB SR Class A und BE Traffic. Sollte der Stack korrekt funktionieren so wird der AVB SR Class A Traffic priorisiert und es findet kein Einbrechen der Bandbreite auch bei Last statt.

Da dieses Testszenario einer realen Anwendung für diesen Stack entspricht ist ebenfalls die CPU-Auslastung relevant.

6.3 Derzeitiges Fehlerbild

Wärend der Entwicklung des Stacks kam es vermehrt zu Problemen beim Packen und Entpacken der MRP PDUs. Daraus resultierte das Registrieren und Verschicken von nicht validen Attributen. Um dieses Problem zu beheben, wurden die hier vorher ausgeführten Tests entwickelt. Diese Tests decken jede mögliche Variante, die diversen Attributtypen der verschiedenen MRP-Implementationen zu packen, ab.

Nachdem diese Fehler beseitigt waren, traten vereinzelt immer noch Fehler beim Entpacken der PDUs auf. Da die Muster der ankommenden PDUs ebenfalls bereits getestet waren, musste der Fehler woanders liegen.

6.3.1 Diagnose

Beim Eintreten eines Verarbeitungsfehlers auf Grund von nicht korrekten Werten in einem Frame, wurde dieser Frame auf der seriellen Konsole des NetX-Boards ausgegeben. Da dieser nicht mit dem in Wireshark angezeigten Frame übereinstimmte, wurden zwei Funktionen geschrieben, die dem Überprüfen des Speichers auf Änderungen während der Laufzeit dienen.

Mit einer Initialisierungsfunktion kann eine Stelle im Speicher angegeben werden von der eine bestimmte Anzahl an Bytes gesichert werden soll. Die Speicheradresse wird ebenfalls gespeichert. Mit dem Aufruf der zweiten Funktion wird der Speicher an der gespeicherten Speicherstelle mit dem gesicherten Speicher verglichen.

Der in Abbildung 6.2 dargestellte Testaufbau wurde gewählt, um das Problem näher einzugrenzen. Im Ruhezustand sendet der AVB-Switch MSRP-Frames in einem Intervall von etwa 5s. Die PDU in diesem Frame enthält für jeden Attributtyp eine Nachricht. Die Attributwerte sind mit Nullen gefüllt und das leaveAll Event gesetzt.

Da der Fehler beim Entpacken und Interpretieren, aber nicht beim Verarbeiten der Attributwerte aufgetreten ist, wurde die Attributverarbeitung für MSRP deaktiviert. Auf diese Weise wurden nur ankommende MSRP Frames geöffnet, entpackt, die Attribute interpretiert und der Frame danach wieder geschlossen. Der Zugriff auf den Frame findet nur lesend statt.

Beim Öffnen des Frames wurden die ersten 100 Bytes gesichert. Nach jedem kompletten Entpacken eines Attributs die Speichervergleichsfunktion aufgerufen. Danach wurde, wenn vorhanden, das nächste Attribut entpackt.

Listing 6.1: veränderte Bytes in den ersten 40 Bytes eines MSRP Frames

```
original memory:
startAddress: 0x8060D558
00011900 1D200000 00000000 00000000 00000000
00000000 00000000 00000000 00000222 00262000

current memory:
startAddress: 0x00011E86
00010220 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
```

Listing 6.1 zeigt die gekürzte Ausgabe der Speichervergleichsfunktion. Diese gab nach 20min Laufzeit des NetX-Boards eine Veränderung des Speichers eines Frames an. Zu sehen sind die ersten 40 Bytes. Einige Bytes haben ihren Wert verändert.

Der Fall, dass sich der Speicher eines Frames während des Zugriffs verändert tritt nur sehr selten ein. Die Wartezeit bis dies der Fall ist variiert sehr stark. Dies kann teils nur einige Sekunden dauern, oder wie in diesem Fall ca. 20min. Es ist kein System erkennbar. Die betroffenen Bytes sind oft an der gleichen Stelle. Dies ist aber auch nicht immer der Fall. Das Lesen der Frames ist die einzige Aktivität, die in der Background Task. Andere Tasks, z.B. im TTE Scheduler, sind nicht vorhanden.

Die einzige nebenläufige Aktivität ist das Starten der Receive ISR der Netzwerkschnittstelle. Diese übergibt den Speicher des ankommenden Frames an den Bufferpool.

Die betreffende Stelle im Code, die den Speicher verändert, konnte nicht gefunden werden. Die Lösung dieses Problems ist aber unverzichtbar für die korrekte Funktion des AVB-Stacks.

6.3.2 Notwendige Massnahmen zur Behebung des Fehlers

Es besteht der Verdacht, dass die Verwaltungslogik des Bufferpools nicht korrekt arbeitet bzw. Effekte auftreten, die bei vorheriger Nutzung des Bufferpools mit Zugriff über den TTE Scheduler nicht aufgetreten sind.

Um den Fehler zu finden sollte als nächstes die Verwaltungslogik des Bufferpools näher analysiert werden. Dies konnte aus Zeitgründen nicht mehr getan werden.

7 Fazit und Ausblick

Abschliessend zu dieser Arbeit soll eine kurze Zusammenfassung gegeben werden. Ebenfalls soll eine Übersicht über notwendige Fehlerbehebungen bzw. Verbesserungen erfolgen. Ein Ausblick soll Möglichkeiten der Erweiterung dieses Stacks betrachten.

7.1 Zusammenfassung

Da die Anzahl und Komplexität der heute verbauten elektronischen Endgeräte in modernen Fahrzeugen stetig zunimmt, stoßen traditionelle Bussysteme immer öfter an ihre Grenzen. Um mit den wachsenden Anforderungen schritthalten zu können, ist die Entwicklung neuer Bustechnologien notwendig. Das auf Standard Ethernet aufbauende TTEthernet ist solch eine neue Technologie. Jedoch werden die Vorteile dieses Protokolls mit erheblichem Konfigurationsaufwand erkauft, der mit stetig steigender Netzwerkkomplexität ebenfalls steigt. Ein weiterer Lösungsansatz ist das Audio/Video-Bridging Protokoll. Es ermöglicht eine dynamische Konfiguration. Diese erlaubt es während des Betriebs Netzwerkknoten hinzuzufügen oder zu entfernen. Um eine Implementation eines AVB-Stacks zu schaffen, ist umfangreiches Wissen zu den IEEE Standards 802.1ba, 802.1Qat, 802.1Qav und 802.1Qas angeeignet worden. Es wurde ein Konzept zur Umsetzung dieser Protokolle erarbeitet, das schliesslich in einer konkreten Implementation mündete. Es entstanden voneinander unabhängig benutzbare Module, die aber wie die Evaluation zeigte, alle mit demselben Fehler behaftet sind. Dieser Fehler führt zu Datenkorruption in ankommenden Frames und konnte auf Grund von zeitlichen Limitierungen nicht mehr zeitnah zur Abgabe dieser Arbeit behoben werden.

7.2 Offene Punkte und Verbesserungen

Die Behebung des genannten Fehlers ist eine Notwendigkeit den Stack vollständig einsatzbereit zu machen. Im Falle von Datenkorruption innerhalb von ankommenden Frames besteht keine Datensicherheit und es muss grundsätzlich davon ausgegangen werden, dass jeder Frame korrumpiert sein könnte. Ein Hinweis zur weiteren Vorgehensweise ist im Evalutionskapitel beschrieben.

Die aktuelle Version des CBS-Moduls unterstützt entgegen des ursprünglichen Konzepts nur einen einzigen AVB SR Class A und einen BE Stream. Dies ist ebenfalls auf zeitliche Limitierungen während der Implementierungsphase zurückzuführen. Es währe wünschenswert, wenn das ursprüngliche Konzept umgesetzt werden könnte, um so flexiblere Einsatzszenarien zu haben.

7.3 Ausblick

Die Umbenennung der AVB Task Group zur Time-Sensitive Networking (TSN) Task Group (siehe [4]) führte zu Erweiterungen des AVB-Standards, aus dem schließlich der Nachfolger TSN hervorging. TSN steht für Time Sensitive Networking und besitzt noch geringere Latenzen und ermöglicht preemptiven Netzwerkverkehr.

Eine Möglichkeit TSN-artige Eigenschaften mit dem AVB-Stack zu erreichen besteht darin, diesen so zu erweitern, dass ein Parallelbetrieb mit dem vorhandenen TTE-Stack möglich ist.

Literaturverzeichnis

[1] CoRE Research Group: *Communication over Real-time Ethernet.* – URL https://core-researchgroup.de/

[2] Extreme Networks, Inc.: *Extreme Networks, Inc. Homepage.* – URL www.extremenetworks.com

[3] FlexRay Consortium: Protocol Specification / FlexRay Consortium. Stuttgart, Dezember 2005 (2.1). – Specification

[4] IEEE 802.1 TSN Task Group: *IEEE 802.1 Time-Sensitive Networking Task Group.* – URL https://1.ieee802.org/tsn/

[5] Institute of Electrical and Electronics Engineers: IEEE 802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications / IEEE. September 2008 (IEEE 802.3-2008). – Standard. – ISBN 973-07381-5796-2

[6] Institute of Electrical and Electronics Engineers: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems / IEEE. Juli 2008. – Standard. – 1–300 S

[7] Institute of Electrical and Electronics Engineers: IEEE 802.1Qav - IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks - Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams / IEEE. Dezember 2009 (IEEE 802.1Qav-2009). – Standard. – ISBN 978-0-7381-6143-3

[8] Institute of Electrical and Electronics Engineers: IEEE 802.1Qat - IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks - Amendment 14: Stream Reservation Protocol (SRP) / IEEE. September 2010 (IEEE 802.1Qat-2010). – Standard. – ISBN 978-0-7381-6501-1

[9] Institute of Electrical and Electronics Engineers: IEEE 802.1AS - IEEE Standard for Local and metropolitan area networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks / IEEE. Februar 2011 (IEEE 802.1AS-2011). – Standard. – ISBN 978-0-7381-6536-3

[10] Institute of Electrical and Electronics Engineers: IEEE 802.1BA - IEEE Standard for Local and metropolitan area networks - Audio Video Bridging (AVB) Systems / IEEE. September 2011 (IEEE 802.1BA-2011). – Standard. – ISBN 987-0-7381-7639-8

[11] Institute of Electrical and Electronics Engineers: IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks / IEEE. Dezember 2014. – Standard. – 1–1832 S

[12] LIN-Administration: *Local Interconnect Network*. – URL http://www.lin-subbus.org/. – Zugriffsdatum: 2011-01-06

[13] Lipfert, Jan: *Technical Data Reference Guide - netX500/100*. Hilscher GmbH. Dezember 2008. – URL http://www.hilscher.com

[14] Ltd., XMOS: *XMOS Ltd. Homepage*. – URL www.xmos.com

[15] MOST Cooperation: *Media Oriented Systems Transport*. – URL http://www.mostcooperation.com/. – Zugriffsdatum: 2011-01-06

[16] Müller, Kai: *Time-Triggered Ethernet für eingebettete Systeme: Design, Umsetzung und Validierung einer echtzeitfähigen Netzwerkstack-Architektur*. Hamburg, Hochschule für Angewandte Wissenschaften Hamburg, bachelorsthesis, August 2011

[17] Steiner, Wilfried: *TTEthernet Specification*. TTTech Computertechnik AG. November 2008. – URL http://www.tttech.com

A Anhang

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

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

Ort	Datum	Unterschrift im Original