Entwurf Aufgabe 3

Team (Teamnummer, Teammitglieder)	Nummer 2, Lukas Lühr & Florian Stäps
Aufgabenteilung	Lukas Lühr : Implementation Florian Stäps : Planung
Quellenangaben	• Linux Man pages • https://users.informatik.haw-hamburg.de/~schulz/pub/Verteilte-Systeme/AI5-VSP/Aufgabe3/ • Vorlesungsfolien Verteilte Systeme WS16/17 - Prof. Dr. Klauck
Bearbeitungszeitraum	40h Arbeitszeit +17h Arbeitszeit
Aktueller Stand	• Planung abgeschlossen • Grundlegendes Framework implementiert • Logik fast vollständig implementiert
Änderungen des Entwurfs	Die Zeit Synchronisation Klasse wird nicht mehr mittels Callback angesprochen. (ABB004 ist somit ungültig)
Entwurf	Siehe nachfolgendes Dokument

Inhalt

Überblick der Aufgabenstellung

Es sollte ein Zeitmultiplexverfahren (TDMA) in einer Sprache der Wahl implementiert werden, wir haben uns für C++ entschieden. Das Multiplexing erfolgt auf einem mittels UDP-Broadcast simulierten Funkkanal.

Hierfür werden Frames mit einer Länge von 1000ms und jeweils 25 Slots, welche zum Senden zur Verfügung stehen, angenommen. Dadurch ist auch die maximale Anzahl der Teilnehmer auf dem Funkkanal auf 25 limitiert.

Sollten zwei Nachrichten in einem Frame empfangen werden, so müssen beide als nicht empfangen angesehen werden.

Warum C++

Die Sprachwahl fiel auf C++, da grade in C++ ein kontrollierter und schneller Zugriff auf Netzwerkdaten via Sockets möglich ist. Auch ein Zeit Overhead durch eine Virtuelle Maschine entfällt vollständig.

Auch besteht meinerseits Interesse daran, den Algorithmus auf einem Micro Controller mit echter Funk Hardware zu realisieren.

Planung

Dieses Kapitel enthält die nicht technische Planung. Im Folgenden bezeichnen wir das Programm für die Sende und Empfangsstation Station als TIMUX. Jede Station welche an der Kommunikation teilnimmt ist eine eigenständige Anwendung.

Datenquelle und Datensenke

Als Datenquelle dient ein Programm, das bereitgestellt wurde und zyklisch zu sendende Daten in 24byte Paketen über die Standardausgabe ausgibt. Unsere Implementation muss also seine Daten über die Standardeingabe beziehen.

Eine Datensenke ist nicht gefordert, daher geben wir sämtliche empfangene Daten über die Standard Ausgabe aus.

Aus der Aufgabenstellung ergibt sich also dieses Übersichtsdiagramm(ABB01). Es ist zu beachten, dass bis zu 25 Sendestationen an dem Funkkanal angeschlossen sein können. Die Interne Struktur von TIMUX wird im entsprechenden Kapitel beschrieben.

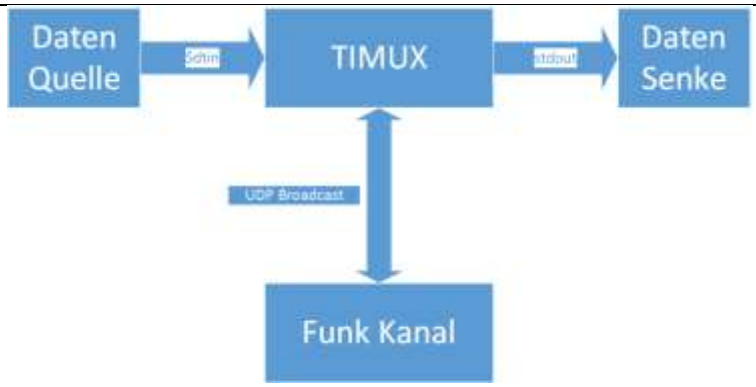

ABB01 – Grobe Übersicht

Nachrichtenkodierung

Die Nachrichten, welche über den Funkkanal ausgetauscht werden, sind jeweils 34 Byte groß und wie folgt aufgebaut:

1 Byte	24 Byte	1 Byte	8 Byte
Stationsklasse(A/B)	Nutzdaten	Next Reserve Slot	Sendezeit Stempel

Dies lässt sich als __packed__ struct, in welche direkt geschrieben wird, gut umsetzen.

Stationsklassen und Zeitsynchronisation

Wie der Kodierung der Nachrichten zu entnehmen gibt es zwei Stationklassen, A und B.

Stationen vom Typ A haben eine genaue Uhr und dienen für Stationen der Klasse B als Referenz um ihre eigene Uhr zu synchronisieren. Auch Stationen der Klasse A synchronisieren ihre Uhren untereinander.

Simulierter Funkkanal/Multicast

Wie im groben Überblick erwähnt, wird der Funkkanal mittelst UDP Broadcast simuliert, dafür wird die Adresse 225.10.1.2(Klasse D) mit einem Port von 15000 + Team Nummer (Also 15002) verwendet.

Hierbei ist zu beachten, dass mehrere UDP Sockets sich auf dem selben Port eines Rechners anmelden können müssen(SO_REUSEADDR [MAN01]). Auch ist darauf zu achten das Netzwerk nicht unnötig mit Broadcast Nachrichten zu fluten, daher sollte die TTL auf 1 gesetzt werden (IP_MULTICAST_TTL [ORA01])

UDP Nachrichten Fluss

Es wird er Nachrichten Fluss in 2 Interessanten Fällen Modelliert, Dem Beitritt eines Clients und dem Beitritt von 2 Clients mit Kollision, der Normale Betrieb wird nicht modelliert, da dieser trivial ist und keine Kollisionen mehr auftreten können.

Erfolgreicher Beitritt eines Clients

Wenn nur ein Client beitritt, kann keine Kollision auftreten, wenn alle Teilnehmer sich richtig verhalten.

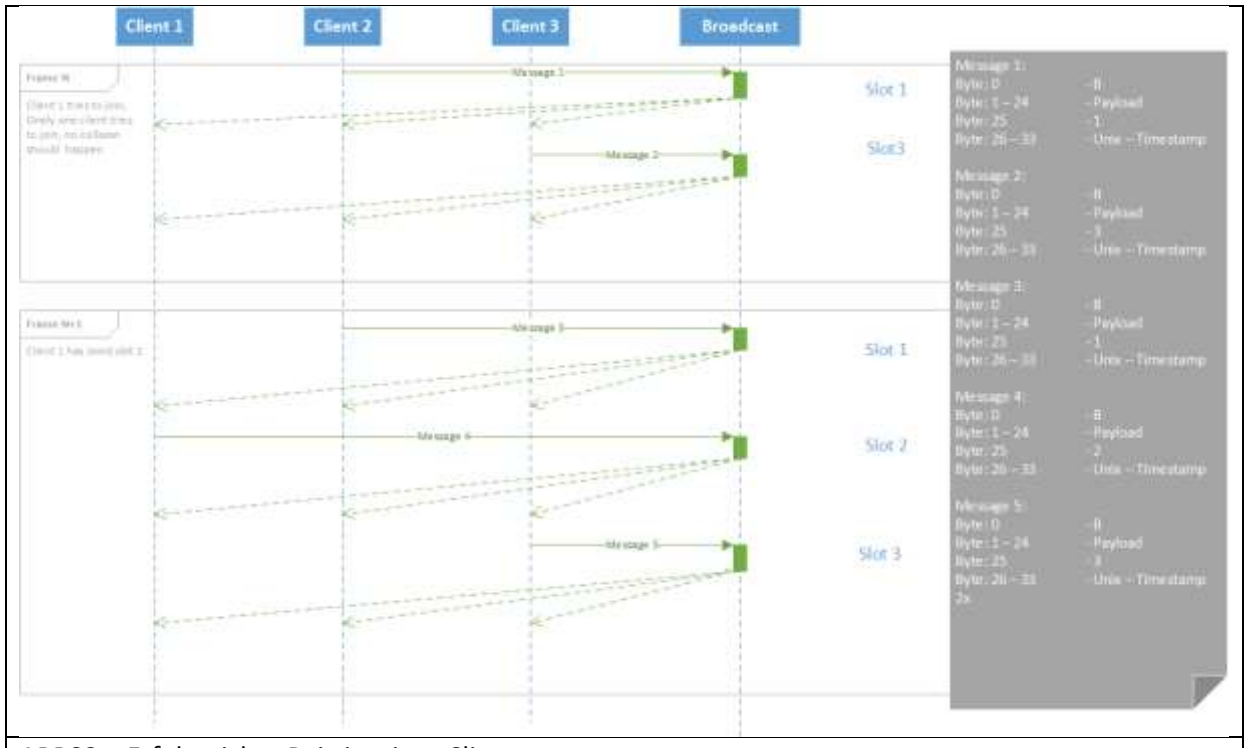

ABB02 – Erfolgreicher Beitritt eines Clients

Es ist notwendig einen Frame vor dem Beitrittsversuch zu Lauschen, um frei Slots des Nächsten Frames zu ermitteln. Im Darauf folgenden Frame wir einer der Freien Slots von dem Client verwendet um mit selbst Daten zu senden.

Beitritt mit Fehlerfall

Eine Kollision Kann Theoretisch nur Auftreten, wenn 2 Clients gleichzeitig versuchen dem System Bei zu Treten.

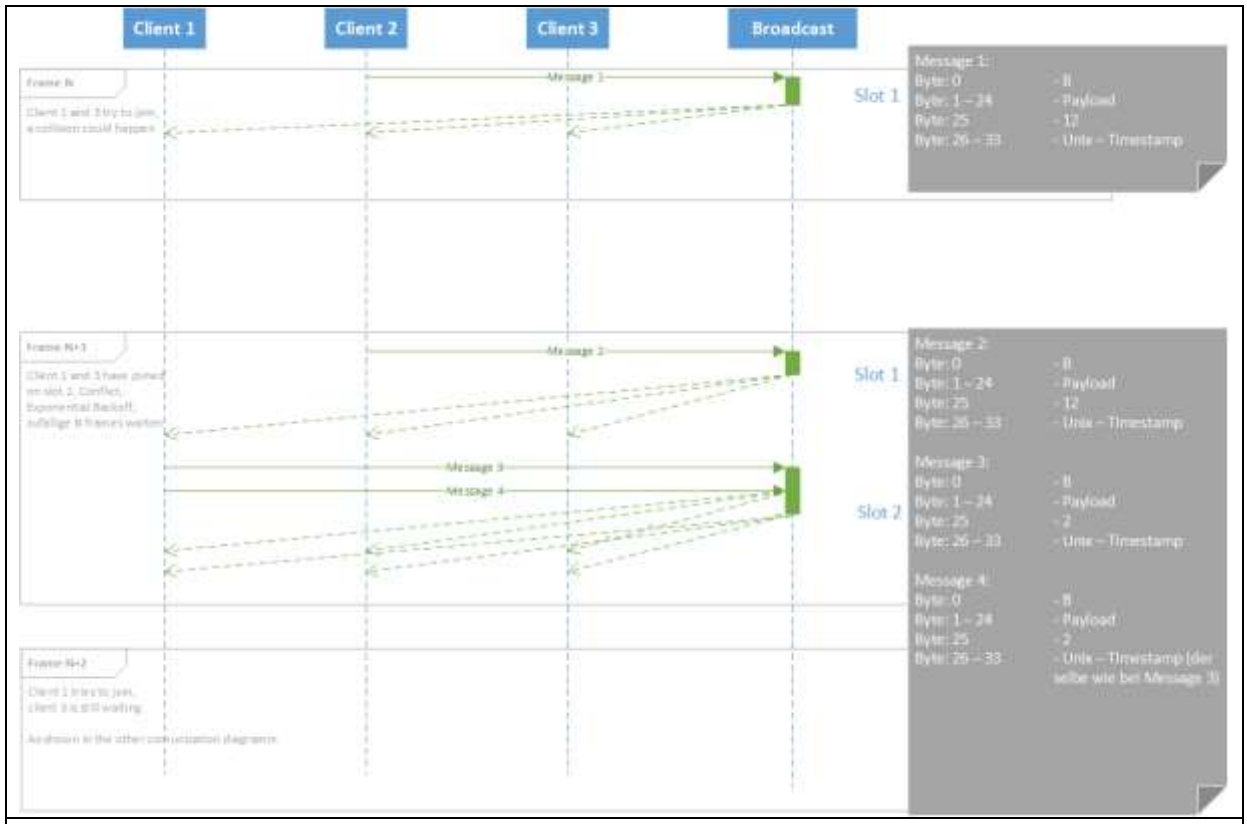

ABB03 – Beitritt mit fehlerfall

Zwei Clients Versuchen Gleichzeitig bei zu treten, und entscheiden sich für denselben freien Slot im Kommenden Frame, eine Kollision wird ausgelöst, und die Clients warten eine Zufällige anzahl von Frames bis zu einem neuen Beitrittsversuch.

Technische Planung

Nach einer groben Planungsübersicht folgt die technische Planung.

Interne Struktur

Intern haben wir das System in gut trennbare Komponenten aufgeteilt:

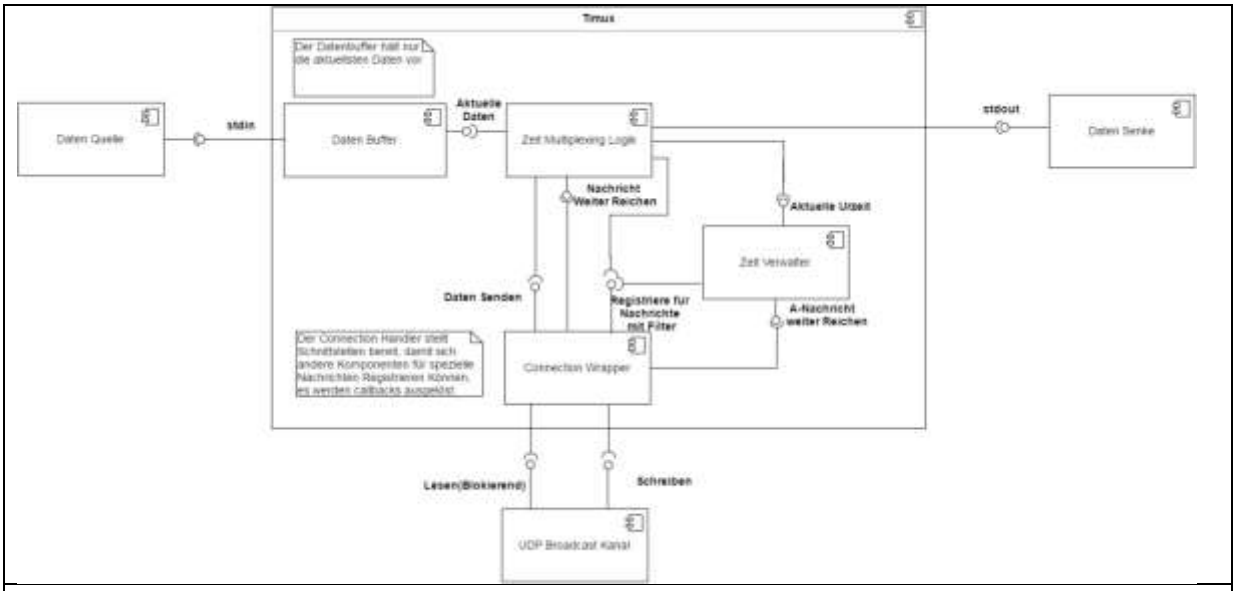

ABB04 - Interne Komponenten Struktur

Eine TIMUX Station besteht somit aus 4 Komponenten, ihre Zuständigkeiten werden nachfolgend beschrieben.

Daten Buffer

Der Datenbuffer ist lediglich dafür zuständig, Daten von der Datenquelle zwischen zu speichern und für die Zeit Multiplex Logik bereit zu stellen. Es werden, wie der Aufgabenstellung zu entnehmen, immer nur die aktuellsten Daten aus der Datenquelle vorgehalten.

Connection Wrapper

Diese Komponente stellt Schnittstellen bereit, um mittels des Observer Patterns auf die Daten aus dem Funkkanal zu zugreifen. So registriert sich etwa die Zeitverwaltungskomponente lediglich für Nachrichten, welche von einer Klasse-A-Station stammen. Die Zeit Multiplexing Einheit registriert sich für sämtliche 34Byte Nachriten, ungeachtet der Sendestationsklasse.

Auch wird eine Thread sichere Schnittstelle zum nicht blockierenden schreiben auf dem Funkkanal bereitgestellt.

Zeit Verwalter

Für das Synchronisieren der Uhren unter den einzelnen Sendestationen ist diese Komponente zuständig. Zur Synchronisierung mit den anderen Uhren wird die Hälfte der Differenz der eigenen Uhrzeit mit dem Offset addiert und die Sendeuhrzeit einer empfangenen Klasse A Nachricht als neues Offset verwendet.

Zeit Multiplexing Logik

Die gesamte Logik des Zeit Multiplexing Verfahrens wird von dieser Komponente abgehandelt. Also das reservieren von Slots, und das senden in einem Speziellen Slot.

Ablauf der Kommunikation zwischen Komponenten

Die Kommunikation zwischen Komponenten beim senden beziehungsweise beim empfangen einer Nachricht, wird hier gargestellt.

Erhalt einer Nachricht

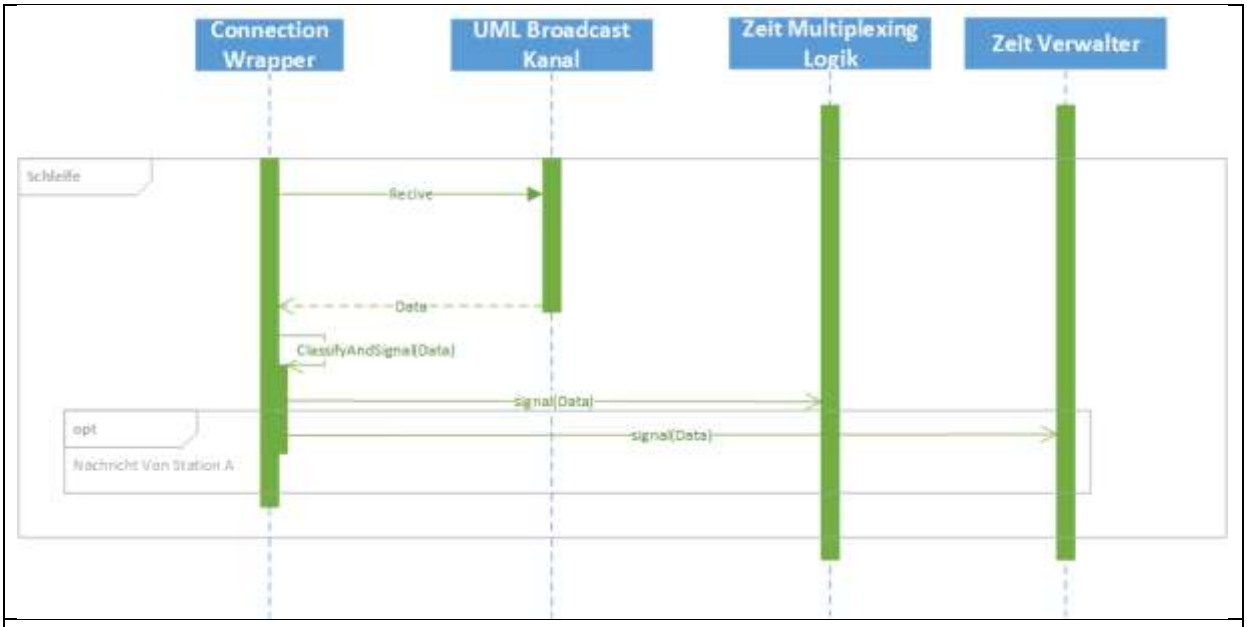

ABB05 – Erhalt einer Nachricht

Der Connection Wrapper liest blockierend auf dem Kanal. Erhält er eine valide Nachricht, so wird in jedem Fall die Zeit Multiplexing Logik asynchron benachrichtigt. Handelt es sich um eine Nachricht von einer Klasse A Station, so wird zusätzlich ebenfalls der Zeit Verwalter benachrichtigt um die Uhr zu kalibrieren.

Senden einer Nachricht

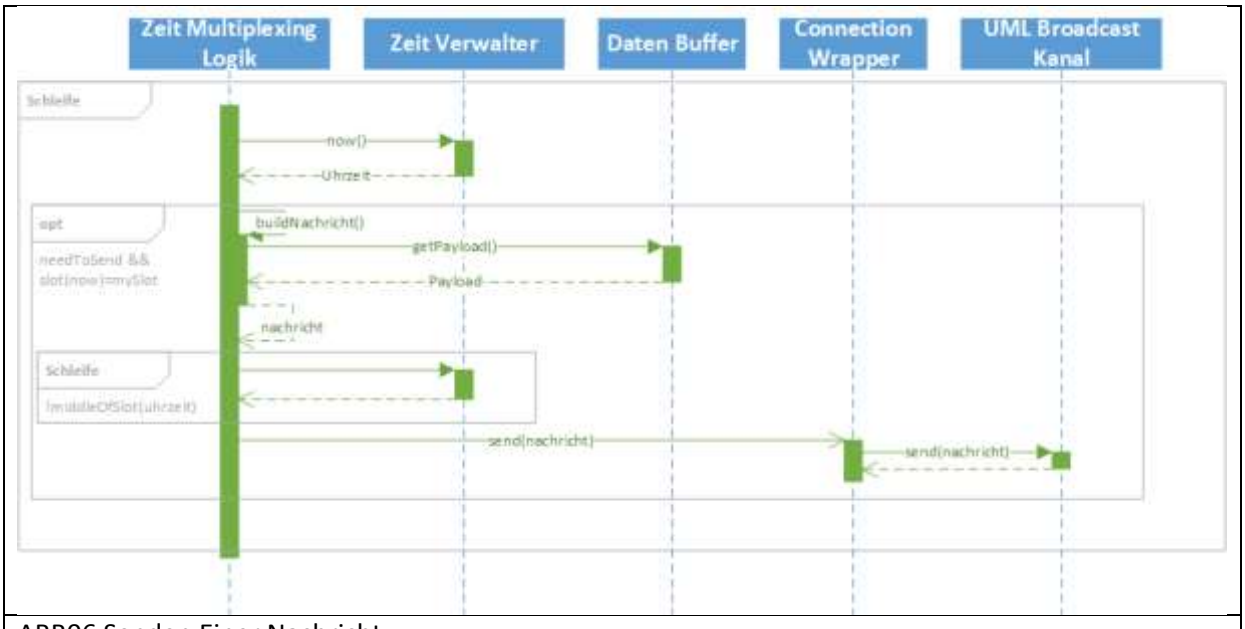

ABB06 Senden Einer Nachricht

Es ist zu beachten, dass vor dem Senden der Nachricht an den Connection Wrapper die aktuelle Uhrzeit in dieser eingetragen wird.

Zeit Multiplexing Logik

Das Zeit Multiplexing lässt sich in zwei Zustände gliedern: Passives Lauschen (Vor dem Eintritt in den Funkkanal muss mitgehört werden, welche Slots im nächsten Frame belegt sind). Und aktives Senden (Die Station sendet Daten, und empfängt gleichzeitig).

ABB07 Zustandsdiagramm der Zeit Multiplexing Logik

Zustandsunabhängig

Um das Zeit Multiplexing nun durchführen und den Verlust von Nachrichten durch Kollisionen simulieren zu können, wird intern pro Slot eines Frames die Anzahl der Nachrichten gezählt, hierüber können Kollisionen erkannt werden. Dies ist, sowohl im aktiven als auch im passiven zustand, notwendig.

Passives Lauschen

Es muss ein Slot gefunden werden, auf welchem im nächsten Frame gesendet werden kann, hierfür werden die Slotreservierungen für den kommenden Frame in einem Array eingetragen.

Sind am ende des aktuellen Frames noch Slots im nächsten Frame verfügbar, so wird zufällig entschieden, ob versucht werden soll an der Kommunikation Aktiv teil zu nehmen. Wird ein solcher Versuch gestartet, so wird zufällig einer der verfügbaren Frames als sende Frame festgelegt, und der Zustand wechselt zu Aktiven Senden.

Es muss mindestens ein Vollständiger Frame abgehört werden, bevor versucht In einem Slot zu senden. Dies ist nur beim initialen Start des Systems zu beachten, da es in der Mitte eines Frames hochgefahren worden sein könnte.

Aktives Senden

Die Station hat sich einen Slot gesucht, auf welchem sie senden möchte.

Es wird nun abgewartet, bis dieser Slot des Aktuellen Frames beginnt. Zu beginn des Slots wird die Nachricht, welche zu senden ist, zusammen gebaut. Nach erstellen der Nachricht, wird auf die Mitte

des Slots gewartet, um die Nachricht zu senden. Ist die Mitte Erreicht, wird der Zeit Stempel in der Nachricht eingetragen und diese über den UDP Kanal gesendet.

Am ende Des Slots wird geprüft, ob genau eine Nachricht in dem Reservierten Slot empfangen wurde, also die eigene Nachricht. Wurden 2 Nachrichten empfangen, so kam es zu einer Kollision und es wird wieder in das Passive Lauschen gewechselt. Wurde keine Nachricht im eigenen Slot empfangen, so wurde der Slot aufgrund von Timing-Problemen verpasst es wird wieder in das Passive Warten gewechselt. Dies sollte allerdings nur bei einem langsamen System(etwa VM) oder einer starken Belastung des Systems auftreten.

Planung der Implementierung

Es wurde eine geeignete Klassenstruktur analog auf Papier entworfen und in C++ erstellt, lediglich Prototypen ohne Implementierung. Diese Struktur wurde mit Hilfe des Tools Doxigen dokumentiert, diese Dokumentation ersetzte den analogen Entwurf.

Nach Fertigstellung der Struktur wurde diese anschließend sukzessiv mit Funktionalität gefüllt.

Die Struktur ist dem angehängtem ZIP Archiv html.zip[ZIP01] zu entnehmen.

Quellen

Bezeichnung	Quelle
MAN01	Linux Man pages
AUF01	https://users.informatik.haw-hamburg.de/~schulz/pub/Verteilte-Systeme/AI5-VSP/Aufgabe3/
VORL01	Vorlesungsfolien Verteilte Systeme WS16/17 - Prof. Dr. Klauck

Anhänge

Bezeichnung	Inhalt	Pfad
ZIP01	Doxygen Dokumentation / Planungsdokumente	./html.zip