Team: 6, Mert Siginc, Michael Müller
Aufgabenaufteilung:

1. Receiver, PayloadServer + Vessel3, SlotFinder
2. Sender, Core, UTCClock, MessageHelper

Quellenangaben:

- http://erlang.org/doc/apps/stdlib/index.html
- http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html

Bearbeitungszeitraum:
Ca. 160 Stunden insgesamt

Aktueller Stand:
Entwurf fertig, erste kleine selbstständige Funktionen getestet.

Änderungen des Entwurfs:

Entwurf:

Gliederung:

1. Allgemeine Beschreibung
2. Wie ist die Architektur aufgebaut?
3. Was sind die verschiedenen Phasen?
    a. Initialisierung
    b. Einstiegsphase
    c. Sendephase
4. Wie sieht der (Funk-) Kanal aus?
    a. Frames
    b. Slots
    c. Socket
    d. Kollisionen
5. Wie sehen die Nachrichten aus?
6. Wie sieht die Station aus?
    a. Stationsname
    b. Wie bekommt man einen Slot und was geschieht danach?
    c. Wie sieht die Datenquelle aus?
    d. Wie sieht die Datensenke aus?
    e. Wie sieht die interne Uhr aus?
    f. Wie verläuft die Initialisierung?
    g. Wie verläuft die Einstiegsphase?
    h. Wie verläuft die Sendephase?
7. Was passiert in den ersten Frames (Beispiel Ablauf)?

Allgemeine Beschreibung:

Die Anwendung die wir Realisieren soll das Zeitmultiplex verfahren umsetzen.
Dabei sollen die Stationen über UDP Multicast (auch Kanal genannt) miteinander kommunizieren. Wobei auf Kollisionen geachtet wird.
Die Vergabe der Sende-Slots wird bei jeder Station lokal individuell bestimmt, indem man Nachrichten aus dem Kanal auswertet. Zudem werden die Nachrichten von Typ A Stationen verwendet, um die Uhren der eigenen Station zu synchronisieren.

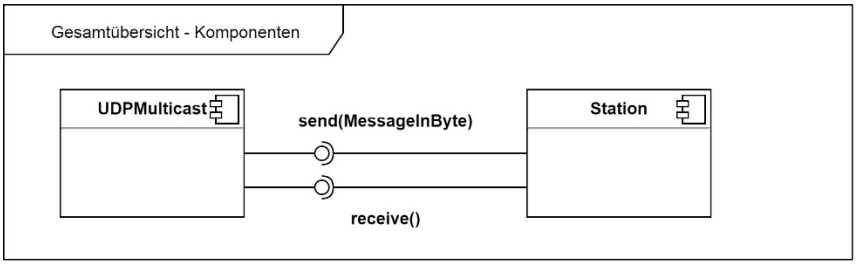
Wie ist die Architektur aufgebaut?

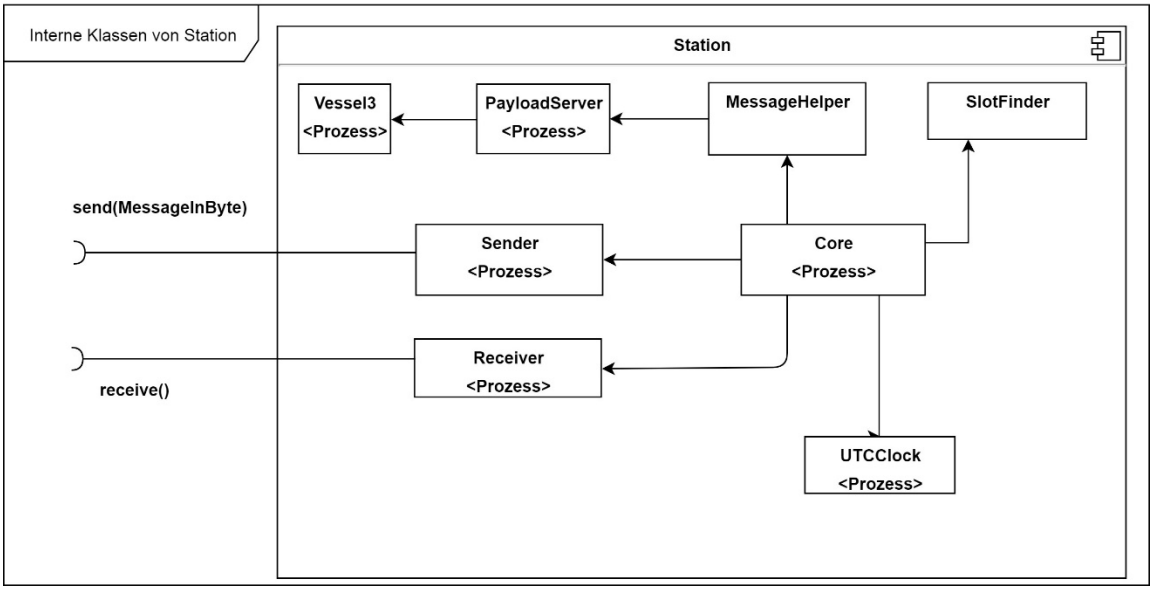
Die Station wird in mehrere Komponenten aufgeteilt.
So besteht eine Station aus:

- Sender (Sendet Nachricht in den UDP Funkkanal)
- Receiver (Empfängt alle Nachrichten aus dem UDP Funkkanal und prüft auf Kollision)
- Core (Verbindet alle Komponenten und besitzt die meiste Logik)
- SlotFinder (Besitzt die Logik aus Nachrichten einen freien Slot (Random) zu finden)
- UTCClock (Die Aktuelle Uhrzeit der Station, gibt zum Beispiel dem Core Bescheid wenn ein neuer Frame beginnt)
- MessageHelper (Hat die Logik der internen Nachrichten Representation und für das Erstellen der zusendenden Nachricht)
- PayloadServer (Bekommt alle Vessel3 Outputs und gibt wenn gefordert den aktuellsten Zurück)
- Vessel3 (Erstellt die Payload für die Nachrichten)

Nähere Infos zu der Station unter „Wie sieht die Station aus?“.

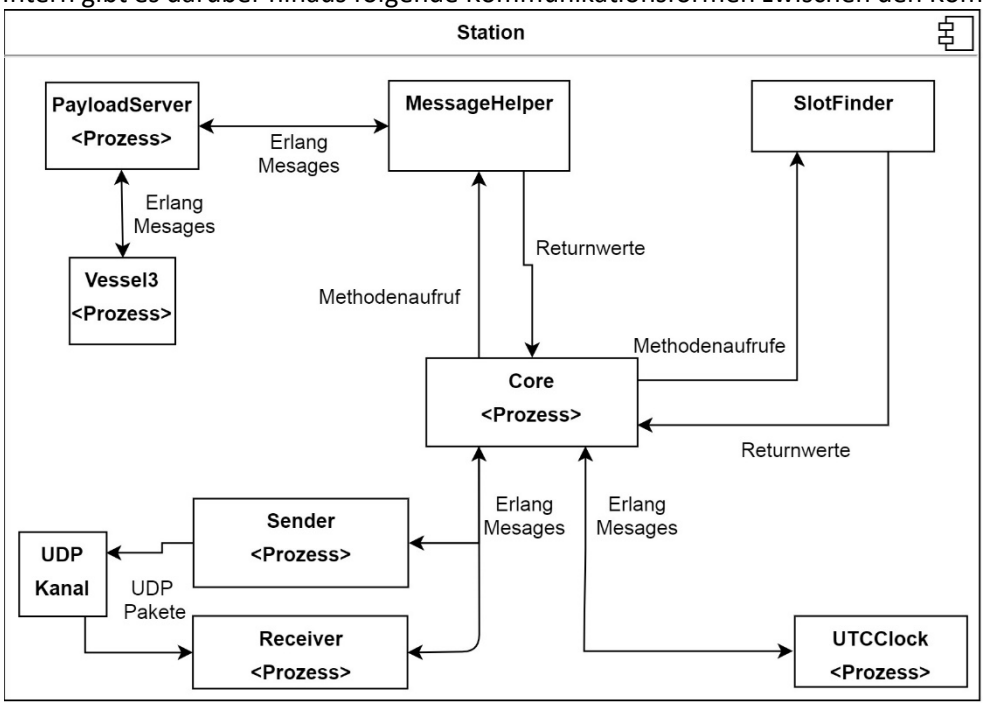
So ergeben sich folgende Diagramme (Interface Funktionen „send“ und „receive“ Beispielhaft):





Da hier nur Stationen senden und empfangen, und alle gleichwertig sind, ist es eine reine Peer-To-Peer Kommunikation zwischen den Stationen über UDP Multicast.

Intern gibt es darüber hinaus folgende Kommunikationsformen zwischen den Komponenten:



Was sind die verschiedenen Phasen?

- Initialisierung

  Die Initialisierung der Station (zum Beispiel der Uhr) ist sehr kurz (da wenig getan werden muss) und geht direkt in die Einstiegsphase über
- Einstiegsphase

  Jede Station hört auf alle Nachrichten mindestens einen ganzen Frame lang, die kollisionsfreien werden ausgewertet.

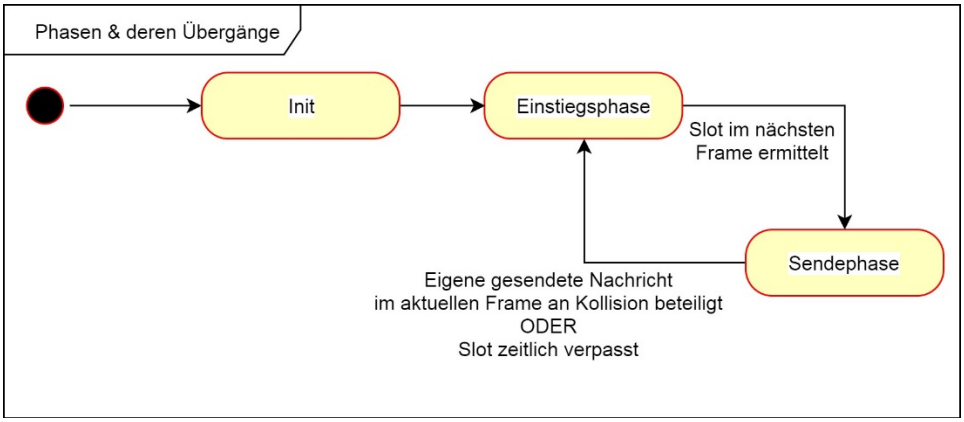
  Ist ein Slot gefunden so wird in die nächste Phase übergegangen.
- Sendephase

  Die Station sendet in dem vorhin gewählten Slot die eigene Nachricht.

  Ergibt sich keine Kollision so wird dieser Slot weiterhin verwendet und die Station sollte an keinen Kollisionen mehr beteiligt sein, wenn die Uhren gut genug synchronisiert wurden.

  Ist jedoch eine Kollision entstanden so geht die Station zurück in die Einstiegsphase.
  Oder wurde der gewählte Slot zeitlich verpasst, so geht es auch wieder zurück in die Einstiegsphase.

  Eine Terminierung durch das Programm gibt es nicht. Das heißt, läuft das Programm wie gesagt erstmal in einem Kollisionsfreien Zustand so wird unendlich weitergemacht.



Wie sieht der (Funk-) Kanal aus?

Der Kanal ist zeitlich in Frames eingeteilt, die wiederum in Slots eingeteilt sind.
Alle Nachrichten dürfen nur über diesen Kanal laufen und nur diese dürfen gelesen (also empfangen) werden. Der Kanal ist über einen bestimmten Socket anzusprechen.

Der Kanal selbst ist nicht zu implementieren, sondern spiegelt das Verhalten innerhalb der Stationen wieder. Jede Station hat also gewisser maßen eigene Frames und Slots (da die Stationen anfangs nicht synchron sind), der dann über die Laufzeit und der Synchronisation der Uhren zu einem einheitlichen Verständnis von einem Frame und deren Slots führt.

- Frames
  - Länge: 1 Sekunde
  - Hält 25 Slots
  - In einem Frame können N Nachrichten gesendet werden, wobei N >= 0 und N <= Anzahl der Stationen.
  - Fängt ab dem 01.01.1970 um 00:00:00 Uhr (UTC) an zu arbeiten (Jede Sekunde ein Frame).
    - Frame 1 beginnt: 01.01.1970 um 00:00:00
    - Frame 2 beginnt: 01.01.1970 um 00:00:01
    - ….
- Slots
  - Länge: 40 MS
  - In dem Slot werden N Nachrichten, wobei N >= 0 und N <= Anzahl der Stationen, geschickt.
- Socket
  - IP: 225.10.1.2, dies ist eine herkömmliche Multicast Adresse die hierbei noch zusätzlich als eine Broadcast Adresse gesehen werden könnte (Wenn darauf nur unsere Stationen hören).
  - Port: 15000 + Teamnummer (zum Beispiel Port 15003)
- Kollisionen

  Kollisionen entstehen, wenn mindestens 2 Stationen im selben Slot senden. Ziel ist es letztendlich einen Komplett Kollisionsfreien Nachrichtenverkehr zu haben, in dem in jedem Slot einmal gesendet wird, wenn Stationenanzahl = Slotanzahl in einem Frame.
  Zudem sendet jede Station maximal einmal in einem Frame. Ist ein kollisionsfreier Nachrichtenverkehr erst einmal passiert darf es zu keinen Kollisionen kommen.

  Unter Umständen kann es vorkommen, dass 2 Stationen eine Kollision verursachen obwohl sie nicht im gleichen Slot senden. Dies könnte passieren, wenn beide Stationen leicht verschiedene Uhren haben und Station 1 denkt noch im Slot 1 zu senden, merkt Station 2, dass die Nachricht im Slot 2 gesendet wurde (Weil die Uhr von Station 1 nach oder die Uhr von Station 2 vor geht). In diesem Fall wird dies auch als Kollision gesehen.

  Ist die Stationenanzahl < Slotanzahl, so sind dann dementsprechend viele Slots ohne Nachricht. Bei Stationenanzahl > Slotanzahl wird in jedem Frame zu Kollisionen kommen da ja nicht genügend Slots vorhanden sind.

Wie sehen die Nachrichten aus?

- TTL = 1
- Gesamtlänge zu jeder Zeit 34 Byte:

Byte 0:	Stationsklasse (A oder B)
Byte 1 – 10:	Stationsname (Bildet mit Byte 11-24 die gesamten Nutzdaten)
Byte 11 – 24:	Restliche Nutzdaten
Byte 25:	Slotnummer, in dem die Station im nächsten Frame senden wird.
Byte 26 - 33:	Zeitpunkt des sendens in MS, 8-Byte Integer, Big Endian, UTC.

Wie sieht die Station aus?

- Stationsname (Atom):
  - team <TEAMNUMMER>-<STATIONSNUMMER>
  - Beispiel für Team 06, Station 01: team06-01

- Wie bekommt man einen Slot und was geschieht danach?

  Ein Slot findet man, in dem man folgenden Ablauf beachtet:

  1. Es wird eine Liste mit allen möglichen Slotnummern erstellt (in dem Fall eine Liste mit den Nummern 1 bis 25).
  2. Von allen empfangenen kollisionsfreien Nachrichten werden die Slotnummern extrahiert
  3. Durch die Liste der extrahierten Slotnummern wird dann wie folgt iteriert:
     a. Nächste extrahiert Slotnummer aus Liste nehmen
     b. Diese Slotnummer aus Liste der verfügbaren Slotnummern löschen
     c. Mit der nächsten extrahierten Slotnummer fortfahren
  4. Am Ende bleiben dann in der Liste der verfügbaren Slotnummern 0 bis 25 Slotnummern übrig (Je nach Anzahl der Kollisionsfreien Nachrichten und der Anzahl der Stationen)
  5. Random wird nun eine dieser Nummern ausgewählt.

  Ergibt sich dann im nächsten Frame im ausgewählten Slot keine Kollision so wird der Slot behalten und kein neuer gesucht. Es darf nun zu keinen Kollisionen, in diesem Slot, mehr kommen.
  Kommt es dagegen zu einer Kollision muss wieder einen **kompletten** Frame zugehört werden um in den dann kollisionfreien Nachrichten wieder eine freie Slotnummer zu finden.

- Wie sieht die Datenquelle aus?

  Ist ein vorgegebenes Programm. Dieses wird über Standard Out (die Konsole) periodisch 24-Byte Nutzdatenpakete generieren (siehe Byte 1-24 der Nachrichten).
  Verwendet wird dann pro Station nur das aktuellste generierte Paket.
  Gestartet wird das bereitgestellte Programm über „java vessel3.Vessel <Teamnummer> <Praktikumsnummer>“. Der Output des Programms wird über die Pipe an den PayloadServer genannt. Der empfängt alle Outputs aber macht vorerst nichts damit. Bis eine Nachricht kommt die dazu führt, dass der PayloadServer an den Absender den aktuellsten (nächsten) Output zurücksendet.

- Wie sieht die Datensenke aus?

  Die Datensenke ist vorerst über den minimalsten weg zu realisieren. In diesem Fall stellt das Standard Out (in die Konsole) dar.

- Wie sieht die Interne Uhr aus?

  Sie zeigt die aktuelle Zeit (für die Station), ausgehend von UTC, an.
  Sie wird konstant synchronisiert mit den Uhren der anderen Stationen. Dies geschieht dann über die empfangenen kollisionsfreien Nachrichten von Typ A Stationen.
  Beim Empfang der Nachrichten wird vermerkt, wann die Nachricht empfangen wurde. Dies ist wichtig um möglichst genau die eigene Abweichung zu errechnen und sich dementsprechend zu synchronisieren. Dabei gehen wir davon aus, dass die Transportzeit 0 MS beträgt.

  Typ A und Typ B Stationen und deren Uhren unterscheiden sich nur darin, dass Typ A Stationen zu Beginn mit einer Abweichung gestartet werden.

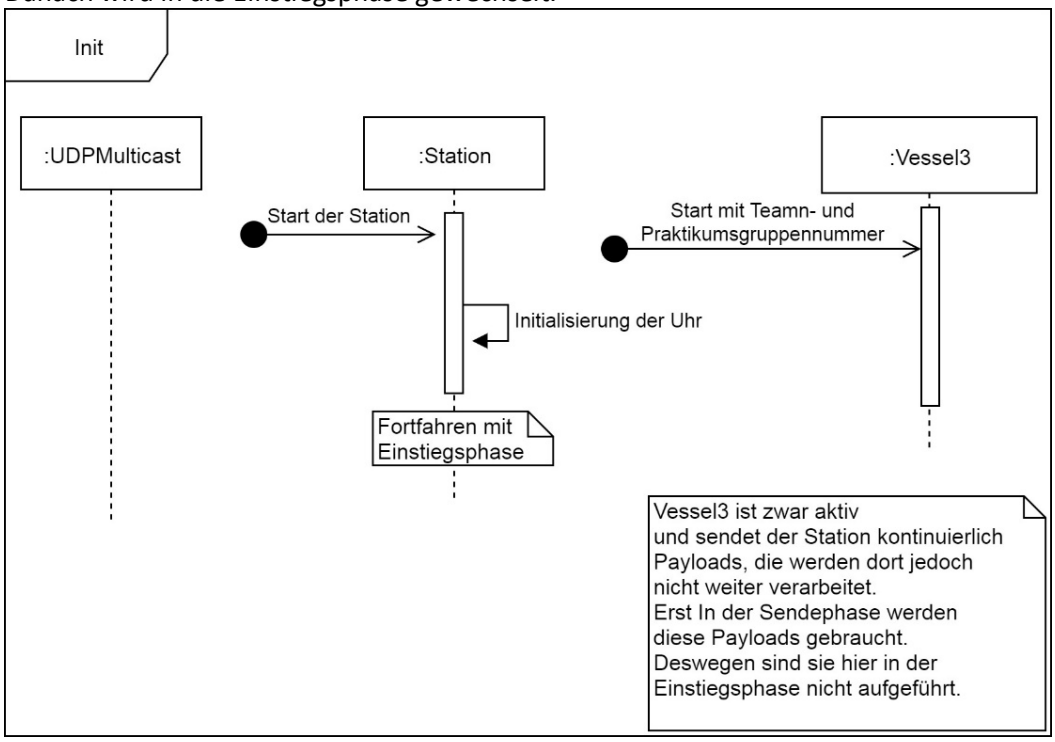
  Intern wird es so gehandelt, dass eine Uhr einen Offset besitzt den man zu Beginn (Initialisierung) pro Station festlegen kann. Dieser Offset mit der aktuellen Systemzeit (zum Beispiel mit vsutil:getUTC/0) und dem Startzeitpunkt (der Uhr Erstellung) ergibt dann die Zeit der Station.

  Eine Uhr wird wie folgt synchronisiert (Alle Zeitstempel in MS und UTC):

  1. Alle Nachrichten bekommen beim Empfang einen Zeitstempel von der Stationsuhr
  2. Alle kollisionsfreien Nachrichten von Stationstyp A werden dann weiterverarbeitet
     a. Es wird pro empfangener Nachricht die Zeitdifferenz ausgerechnet (Empfangszeit – Sendezeit)
     b. Dann wird aus allen Differenzen ein arithmetisches Mittel gebildet
     c. Dieses Mittel wird dann auf unseren Uhren Offset addiert.

- Wie verläuft die Initialisierung?

Die Station wird mit einer Uhr Offset (in MS) gestartet. Intern speichert sich die Uhr zusätzlich in welchem Moment die Uhr erstellt wurde (zum Beispiel mit vsutil:getUTC/0).
Zu dem werden Sender / Receiver / PayloadServer (Somit auch Vessel3) gestartet.
Danach wird in die Einstiegsphase gewechselt.



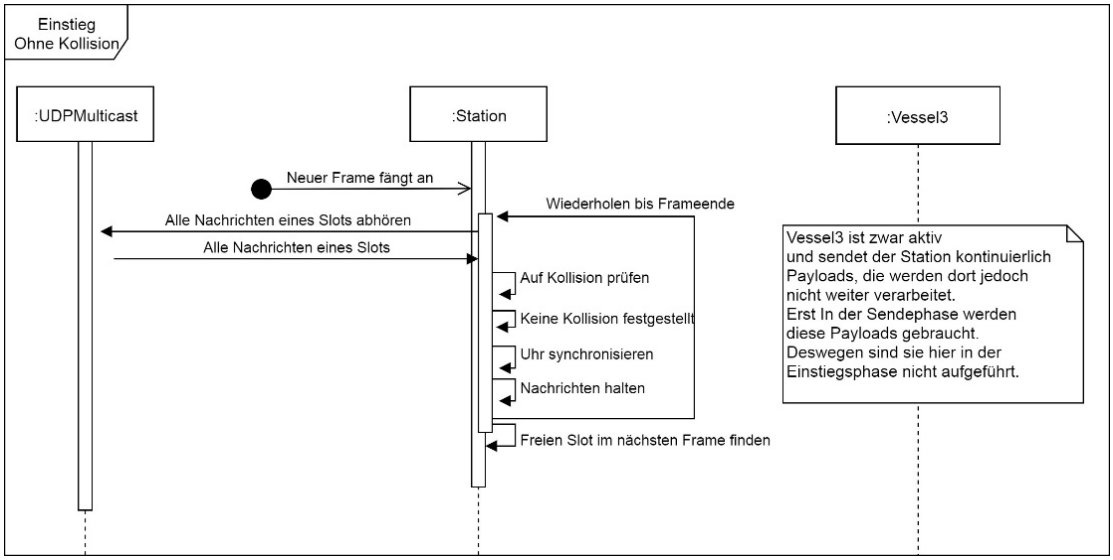
- Wie verläuft die Einstiegsphase?

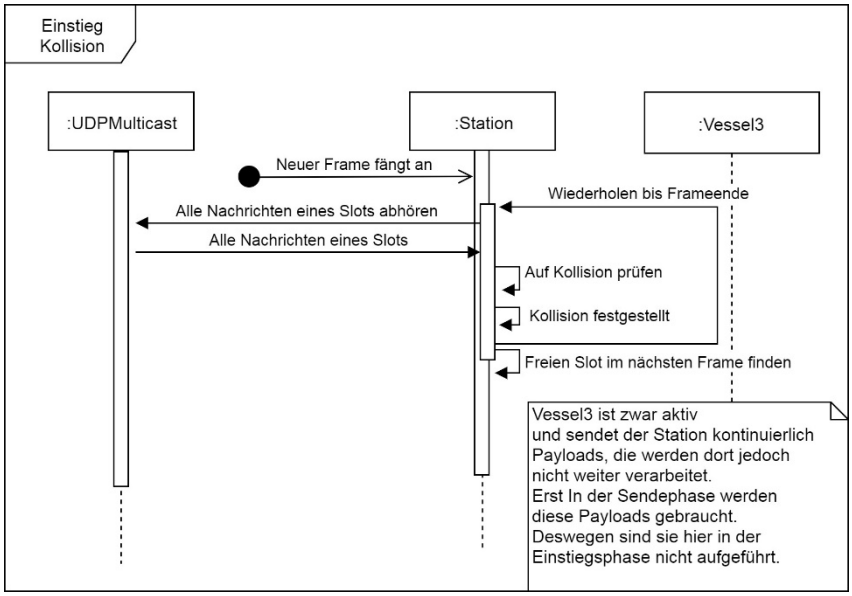
  Sie beginnt damit, dass ein neuer Frame beginnt und die Station auf alle Nachrichten aus dem Funkkanal hört. Die kollisionsfreien Nachrichten werden weiterverwendet um die eigene Uhr zu synchronisieren und um einen freien Slot im nächsten Frame zu senden. Ist ein Slot gefunden ist dies der Übergang zur Sendephase.

  Eine Kollision wird wie folgt festgestellt:

  1. Station hört auf Funkkanal
  2. Station bekommt N Nachrichten in einem Slot
     a. N = 0, Keine Nachrichten erhalten, keine Kollision
     b. N = 1, Eine Nachricht erhalten, keine Kollision
        i. Nachricht wird weiterverwendet: Uhren Synchronisation und Slot Findung
     c. N >= 2, mindestens 2 Nachrichten erhalten, Kollision!
        i. Diese Nachrichten werden dann nicht mehr weiter behandelt. Also implizit vergessen.

  So ergeben sich die folgenden Diagramme:





- Wie verläuft die Sendephase?

  Auch in der Sendephase hört die Station noch auf jede eingehende Nachricht. Man prüft darauf ob eine Kollision entstand und ob eine Nachricht der eigenen Station daran beteiligt war. Ist dies der Fall so ist dies der Übergang zur Einstiegsphase. Zudem wird (wie in der Einstiegsphase) die Uhr mit den kollisionsfreien Nachrichten synchronisiert.

  Die Verarbeitung der eingehenden Nachrichten und die Vorbereitung / Senden neuen Nachricht läuft dabei nebenläufig. Deswegen kann es dazu kommen, dass Die Uhrzeit ggf. so synchronisiert wird das sie einen kleinen Sprung in die Zukunft macht und dies dazu führt, dass die Slotzeit zum Senden verpasst wird. Dies ist dann auch ein Übergang zur Einstiegsphase.

  Der Sender sendet zu einem bestimmten Zeitpunkt ein Nachrichtenpaket. Dies geschieht genau ein- oder keinmal in einem Frame.

  Der Ablaut ist wie folgt:

  Empfangsprozess:

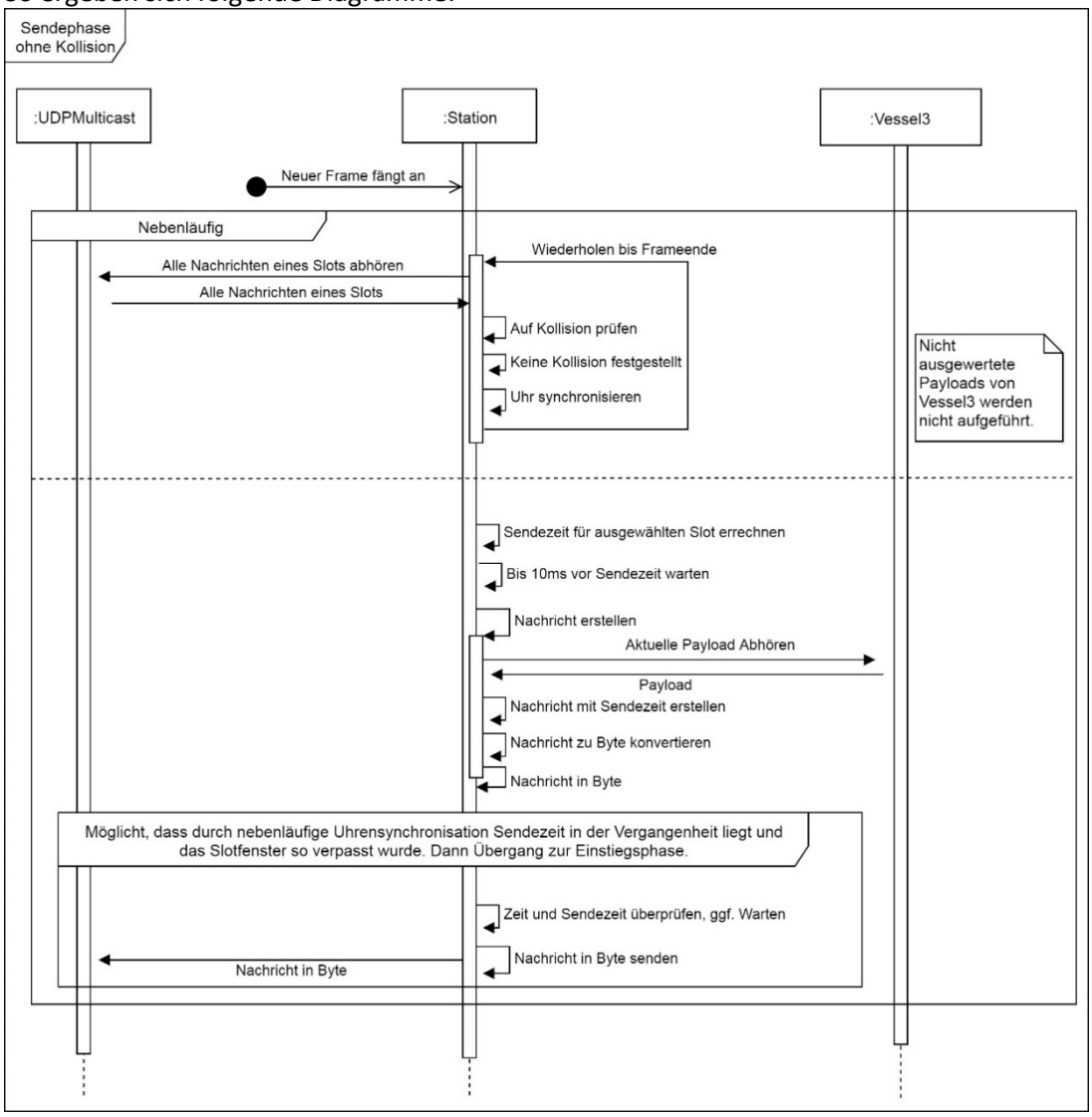
  1. Verarbeitung und Kollisionscheck wie in der Einstiegsphase
  2. Mit Zusatz: Wenn eigene Nachricht an Kollision beteiligt:
     a. Übergang zur Einstiegsphase

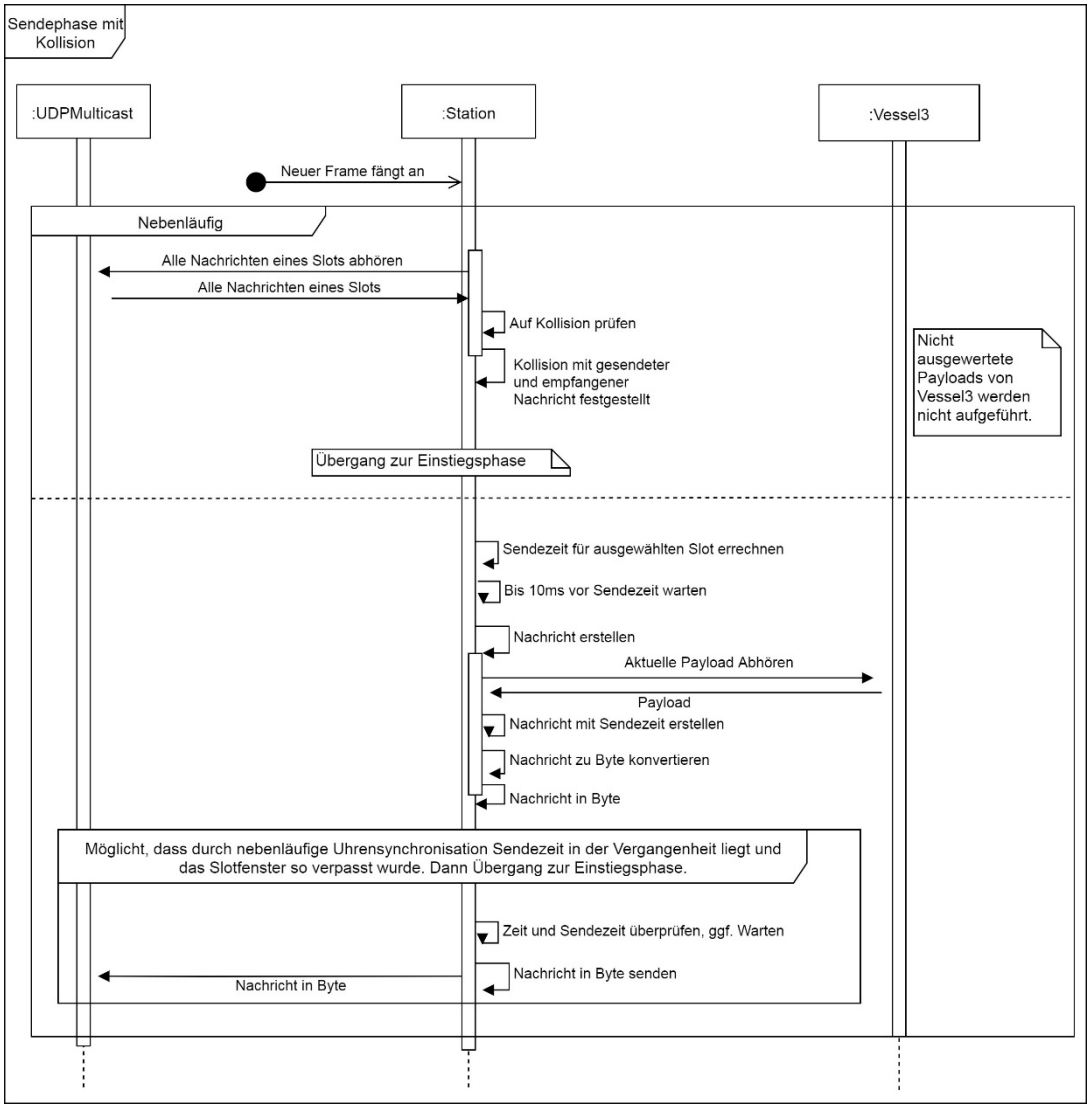
  Sendeprozess:

  1. Es wird für eine Nachricht ermittelt, wann gesendet werden muss

    a. Dies wird mit Hilfe der Slotnummer und der eigenen UTC Clock errechnet, da jede Sekunde ein neuer Frame beginnt und ein Slot 40 MS lang ist.
2. Es geschieht nichts bis die Uhr dem Sender ein Signal gibt, dass die Nachricht in 10 MS raus muss.
3. Dann wird die Nachricht vorbereitet:
    a. Die Daten:
        i. Typ: Bekannt
        ii. Payload: Wird von Datenquelle geholt
        iii. Slotnummer: Bekannt
        iv. Zeit des Sendens: vorhin errechnet
    b. Diese Nachricht wird dann zu Byte konvertiert
4. Es wird gewartet bis ein Signal von der Uhr kommt, dass die Nachricht jetzt raus muss.
    a. Da sich die Uhr in der ganzen Zeit des Ablaufs ggf. neu synchronisiert könnte sie auch soweit vorgestellt werden, dass der Slot verpasst wird, dann wird nicht gesendet.
    b. Der Sender sendet die vorbereitete Nachricht an die spezifizierte Multicast IP Adresse.

So ergeben sich folgende Diagramme:




Sendephase mit Kollision
:UDPMulticast
:Station
:Vessel3
Neuer Frame fängt an
Nebenläufig
Alle Nachrichten eines Slots abhören
Alle Nachrichten eines Slots
Auf Kollision prüfen
Kollision mit gesendeter und empfangener Nachricht festgestellt
Nicht ausgewertete Payloads von Vessel3 werden nicht aufgeführt.
Übergang zur Einstiegsphase
Sendezeit für ausgewählten Slot errechnen
Bis 10ms vor Sendezeit warten
Nachricht erstellen
Aktuelle Payload Abhören
Payload
Nachricht mit Sendezeit erstellen
Nachricht zu Byte konvertieren
Nachricht in Byte
Möglicht, dass durch nebenläufige Uhrensynchronisation Sendezeit in der Vergangenheit liegt und das Slotfenster so verpasst wurde. Dann Übergang zur Einstiegsphase.
Zeit und Sendezeit überprüfen, ggf. Warten
Nachricht in Byte senden
Nachricht in Byte

Was passiert in den ersten Frames (Beispiel Ablauf)?

Zum Start wird erst einmal jede Station gestartet (Initialisiert).
Danach geht sie direkt zur Einstiegsphase über. Da die Station direkt im Init jedoch schon im ersten befindet, und die ersten Millisekunden mit dem Init verbracht werden muss der 2. Frame auch komplett abgehört werden.
Im 3. Frame wird dann im gefundenen Slot eine Nachricht gesendet.

Alle starten (haben zum Beispiel ihr Init) im selben Frame, wenn es keine Zeitdifferenzen gäbe.
Wenn es diese Differenz gibt ist es unterschiedlich in welchem Frame die Stationen starten.
Abstrakt passiert jedoch das selbe. Da es hierfür egal ist in welchem Frame (0/1/2/3/…) die Station intern meint zu sein. Es muss immer min. einen kompletten Frame zugehört werden bevor gesendet werden kann.

Deswegen gilt der folgende Ablauf im Abstrakteren Sinne generell, ganz konkret aber nur für den Fall ohne Zeitdifferenz: