Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Referat

Daniel Schruhl

Entwurf und Implementierung einer STDMA Station

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniel Schruhl

Entwurf und Implementierung einer STDMA Station

Referat eingereicht im Rahmen der Referat eingereicht im Rahmen der Vorlesung Verteilte Systeme

im Studiengang Angewandte Informatik (AI)
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. C. Klauck

Eingereicht am: 31. Mai 2017

Daniel Schruhl

Thema der Arbeit
Entwurf und Implementierung einer STDMA Station

Stichworte
STDMA, Verteilte Systeme, Clojure

Kurzzusammenfassung
Umsetzung einer STDMA Station auf IP-Multicast Socket Basis

Daniel Schruhl

Title of the paper
Design and implementation of an STDMA station

Keywords
STDMA, Distributed Systems, Clojure

Abstract
Implementation of an STDMA Station via IP-Multicast Sockets

Inhaltsverzeichnis

1 Einführung und Ziele **1**
- 1.1 Randbedingungen . . . 1
- 1.2 Kontextabgrezung . . . 3

2 Gesamtsystem **4**
- 2.1 Architekturüberblick . . . 4
- 2.2 Konfigurationsparameter . . . 5
- 2.3 Benutzungsschnittstellen . . . 5

3 Subsysteme und Komponenten **9**
- 3.1 Datagram . . . 9
  - 3.1.1 Aufgabe und Verantwortung . . . 9
  - 3.1.2 Schnittstelle . . . 9
  - 3.1.3 Entwurfsentscheidungen . . . 10
- 3.2 Connector . . . 10
  - 3.2.1 Aufgabe und Verantwortung . . . 10
  - 3.2.2 Schnittstelle . . . 10
  - 3.2.3 Entwurfsentscheidungen . . . 11
  - 3.2.4 Konfigurationsparameter . . . 11
- 3.3 Payload-Source . . . 12
  - 3.3.1 Aufgabe und Verantwortung . . . 12
  - 3.3.2 Schnittstelle . . . 12
  - 3.3.3 Entwurfsentscheidungen . . . 12
  - 3.3.4 Konfigurationsparameter . . . 12
- 3.4 Message-Writer . . . 12
  - 3.4.1 Aufgabe und Verantwortung . . . 12
  - 3.4.2 Schnittstelle . . . 13
  - 3.4.3 Entwurfsentscheidungen . . . 13
  - 3.4.4 Konfigurationsparameter . . . 13
- 3.5 Clock . . . 13
  - 3.5.1 Aufgabe und Verantwortung . . . 13
  - 3.5.2 Schnittstelle . . . 13
  - 3.5.3 Entwurfsentscheidungen . . . 14
  - 3.5.4 Konfigurationsparameter . . . 15
- 3.6 Station . . . 15
  - 3.6.1 Aufgabe und Verantwortung . . . 15

3.6.2 Schnittstelle 15
3.6.3 Entwurfsentscheidungen 15
3.6.4 Konfigurationsparameter 16

4 Kollisionsfreier Ablauf **17**

Erklärung zur schriftlichen Ausarbeitung des Referates **18**

1 Einführung und Ziele

In verteilten Systemen werden oft Nachrichten unter verschiedenen Teilnehmern eines Netzes ausgetauscht. Dabei sind diese Teilnehmer oft nur durch ein Medium miteinander verbunden. Dieses eine Medium muss dann mehrere Teilnehmer unterstützen können.

Um dieses Problem zu lösen soll ein Softwareprodukt erstellt werden, das eine Station darstellt, die auf einem Medium Nachrichten empfangen und senden kann.

Dabei sollen mehrere Stationen auf einem Medium Nachrichten verschicken und senden. Das geschieht, indem das Medium in feste Frames aufgeteilt wird und jeder Frame eine feste Anzahl an Slots hat, in dem aus allen teilnehmenden Stationen immer nur eine in ihrem zugeteilten Slot Senden darf. Das Medium wird hierbei im Betracht der Zeit aufgeteilt. Diese Aufteilung (Multiplexing) wird auch time-division multiple access (TDMA) genannt. In diesem Fall sollen die Stationen selber untereinander ihre Slots zum Senden verwalten und untereinander gleichmäßig aufteilen. Das wird auch Self-organized time-division multiple access (STDMA) genannt.

1.1 Randbedingungen

Das zu verwendende Medium ist ein Socket, der per IP Multicast Nachrichtenpakete (Datagrams) an mehrere Klienten austeilen kann. Die TTL der Multicast-Pakete ist auf 1 zu setzen, um unnötige Netzlasten und Netzstörungen zu vermeiden.

Die Frames haben eine Länge von einer Sekunde. Ein Frame hat 25 Slots. Diese sind von 1 bis 25 nummeriert. Die Frames sind in Sekunden seit dem 1.1.1970, 00:00 Uhr nummeriert.

Jede Station darf nur einmal pro Frame senden. Das muss in der Mitte ihres zugeteilten Slots passieren.

Die Vergabe der Slots soll ohne zentrale Vergabeinstanz geschehen. Das geschieht, indem jedes Nachrichtenpaket ein Datenfeld enthält, in das die sendende Station die Nummer ihres Slots

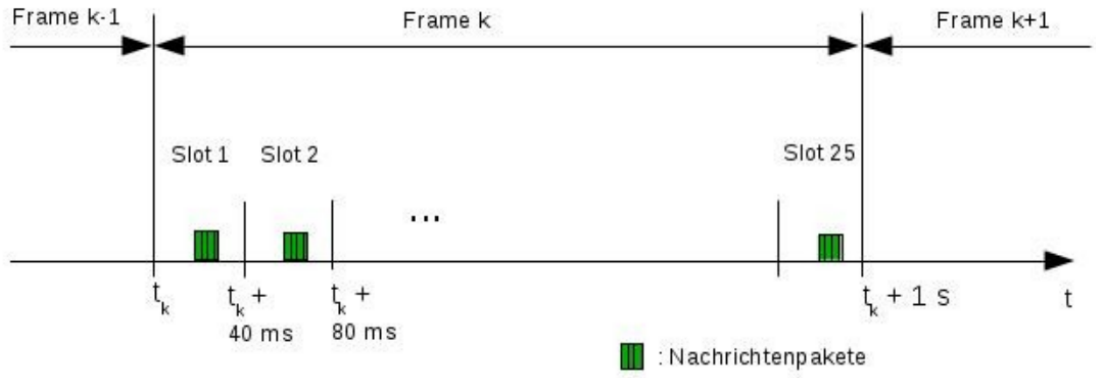

Abbildung 1.1: Aufteilung des Mediums in Frames und Slots, Quelle: **?**

einträgt, die sie für den nächsten Frame zum Senden verwendet. Das bedeutet also, dass durch das Lesen der Nachrichtenpakete in einem Frame bestimmt werden kann, welche Slots im nächsten Frame frei sind.

Ein Nachrichtenpaket hat folgende Struktur:

- Byte 0: Stationsklasse (’A’ oder ’B’)
- Byte 1 – 24: Nutzdaten. (Darin Byte 1 – 10: Name der sendenden Station.)
- Byte 25: Nummer des Slots, in dem die Station im nächsten Frame senden wird.
- Byte 26 – 33: Zeitpunkt, zu dem dieses Paket gesendet wurde. Einheit: Millisekunden seit dem 1.1.1970 als 8-Byte Integer, Big Endian.

Gesamtlänge: 34 Bytes

Stationen haben dabei eine Unterteilung in Klasse A und B. Diese Klassen werden für eine Synchronisierung der Zeit verwendet. Generell wird die UTC Zeit verwendet. Die Uhr einer Station kann einen anfänglichen Offset haben. Die Stationen synchronisieren dann untereinander im Laufe der Teilnahme am Netz ihre Uhren untereinander, so dass der Offset sich verändern kann.

Klasse A Uhren gelten als hinreichend genau und als Referenz zur Synchronisation. Klasse B und Klasse A Stationen können dann anhand anderer empfangen Nachrichtenpakete von Klasse A Stationen ihren Offset verändern. Dabei wird die Sendezeit und die Klasse innerhalb eines Nachrichtenpaketes verwendet.

Bei Kollisionen sollen die betroffenen Nachrichtenpakete als nicht empfangen betrachtet werden. Ein kollisionsfreier Betrieb mit maximal 25 Stationen muss nach endlicher Zeit erreicht werden und darf nicht wieder verlassen werden.

Das Produkt soll über die Kommandozeile gestartet werden können. Dabei müssen folgende Parameter mit übergeben werden können:

- Interfacename des Kommunikationsendpunktes
- Adresse der Multicast Gruppe, Klasse D IP-Adresse
- Port des Sockets
- Stationsklasse, A oder B
- Anfänglicher UTC Offset

Als Datenquelle für die Nutzdaten soll das STDIN des Produktes verwendet werden, um der Station neue Nutzdaten zukommen zu lassen.

1.2 Kontextabgrezung

Das Produkt muss auch in einer ssh-Sitzung auf einem anderen Rechner gestartet werden können. Es muss auf Rechnern mit Linux Betriebssystem lauffähig sein.

2 Gesamtsystem

Das Produkt ist in Clojure (v1.9) umgesetzt worden. Clojure ist eine funktionale Programmiersprache, die auf der JVM läuft. Dabei ist eine interoperabilität mit Java möglich. Das erleichtert den Umgang mit Sockets, da der Umgang mit diesen in Java gut dokumentiert und verwendbar ist. Außerdem hat Clojure im Vergleich mit Erlang einige andere Vorteile wie lazily evaluated Sequences oder eine bessere Testumgebung.

Funktionale Programmiersprachen haben den Vorteil in verteilten Systemen, das die Ergebnisse von Funktionen bei gleichen Parametern immer gleich sind. Diese Immutability von Daten hat den Vorteil, das nebenläufige Prozesse dadurch keine Datensynchronisation machen müssen. Dadurch wird eine erhöhte Threadsicherheit erreicht.

2.1 Architekturüberblick

Das System ist in drei Pakete aufgeteilt, die jeweils aus verschiedenen Komponenten bestehen. Die Pakete sind die Business Logik (Basis Paket), die Verbindungs Logik (Network Paket) und die Daten Verbindungs Logik (Data-Links Paket).

Das Paket mit der Business Logik enthält die Stations Komponente, die Uhren Komponente und den Haupteinstiegspunkt des Systems, das alle Komponenten zusammen baut. Hier sind die fachlichen Logiken, die das Verhalten der Station beeinflussen definiert.

Das Paket mit der Verbindungslogik enthält die Verbindungs Komponente und die Datagram Komponente. Hier werden alle verbindungsrelevanten Informationen verarbeitet. Dazu gehört der Aufbau und der Abbau und das Versenden und Verschicken von Nachrichtenpaketen auf Netzebene (Datagramme).

Das Daten Verbinungs Paket beinhaltet die Payload Source Komponente und die Message Writer Komponente. Dieses Paket ist für die Datensenke und Datenquelle zuständig. Sie Laufen Nebenläufig zu den anderen Komponenten, damit Sie diese nicht blockieren und so die fachliche Aufgabe des Gesamtsystems isoliert davon arbeiten kann.

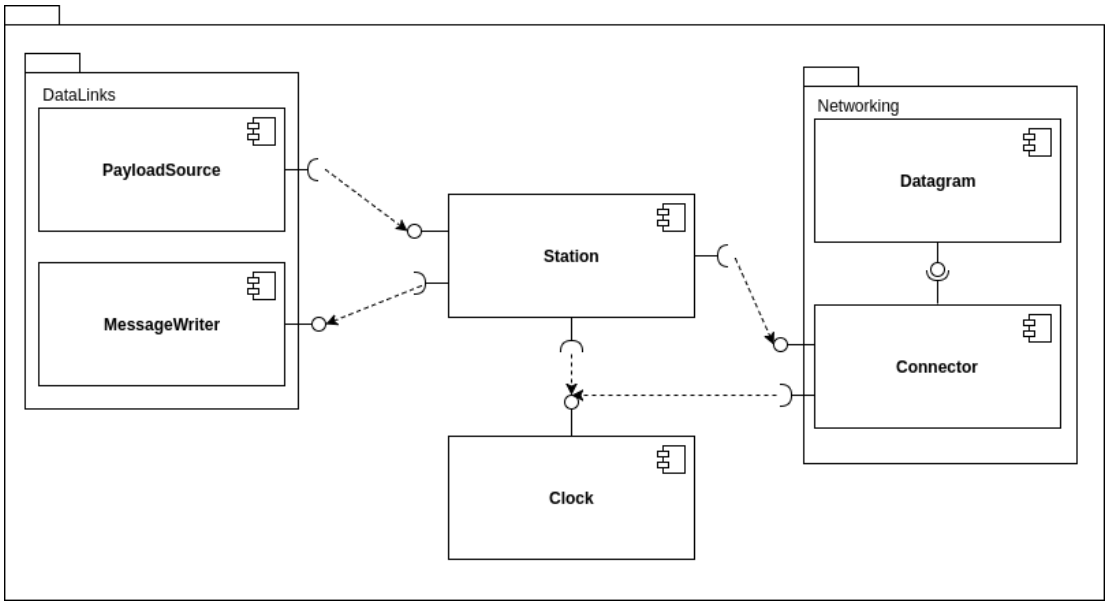

Abbildung 2.1: Komponentendiagramm des Gesamtsystems

2.2 Konfigurationsparameter

Es existiert eine Konfigurationsdatei (./resources/default.edn), die folgende Parameter bestimmt:

- Anzahl der slots
- Länge eines Frames in ms
- Länge eines Nachrichtenpaketes
- Länge der Nutznachricht
- Datei für die Datensenke

2.3 Benutzungsschnittstellen

Das System wird als .jar über die Kommandozeile gestartet mit folgenden Parameter (in Reihenfolge):

- Interfacename
- Multicastadresse
- Port

- Stationsklasse
- Initialer UTC Offset

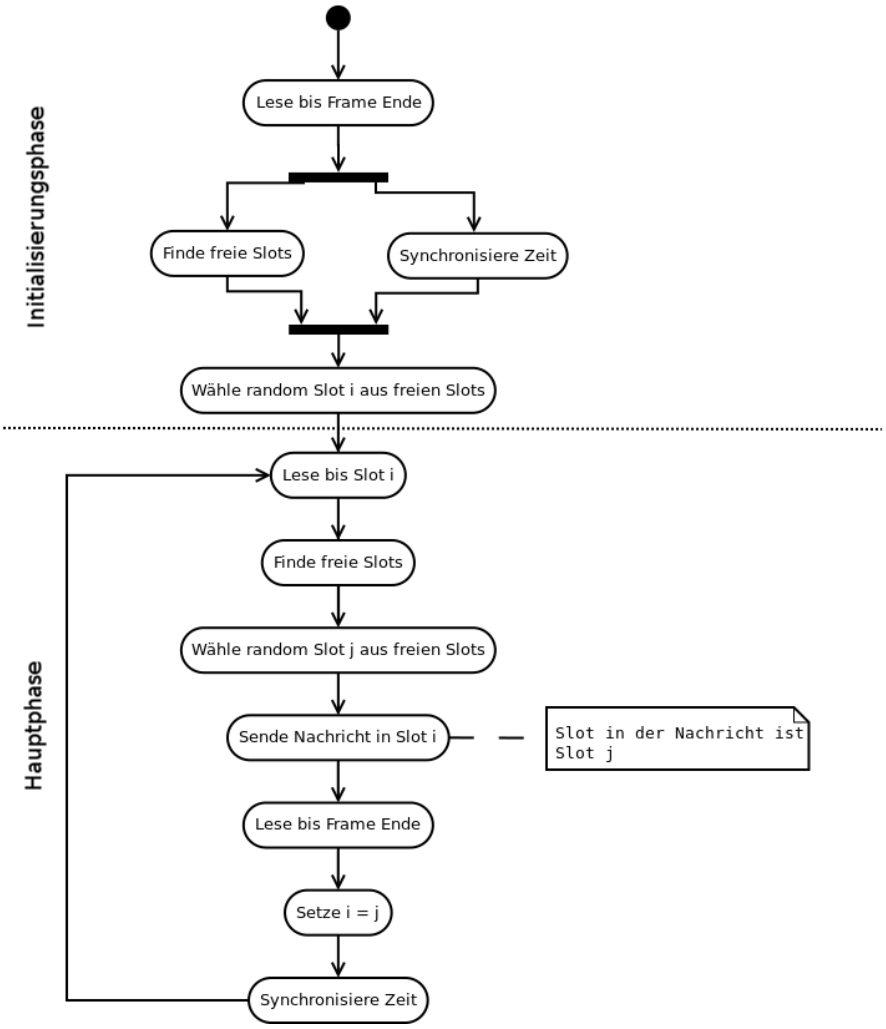

Abbildung 2.2: Aktivitätendiagramm der Nachrichtenbehandlung im Gesamtsystem

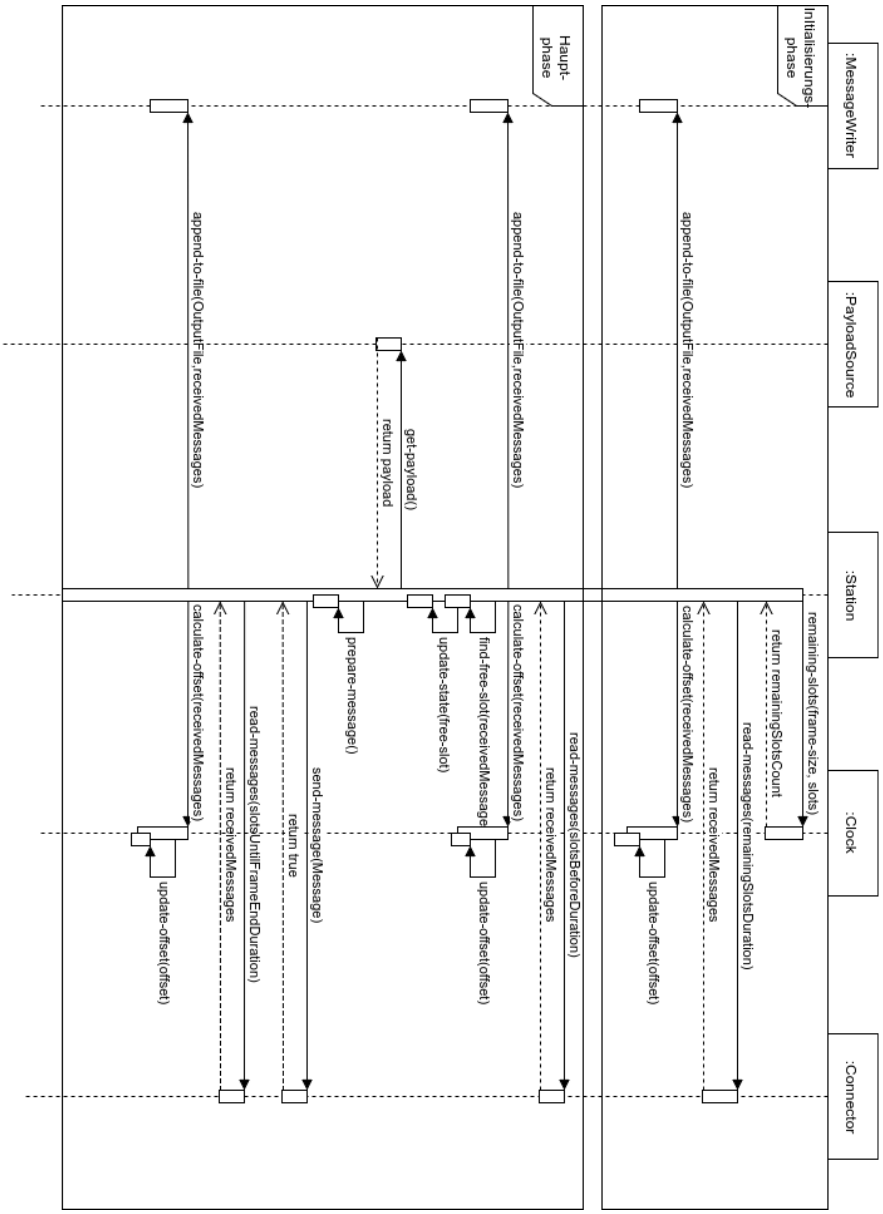

Abbildung 2.3: Sequenzdiagramm der Nachrichtenbehandlung im Gesamtsystem

3 Subsysteme und Komponenten

3.1 Datagram

3.1.1 Aufgabe und Verantwortung

Das Datagram Modul ist dafür verantwortlich aus vorhanden Nachrichten Byte Pakete zu erstellen und diese wieder in Nachrichten zurück zu konvertieren.

3.1.2 Schnittstelle

```
; Prueft ob uebergebene Bytes nicht leer sind
Bytes -> Boolean
byte-array-empty?(bytes-array)

; Konvertiert uebergebene Bytes in ein Nachricht
Bytes -> Nachricht
datagram->message(bytes-array)

; Kodiert uebergebene Nachricht in ein Bytes
Nachricht -> Bytes
message->datagram(message)
```

Eine Nachricht ist eine Map folgender Gestalt:

```
{:station-class   String
 :station-name    String
 :payload-content String
 :payload         String
 :slot            Nummer
 :send-time       Nummer}
```

3.1.3 Entwurfsentscheidungen

Nutznachrichten werden in ihrer Zeichenketten Form als UTF-8 konvertiert behandelt. Die Stationsklasse wird in Byte der jeweiligen String Repräsentation des Klassennamen konvertiert ('A' oder 'B').

Die Zeitstempel in den Nachrichten werden als Big Endian Bytes behandelt (eventuelles Zero Padding ist links).

3.2 Connector

3.2.1 Aufgabe und Verantwortung

Der Connector abstrahiert die Netzwerkverbindung, die per Multicast Sockets umgesetzt wurde. Über den Connector werden Nachrichten verschickt und empfangen. Das Verschicken und Empfangen ist mit einer Kollisionserkennung ausgestattet, so dass falsche Nachrichtenpakete verworfen werden.

Der Connector versendet Nachrichten und konvertiert diese in Nachrichtenpakete (Bytes). Beim Lesen werden Nachrichtenpakete (Bytes) eingelesen und als konvertierte Nachrichten zurückgegeben.

3.2.2 Schnittstelle

```
; Startet den Connector und verbindet den Socket
Map -> Komponente
new-connector(config)

; Sendet eine Nachricht, wenn das Senden Kollisionsfrei ist
(Komponente, Nachricht) -> Boolean
send-message-collision-safe?(connector, message)

; Liest Nachrichten bis zum Ablauf des Timeouts
(Komponente, Nummer) -> Bytes
read-message-with-collision-detection(connector, timeout)
```

Eine Komponente ist eine Map, die den State und die Konfiguration der Komponente enthalten in jeweils wieder einer Map.

```
Map -> Map
{:config             Map
 :socket-connection  Map}
```

3.2.3 Entwurfsentscheidungen

Der Connector hat einen State, der seine Socketverbindung und Informationen zu dieser beinhaltet. Dazu gehört die IP-Adresse, der Port und das Interface.

Der Connector verbindet seinen Socket beim Start und tritt der angegebenen Multicast Gruppe bei. Beim Herunterfahren, baut er die Verbindung wieder ab.

Der Sendevorgang im Connector liest vor dem Zeitpunkt des Sendens (20 ms nach Aufruf) Nachrichten, um zu überprüfen, ob der geplante Sendevorgang überhaupt gültig ist. Wenn Nachrichten vor dem Senden empfangen wurden, dann wird das eigentlich nun folgende Senden eine Kollision hervorrufen, da dann mehrere Nachrichtenpakete in einem Slot versendet wurden. Wenn vor dem Senden Nachrichten empfangen wurden, dann wird die eigentliche zu sendende Nachricht verworfen und nicht gesendet. Es wird bis zum Ende des Slots gewartet. Die Funktion signalisiert dann, dass nicht gesendet wurde. Falls vorher keine Nachricht empfangen wurde, wird die Nachricht mit dem nun aktuellen Zeitstempel über den Socket aus dem State versendet. Danach wird gewartet, bis der Slot abgelaufen ist. Die Funktion signalisiert ein erfolgreiches Senden.

Der Lesevorgang prüft in einem bestimmten Zeitfenster (timeout) jede ms, ob eine Nachricht auf dem Socket anliegt. Falls eine Nachricht anliegt, wird diese in die Liste der empfangenen Nachrichten angefügt. Die daraus resultierende Liste enthält alle empfangenen Nachrichten. Wenn diese Liste genau ein Element enthält, wird dieses Element als Nachricht zurück gegeben. Ansonsten wird das leere Element (nil) zurückgegeben. Das ist der Fall, wenn eine Kollision während des Lesens erkannt wurde.

Die Lese- und Sendevorgänge sind so aufgebaut, dass diese über eine anzugebende Laufzeit laufen. Alle Zeitabhängigen Funktionen wie der aktuelle Zeitstempel, die Zeit bis zum Ablaufen eines Slots oder die Mitte des Slots wird bei der Clock-Komponente abgerufen.

Die Konvertierung der Nachrichten in Bytes zum Versend und die Konvertierung der Bytes in Nachrichten beim Empfang geschieht durch das Datagram Modul.

3.2.4 Konfigurationsparameter

- Interfacename

- Multicastadresse
- Port

3.3 Payload-Source

3.3.1 Aufgabe und Verantwortung

Die Payloud-Source Komponente ist für das Bereitstellen der zu verwendenden Nutznachrichten zuständig. Diese kommen aus einer externen Datenquelle.

3.3.2 Schnittstelle

```
; Startet die Komponente mit angegebenen Kanal
Kanal -> Komponente
new-payload-source(out-chan)
```

3.3.3 Entwurfsentscheidungen

Die Payload-Source Komponente ist ein eigenständiger Prozess. Die Nutzdaten (24 Bytes) werden in diesem Prozess ständig aus dem STDIN gelesen und auf einen Kanal gelegt. Dieser Kanal ist mit der Stations Komponente verbunden. Dadurch werden auf den Kanal alle empfangenen Nutzdaten aus der externen Datenquelle auf diesem Kanal nebenläufig hinzugefügt. Wenn der Kanal eine Größe größer Null hat, dann dient dieser gleichzeitig als Nutzdatenbuffer und hält immer eine bestimmte Anzahl an Nutzdaten bereit.

3.3.4 Konfigurationsparameter

- Kanalgröße

3.4 Message-Writer

3.4.1 Aufgabe und Verantwortung

Der Message-Writer dafür zuständig empfangene Nachrichten in einer Datei bereit zu stellen.

3.4.2 Schnittstelle

```
; Startet die Komponente mit angegebenen Kanal
Kanal -> Komponente
new-message-writer(in-chan)
```

3.4.3 Entwurfsentscheidungen

Der Message-Writer ist ein eigener Prozess, der kontinuierlich eine Nachricht (falls vorhanden) aus dem Kanal herausnimmt und die Nutzdaten aus dieser an eine Datei als neue Zeile anfügt. Der Kanal aus dem entnommen wird kann eine feste Größe haben. Eine Größe größer Null veranlasst, dass der Kanal als Buffer von gelesenen Nachrichten fungiert.

3.4.4 Konfigurationsparameter

- Kanalgröße
- Ausgabedatei

3.5 Clock

3.5.1 Aufgabe und Verantwortung

Die Clock ist der zentrale Anfragepunkt für die aktuellen Zeit und zeitabängige Werte. Dazu gehören die restliche Zeit bis zu einem Slot- oder Frameende, der aktuelle Slot in Abhängigkeit von der Zeit, die restlichen Slots in Abhängigkeit von der Zeit und die Zeit bis zur Mitte eines Slots. Die aktuelle Zeit hat eine einstellbare Abweichung.

3.5.2 Schnittstelle

```
; Enthaelt den State mit der Abweichung
() -> State
offset

; Gibt die aktuelle Zeit + Abweichung zurueck
() -> Nummer
current-time
```

```
8
9  ; Gibt den aktuellen Frame zurueck, abhaengig der Framelaenge
10 Nummer -> Nummer
11 current-frame(frame-size)
12
13 ; Gibt den aktuellen Slot zurueck, abhaengig der Framelaenge und
14 ; Slotanzahl
15 (Nummer, Nummer) -> Nummer
16 current-slot(frame-size, slot-count)
17
18 ; Gibt Anzahl der restlichen Slots im Frame zurueck
19 (Nummer, Nummer) -> Nummer
20 remaining-slots(frame-size, slot-count)
21
22 ; Gibt die restlichen ms pro festen Zeitabstand zurueck
23 Nummer -> Nummer
24 remaining-time-until-end(duration)
25
26 ; Blockiert den aufrufenden Prozess bis ein Slot abgelaufen ist
27 Nummer -> ()
28 wait-until-slot-end(slot-duration)
29
30 ; Zeit bis die Mitte eines Slots erreicht ist
31 Nummer -> Nummer
32 ms-until-slot-middle(slot-duration)
33
34 ; Startet die Komponente mit dem einkommenden Kanal
35 Kanal -> Komponente
36 new-clock(in-chan)
```

3.5.3 Entwurfsentscheidungen

Die Komponente ist ein eigenständiger Prozess und stellt Funktionen, die in Abhängigkeit zur aktuellen Zeit stehen zur Verfügung. Sie enthält einen State, der die Abweichung speichert. Die Komponente an sich wird nur vom Connector und von der Station verwendet.

Die Clock Komponente verarbeitet in ihrem Prozess die Nachrichten, die im einkommenden Kanal von der Stationskomponente sind. Die Nachrichten in diesem Kanal enthalten zusätzlich die Zeit, zu der sie angetroffen sind. Wenn die Nachricht von einer Station der Klasse A kommt, wird sie weiter behandelt. Nachricht von Stationen der Klasse B werden nicht weiter betrachtet.

Anhand der Ankunftszeit und der Sendezeit und dem vorherigen Offset, wird die Abweichung neu berechnet (siehe Gleichung 3.1).

$$Offset_{i+1} = \frac{Offset_i + (Empfangszeit - Sendezeit)}{2} \tag{3.1}$$

3.5.4 Konfigurationsparameter

- Anfänglicher Offset

3.6 Station

3.6.1 Aufgabe und Verantwortung

Die Station ist für die Verwaltung der Slots und das Koordinieren des sequentielle Senden und Empfangen von Nachrichten verantwortlich. Dabei werden die Nutzdaten des einkommenden Kanals von der Payload-Source Komponente zum Versenden der Nachrichten genommen. Einkommende Nachrichten werden auf den ausgehenden Kanal zur Message-Writer Komponente gelegt.

3.6.2 Schnittstelle

```
; Startet die Komponente mit den Kanaelen
(Kanal, Kanal, Kanal) -> Komponente
new-station(in-chan, out-chan-clock, out-chan-message-writer)
```

3.6.3 Entwurfsentscheidungen

Die Station ist ein eigener Prozess. Sie hat einen State, in dem der aktuelle Slot zum Senden und die Stationsklasse gespeichert ist. Sie benutzt den Connector, um Daten zu senden und zu empfangen. Die Clock Komponente wird verwendet um die jeweiligen Slots und Frames zu identifizieren.

Der Arbeitsablauf der Station besteht aus einer Initalisierungsphase und einer Hauptphase. Die Initialisierungsphase beginnt damit, alle Nachrichten bis zum Ende des aktuellen Frames n zu lesen und die erhaltenen Nachrichten nach freien Slots und der Zeitsynchronisation auszuwerten. Die empfangenen Nachrichten erweitert um die jeweiligen Zeit des Empfanges

werden auf den ausgehenden Kanal zur Clock Komponente getan. Dort werden Sie dann ausgewertet. Die Empfangenen Nachrichten werden außerdem auf den ausgehenden Kanal zum Message-Writer gelegt. Freie Slots werden ausgewählt, indem von den 25 verfügbaren Slots die Slots aus den erhaltenen Nachrichten entfernt werden. Dadurch bleiben nur noch die Slots übrig, in denen im nächsten Frame $n + 1$ nicht gesendet wird. Von diesen freien Slots wird ein zufälliger Slot gewählt, den die Station zum Senden im Frame $n + 1$ verwendet und im State speichert.

Nun beginnt die Hauptphase. In der Hauptphase wird bis zum derzeitigen gespeicherten Slot aus dem State gelesen. Die empfangenen Nachrichten werden wieder nach freien Slots für das nächste Frame und für die Zeitsynchronisation ausgewertet (wie in der Initialisierungsphase). Aus den freien Slots wird wieder ein zufälliger gewählt und im State gespeichert. Das ist der Slot zum Senden im nächsten Frame. Danach fängt die Station an zu senden. Dazu wird ein Nutzdatenpaket aus dem einkommenden Kanal von der Payload-Source Komponente genommen, falls eins vorhanden ist. Wenn keins vorhanden ist, wird nicht gesendet. Wenn eins vorhanden ist, wird diese an die zu sendende Nachricht hinzugefügt, wie auch die Stationsklasse und der Slot aus dem State (Slot für das Senden im nächsten Frame). Diese Nachricht wird dann mit dem Connector versendet. Wenn das Versenden ein Erfolg war (keine Kollision), dann wird bis zum Ende des Frames wieder gelesen. Die dadurch erhaltenen Nachrichten werden wieder zur Zeitsynchronisation auf den ausgehenden Kanal zur Clock Komponente gelegt. Wenn das Versenden nicht erfolgreich war (Kollision), dann wird wie im erfolgreichen Fall vorgegangen, nur wird zusätzlich am Ende des Frames wieder ein neuer freier Slot zum Senden gewählt (in Abhängigkeit der empfangenen Nachrichten). Dieser Slot wird dann wieder im State gespeichert. Danach fängt die Hauptphase wieder von vorne an (Schleife).

3.6.4 Konfigurationsparameter

- Stationsklasse
- Framegröße
- Gesamtanzahl der Slots

4 Kollisionsfreier Ablauf

Es wird ein kollisionsfreier Ablauf in anbetracht von maximal 25 Teilnehmern auf dem Medium angestrebt und näher untersucht.

Die Station stößt in festen Frames sequentiell das Senden und Empfangen an. Kollisionen können hier durch verschiedene Gegebenheiten auftreten. Zum einen kann ein zeitlicher Versatz vorhanden sein, wodurch nicht in den richtigen Slots empfangen und vor allem gesendet wird. Das wird durch die Auswertung der empfangenen Nachrichten und der Neuberechnung der Abweichung der UTC-Zeit (Zeitsynchronisation) abgefangen. Die Station muss also darauf achten, ihre sequentielle Arbeit in den Frames einzuhalten und sich immer in den festen Slots im Frame bewegen. Das wird durch den Connector garantiert, dessen Lese- und Sendevorgänge immer mit Timeouts arbeiten. Dadurch kann sich ein kollisionsfreier Ablauf in endlicher Zeit einpendeln.

Eine andere Quelle für Kollisionen kann entstehen, wenn alle Stationen nach einer festen Regel freie Slots auswählen und dadurch die gleichen Slots zum Senden im nächsten Frame haben. Das wird wiederum durch die Stations Komponente umgangen, in dem immer ein zufälliger freier Slot gewählt wird. Dadurch verteilt sich die Slot Auswahl und ein kollisionsfreier Ablauf kann sich mit der Zeit wieder einpendeln.

Zusätzlich dazu werden Kollisionen vermieden, in dem der Connector das Senden auf Kollisionen überprüft und das der Station meldet. Die Station wählt dadurch ja am Ende des Frames eine neue Station zum Senden im nächsten Frame aus den vorhandenen Nachrichten aus, was eine Kollision aktiv verhindert und zum kollisionsfreien Ablauf beiträgt.

Durch diese Maßnahmen kann ein kollisionsfreier Ablauf in endlicher Zeit erreicht werden.

Erklärung zur schriftlichen Ausarbeitung des Referates

Hiermit erkläre ich, dass ich diese schriftliche Ausarbeitung meines Referates selbstständig und ohne fremde Hilfe verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie die aus fremden Quellen (dazu zählen auch Internetquellen) direkt oder indirekt übernommenen Gedanken oder Wortlaute als solche kenntlich gemacht habe. Zudem erkläre ich, dass der zugehörige Programmcode von mir selbständig implementiert wurde ohne diesen oder Teile davon von Dritten im Wortlaut oder dem Sinn nach übernommen zu haben. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Hamburg, 31. Mai 2017 Daniel Schruhl