**Team**: 9, David Asmuth, Vladimir Malikov

**Aufgabenaufteilung**:

1. Entwurf, Vladimir Malikov.
2. Entwurf, David Asmuth.

**Quellenangaben**: <http://users.informatik.haw-hamburg.de/~scotty/pub/Verteilte-Systeme/AI5-VSP/Aufgabe4/VSP_Aufgabe4.pdf>

**Begründung für Codeübernahme**: Es wurde kein Code übernommen.

**Bearbeitungszeitraum**:

10.06.2015: 3 Stunden  
11.06.2015: 9 Stunden

**Aktueller Stand**: Entwurf fertig. Die Implementation noch nicht angefangen.

**Änderungen im Entwurf**:

**Entwurf**:

# Ziel

In diesem Dokument wird eine P2P Anwendung beschrieben die mittels Zeitmultiplexverfahren die Datenübertragung vornimmt. Dabei wird mittels Zeitfenster und Slots ein einziger Kanal simuliert auf dem mehrere Clients koordiniert Daten austauschen können.

# Rahmenbedingungen

Die Client-Software soll in der Lage sein Kollisionen mit anderen Clients zu erkennen und zu minimieren und spätestens im festen (d.h. keine neuen Clients) laufenden Betrieb keine weiteren Kollisionen mehr zu erzeugen. Es bestehen keine Abhängigkeiten zu anderen Systemen.

## Verwendete Bilbliotheken/externe Module

Für die Umsetzung wird das vorgegebene Programm DataSource verwendet, welches den Inhalt für die Versendeten Nachrichten erzeugt. Zusätzlich wird das von Prof. Klauck bereitgestellte werkzeug.erl für UDP-Multicast, zur Nachrichtenverarbeitung, sowie zum logging verwendet.

## Beschreibung des Startvorgangs

Da die Software aus einem einzigen Client besteht, muss lediglich dieser gestartet werden. Zu Start müssen Adresse, Port und Netzwerkinterface als Parameter angegeben werden.

# Modularisierung

## Client

### Verantwortung:

Versendet Nachrichten innerhalb eines vorgegebenen Intervalls in ein Multicast Netzwerk und empfängt Nachrichten aus dem Netzwerk.

### Aussensicht:

Der Client ist eine Standalone-Anwendung und bietet keine Schnittstellen für weitere Systeme an.

### Innensicht:

Der Client besteht aus:

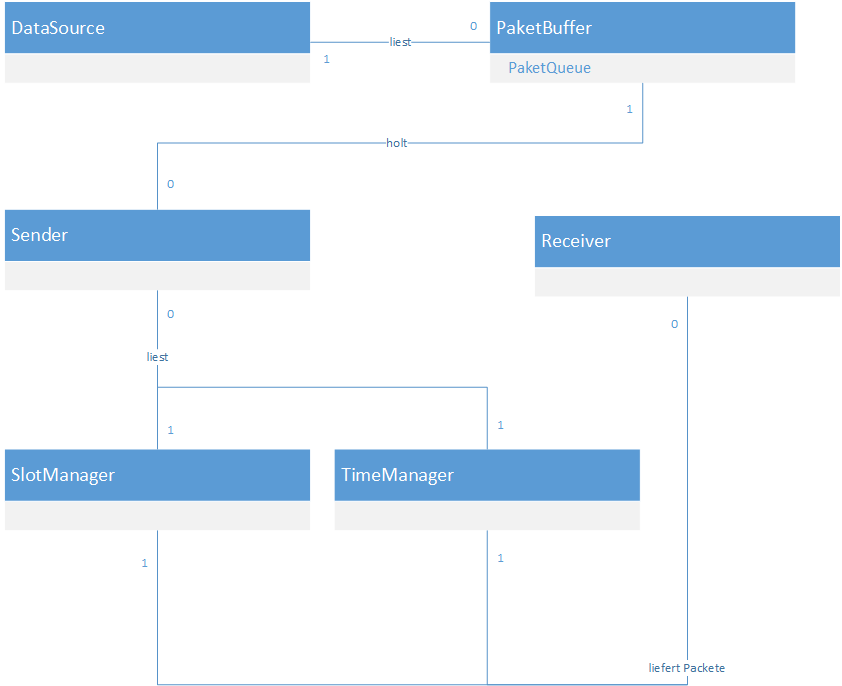
* Der Komponente PaketBuffer, welcher aus den vom DataSource versendeten Daten Pakete baut und in einer Warteschlange einreiht.
* Dem SlotManager, welcher aus empfangenen Paketen den Slot für den nächsten Versand errechnet.
* Dem TimeManager, der die Zeitsynchronisation durchführt und Kollisionen verarbeitet.
* Dem Sender der aus dem Paketbuffer Pakete nimmt, und mittels Informationen von TimeManager und SlotManager entsprechend versendet.
* Dem Receiver der die eingehenden Pakete an den TimeManager und Slotmanager weitergibt, die daraus die Daten für den nächsten Frame berechnen.

### Entwurfsendscheidungen:

Der PaketBuffer ist ein eigener Thread, um die Arbeit der Paketerstellung aus dem Sender auszulagern. Dies minimiert die Zeit die der Sender vor dem tatsächlichen Sendevorgang benötigt. Aus demselben Grund sind Slot- und TimeManager ebenfalls in eigenen Threads.

# Klassenbeschreibung

## Klassendiagram



## Methodenbeschreibung

Im Folgenden werden essenzielle Methoden der einzelnen Klassen beschrieben.

### Main

#### Start()

Startet den Client. Die Startreihenfolge wird später im Code mit den entsprechenden Buchstaben markiert (Nachvollziehbarkeit).

* Erst wird der PacketBuffer Thread gestartet. (a)
* Anschließend werden die SlotManager und TimeManager Theads gestartet (b).
* Deren PIDs werden dem nun zu startenden ReceiverThread übergeben. (c)
* Als letztes wird der Sender Thread gestartet, dem die PIDs des PacketBuffers, und der Time- und SlotManager übergeben werden. Dies sollte verzögert geschen um keinen leeren NachrichtenBuffer zu haben. (d)

### BufferManager

#### Pop()

Liefert ein Paket aus der PaketQueue zurück.

### SlotManager

#### Receive(RecveivedPacket)

Übergibt dem SlotManager ein empfangenes Paket, daraus wird der nächste Slot berechnet.

#### GetSlot()

Liefert eine Slotnummer für den Nächsten Frame. Kann der SLotManager keinen Gültigen Slot liefern, sei es durch Erststart oder Kollision, so gibt er -1 zurück.

### TimeManager

#### Receive(RecveivedPacket)

Übergibt dem TimeManager ein empfangenes Paket, daraus wird der nächste Zeit berechnet.

#### GetTime()

Liefert einen Zeitstempel für den nächsten Slot.

### Receiver

#### Loop()

Empfängt eingehende Pakete und gibt diese an die Manager-Klassen weiter, die die Werte für das nächste Paket berechnen.

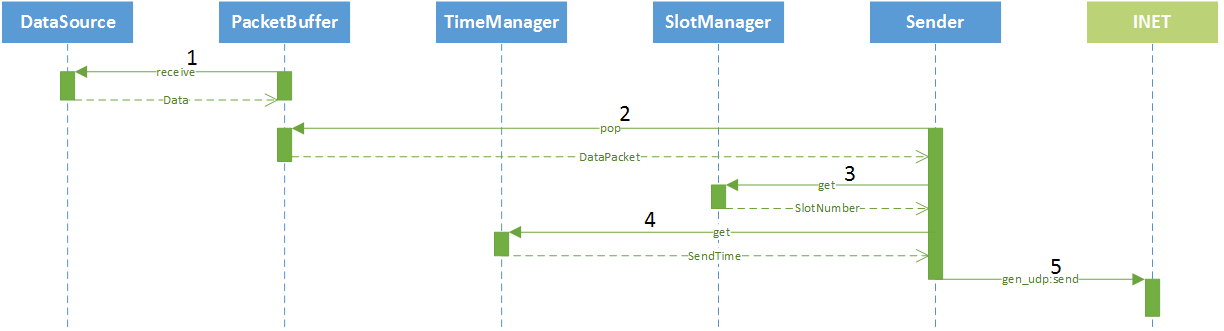
### Sender

#### Loop()

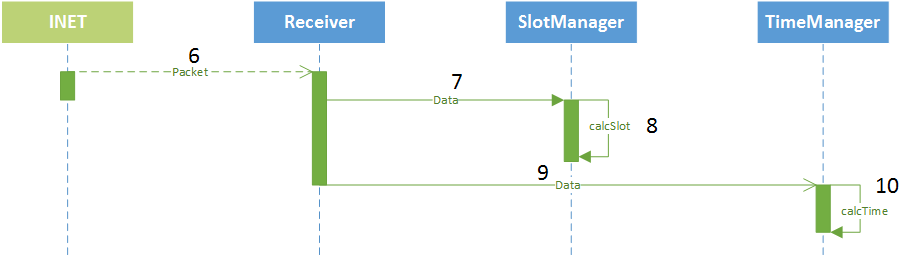
Entnimmt dem Paketbuffer ein Paket, setzt Slot und Zeit mit den Werte von SlotManager und TimeManager und verschickt das Paket anschließend. Dieser Vorgang geschieht in einem Zuvor festgelegten Intervall. Können durch Erststart oder Kollision von den Managern keine gültigen Werte erzeugt werden, wird das Senden für diesen Frame übersprungen.

# Sequenzdiagramme

Paket Senden:



Paket empfangen:



Das folgende Sequenzdiagramm zeigt einen kompletten möglichen Ablauf, dieser umfasst:

* Die Initialisierungsphase
* Erstmaliges Senden
* Weiteres Senden
* Kollision

# C:\Users\Piranha\Dropbox\Projekte\VSPraktikum\Aufagbe 4\Entwurf\Bilder\EinRun1.png

# C:\Users\Piranha\Dropbox\Projekte\VSPraktikum\Aufagbe 4\Entwurf\Bilder\EinRun2.png

# C:\Users\Piranha\Dropbox\Projekte\VSPraktikum\Aufagbe 4\Entwurf\Bilder\EinRun3.png