Team: 9, David Asmuth, Vladimir Malikov

Aufgabenaufteilung:

1. Entwurf, Vladimir Malikov.
2. Receiver.java, Vladimir Malikov
3. DataExchange.java, Vladimir Malikov
4. Ticker.java, Vladimir Malikov
5. SlotManager.java Vladimir Malikov
6. TimeManager.java Vladimir Malikov
7. Entwurf, David Asmuth.
8. Packetbuffer.java David Asmuth
9. Sender.java David Asmuth
10. Starter.java 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
12.06.2015:	3 Stunden
13.06.2015	2 Stunden
14.06.2015	3 Stunden
16.06.2015	10 Stunden
17.06.2015	8 Stunden

Aktueller Stand: Entwurf fertig. Die Implementation ist abgeschlossen.

Änderungen im Entwurf:
Es ist eine Ticker Klasse hinzugekommen, die den Takt für das gesamte System vorgibt . Zudem ist Zwischen Receiver und den Managern eine DataExchange Klasse entstanden die die Kollisionsbehandlung und die Datenweitergabe organisiert.

Entwurf:

1. Ziel

In diesem Dokument wird eine Multicast 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.

2. Rahmenbedingungen

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

2.1 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.

2.2 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.

3. 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:

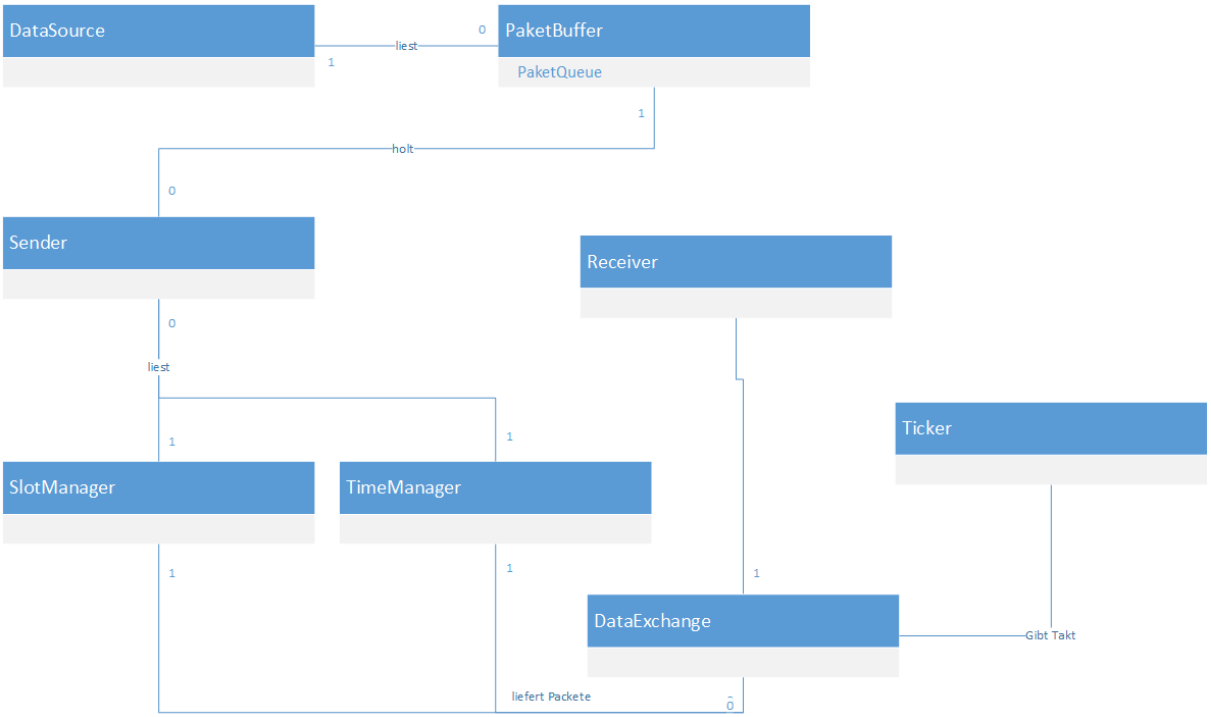
- Der Komponente PacketBuffer, welcher aus den vom DataSource versendeten Daten Pakete baut und in einer Warteschlange einreiht.
- Dem SlotManager, welcher aus empfangenen Paketen den Slot für den nächsten Versand errechnet.
- Dem TimeManager, der die Zeitsynchronisation durchführt und vom Sender abgefragt wird.
- Dem Sender der aus dem PacketBuffer Pakete nimmt, die Daten aus TimeManager und SlotManager schreibt und versendet.
- Dem Receiver der die eingehenden Pakete an den TimeManager und SlotManager weitergibt, die daraus die Daten für den nächsten Frame berechnen.

Entwurfsendscheidungen:

Der PacketBuffer 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.

4. Klassenbeschreibung

4.1 Klassendiagram



4.2 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.

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 PacketBuffer ein Paket, setzt Slot und Zeit mit dem 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.

5. Ablauf

Die Initialisierungsphase

Es wird der Datenverkehr abgehört, um zu bestimmen in welchem Slot man im nächsten Frame senden soll. Der Sender wird für diese Zeit schlafen gelegt.

Erstmaliges Senden

Am Begin vom nächsten Frame, wacht Sender auf und fragt den Slot ab für diesen Frame. An Hand der Framenummer, bestimmt er wie lange er wieder schlafen soll, bis sein Slot erreicht ist. Beim Aufwachen fordert er ein Datenpacket von PacketBuffer an. Danach die Slotnummer für den nächsten Frame vom SlotManager. Im letzten Moment vor absenden, wird die Zeit vom TimeManager abgefragt. Das Paket wird abgeschickt. An Hand der Slotnummer kann der Sender bestimmen, wie lange er jetzt schlafen soll, bis sein Slot wieder dran ist.

Weiteres Senden

Der Sender wacht auf und fordert ein Datenpacket von PacketBuffer an. Danach die Slotnummer vom SlotManager. Im letzten Moment vor absenden, wird die Zeit vom TimeManager abgefragt. Der Packet wird abgeschickt. An Hand der Slotnummer kann der Sender bestimmen, wie lange er jetzt schlafen soll, bis sein Slot wieder dran ist.

Kollision

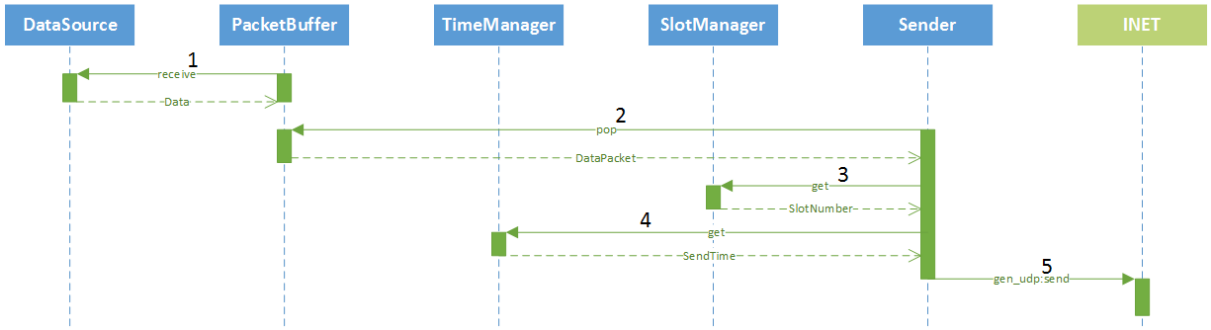
Wird eine Kollision festgestellt, die Reservieren der Slotnummer verhindert hat, wird der Sender aufgeweckt und darüber informiert. Sender und SlotManger übergehen in die Initialisierungsphase.

Anmerkungen

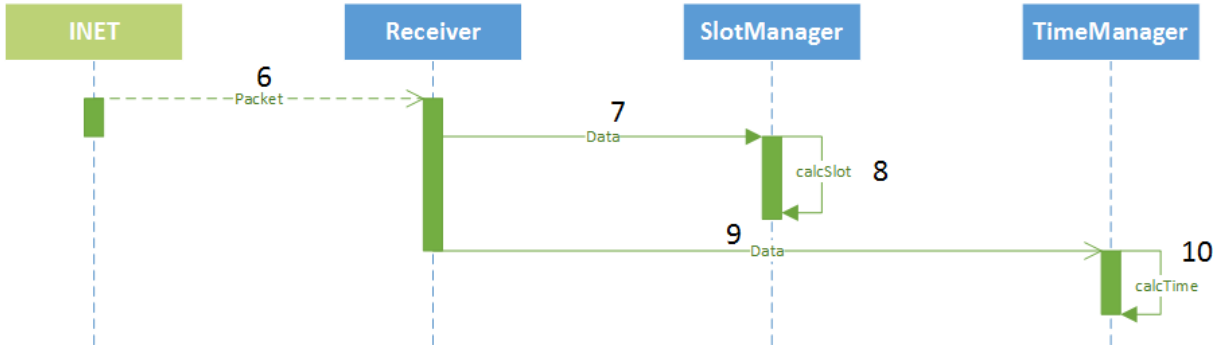
Die Slot-Auswahl erfolgt per Zufall auf Slots die vermeintlich frei sind. Die Zeitberechnung erfolgt nach dem Berkeley Algorithmus.

6. Sequenzdiagramme

Paket Senden:

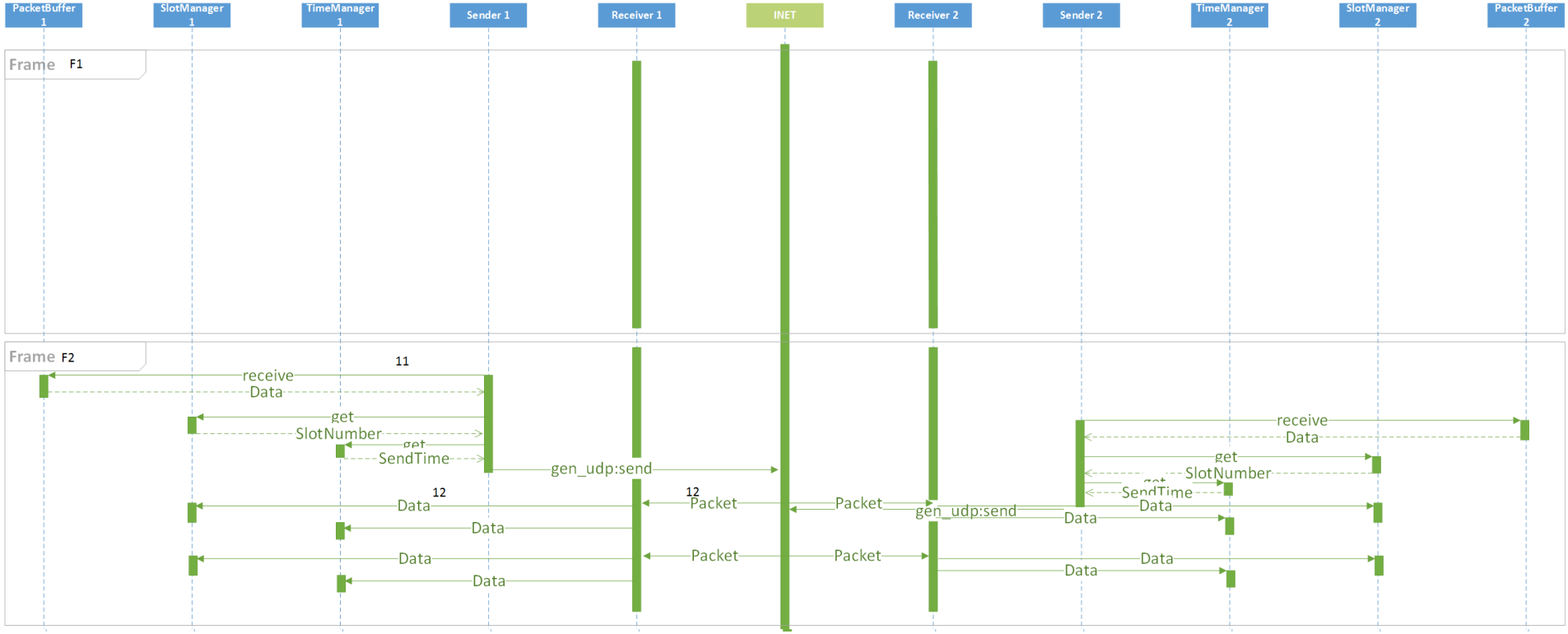


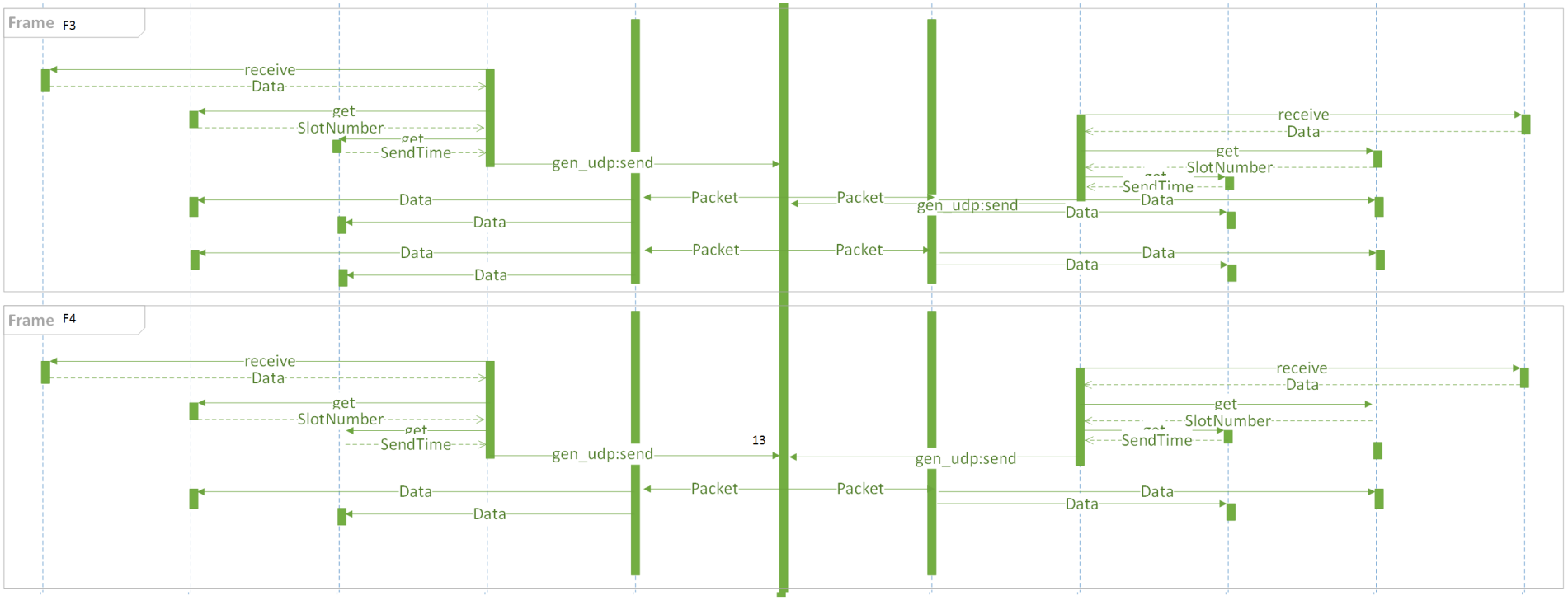
Paket empfangen:

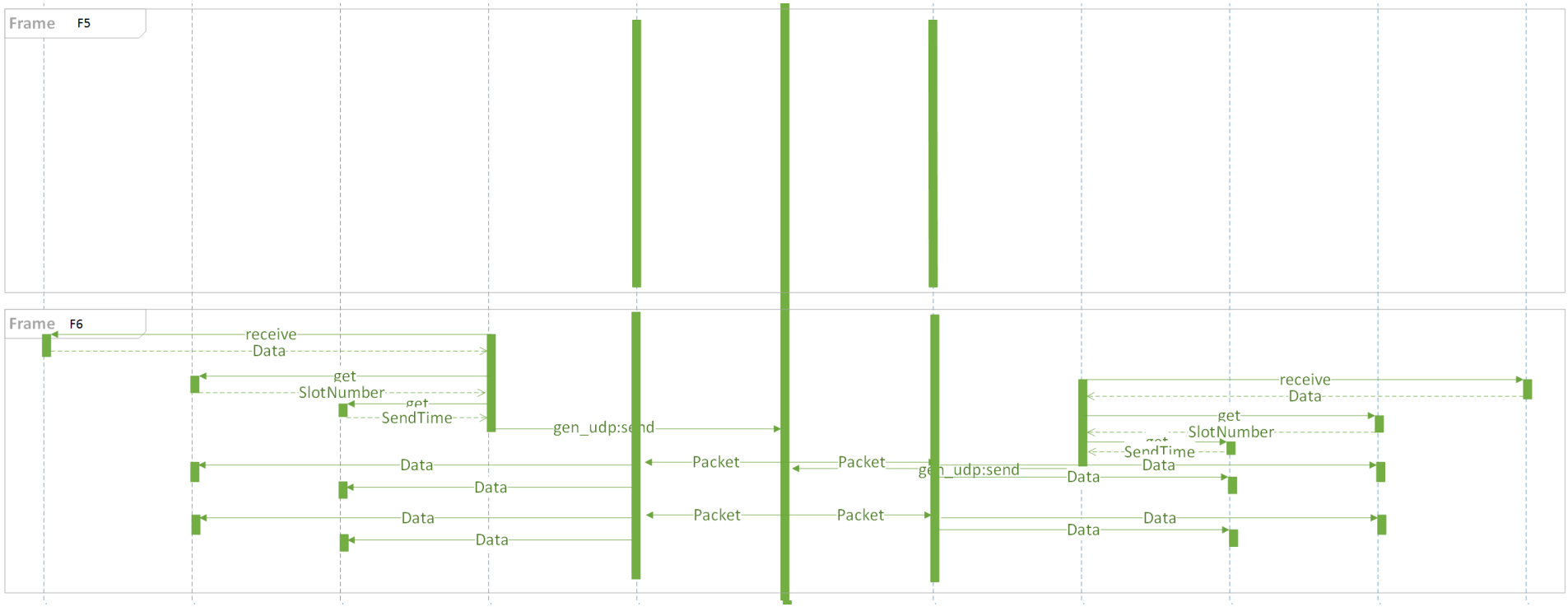


Das folgende Sequenzdiagramm zeigt einen kompletten möglichen Ablauf, dieser umfasst:

- Die Initialisierungsphase
- Erstmaliges Senden
- Weiteres Senden
- Kollision




Frame F3
receive
Data
get
SlotNumber
get
SendTime
gen_udp:send
Packet
Packet
gen_udp:send
receive
Data
get
SlotNumber
get
SendTime
Data
Data
Data
Data
Data
Packet
Packet
Data
Data
Data
Frame F4
receive
Data
get
SlotNumber
get
SendTime
gen_udp:send
13
gen_udp:send
receive
Data
get
SlotNumber
get
SendTime
Data
Data
Packet
Packet
Data
Data


Frame F5
Frame F6
receive
Data
get
SlotNumber
get
SendTime
gen_udp:send
Packet
Packet
Data
Data
Data
Packet
Packet
Data
receive
Data
get
SlotNumber
SendTime
Data
Data
Data
Data