Team: Team 1, Lennart Hartmann, Nils Eggebrecht

Aufgabenaufteilung:

1. Verstehen was für die Aufgabe getan werden muss,

   Recherche wie das Protokoll umgesetzt werden kann,

   überlegen wie man das Programm aufbauen bzw. schachteln kann,

   verstehen wie das Bsp. Programm geschachtelt und aufgebaut ist,

   verstehen von dem Code eines Programms vom früheren Semester dieser Aufgabe, da wir festgestellt haben, dass das neu schreiben des Codes für die Aufgabe zeitlich nicht, bis Mittwoch 15 Uhr, erreicht werden kann,

   UML Klassendiagramm anhand des Codes eines Programms vom früheren Semester erstellen,

   planen was am Code vom Früheren Semester geändert werden muss

2. Verstehen was für die Aufgabe getan werden muss,

   überlegen wie man das Programm aufbauen kann,

   verstehen wie das Bsp. Programm geschachtelt und aufgebaut ist,

   den Beispielcode zum laufen bekommen (herausfinden welche Schnittstelle und welche Adresse ich bei meinem Rechner nutzen muss, damit ich das Starterscript ausführen kann und den Sniffer nutzen kann,

   planen was am Code vom Früheren Semester geändert werden muss,

   verstehen von dem Code eines Programms vom früheren Semester dieser Aufgabe, herausfinden was an der Aufgabe nicht der aktuellen Aufgabe dieses Semesters entspricht,

   Erstellen dieses Dokumentes

   Wir saßen beide während wir gearbeitet haben neben einander und haben pair programming gemacht.

Quellenangaben:

Code von vorherigen Kommilitonen

Java 8 api

https://docs.oracle.com/javase/8/docs/api/

z.B für Sockets,

Änderung von TTL in Ubuntu

https://www.youtube.com/watch?v=AfUYO3thyNg

Bearbeitungszeitraum:

Wir haben ca. 3 Wochen für die Bearbeitung des Praktikum VSP2 gebraucht, da wir uns dort kein Bsp. Code von vorherigen Kommilitonen geholt haben und jede Woche ca. 16h und mehr pro Person in diese Aufgabe gesteckt haben.

Da wir mit dem 2. Praktikum bis letzten Donnerstag beschäftigt waren, war keine Zeit sich vorher an die VSP Aufgabe 3 zu setzen. Es ist geplant den Dienstag für die 3. Aufgabe zu verwenden, damit wir bis Mittwoch eine funktionsfähige Version haben.

19.11.17 9h je Team Mitglied

Aktueller Stand: Der Aktuelle Stand ist, dass wir das Programm soweit funktioniert und das der Next Slot nicht dem letzten Slot entspricht.

========== Frame: 1511112887 ==========

1511112887018 (slot 1): received 'team 01-02' A next slot: 1 TX: 1511112887020 (-2 / 0)

1511112887059 (slot 2): received 'team 01-01' A next slot: 2 TX: 1511112887060 (-1 / 0)

Wir müssen das Programm noch weiter verstehen um festzustellen, ob weitere Probleme der Aufgabenstellung nicht der Aufgabe entsprechen.

Änderungen des Entwurfs: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. des Vorentwurfs vorgenommen worden.>

Entwurf: Das UML Klassendiagramm entspricht dem aktuellen Stand des Programms und es ist im Anhang der Email als extra PDF.

Der unten stehende Entwurf ist nicht unser Entwurf, wir haben ihn übernommen, da wir es zeitlich nicht geschafft hätten einen eigenen zu entwerfen.

In 18 Stunden Arbeit ist es leider nicht möglich genug Wissen über die Aufgabe zu recherchieren, dass wir einen Entwurf erstellen können, der der mit der Testumgebung kompatibel ist.

Inhaltsverzeichnis

Kommunikation

Für die Kommunikation wird das UDP Protokoll verwendet, da es nicht zu einem Verbindungsaufbau zwischen den einzelnen Host kommt, sondern über Multicast kommuniziert wird.

Nachrichtenformat

Für die Nachrichtenpakete ist einheitlich nachfolgendes Format zu verwenden:
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 Byte

Multicast Adresse und Port

Multicast-Adresse und Port müssen bei Initialisierung als Parameter mitgegeben werden oder als Default-Wert mit diesen Werten gesetzt werden:
Adresse: 225.10.1.2
Empfangsport: 15002

Frames und Slots

Es wird versucht jeweils von jedem Sender in jedem Frame einen Slot zu verwenden um zu Kommunizieren. Ein Frame geht dabei genau eine Sekunde und startet immer zur vollen Sekunde. Ein Frame besteht aus 25 Slots, die also 40 ms lang sind.

Datenquelle

Die Datenquelle ist die Erzeugungskomponente der Nutzdaten. Sie erzeugt ein 24Byte Nutzdatenpakete(Byte 0-9: Stationsname, Rest: Daten). Die Datenquelle ist nicht zu implementieren, sondern herunterzuladen.

Sender

Der Sender empfängt Daten aus dem DatasourceBuffer und sendet sie per UDP-Multicast an die Empfänger.

Ablauf

Der Sender holt sich aus dem DatasourceBuffer die Nutzdaten und sendet eine Anfrage an den Empfänger nach einer Slot Nummer sowie einer Zeitangabe(Systemzeit mit Offset[Long]). Aus der Zeitangabe muss ein Offset gebildet werden und mit diesem der Sendezeitpunkt berechnet werden. Die Slot Nummer muss in einem Bereich von 1-25 liegen, um gültig zu sein. Bei einer -1 ist keine Slot Nummer mehr frei. Danach erfolgt eine Kollisionsabfrage beim Empfänger und bei negativem Ergebnis wird gesendet, falls der Slot nicht schon vorbei ist. Bei einem positiven Ergebnis wird ein Logeintrag veranlasst und sich zusätzlich eine neue Slot Nummer für den nächsten Frame besorgt. Wenn der Wert -1 empfangen wird, ist der früher reservierte Slot gültig.

DatasourceBuffer

Der DatasourceBuffer nimmt die 24-Byte Nutzdaten entgegen und hält die zu versendenen Daten vorrätig. Wichtig ist dabei zu wissen das der DatasourceBuffer die Daten entgegen nimmt bzw. vorrätig hält, aber der Sender seine eigene Datenrate r hat mit der er Nachrichten verschickt.

Empfänger

Der Empfänger lauscht auf dem Nachrichtenkanal, um die Kommunikation zu verfolgen und dem
Sender die Informationen zur Verfügung zu stellen. Der Empfänger hat eine Slot Allocation Table.

Initialisierung

Bei der Initialisierung beobachtet der Empfänger zunächst einen kompletten Frame. Dabei wird herausgefunden welcher Slot im nächsten Frame frei ist, außerdem wird die Uhr synchronisiert. Mit diesen Informationen wird dann der Senderprozess gestartet.
Die Datensenke und der DatasourceBuffer werden sofort mit dem Empfänger initialisiert.

Schnittstelle

- int getFreeSlotNextFrame(): Gibt einen bisher noch nicht belegten Slot im nächsten Frame
- long getTime(): Gibt die aktuelle Uhrzeit
- boolean isCollision(): Überprüft ob ein Senden zu eine Kollision führen würde. True wenn dem so ist
- getSlotForCollusion(): Wird vom Sender am Ende eines Frames aufgerufen. Wenn der Sender eine Kollision erzeugt hat, gibt diese Funktion einen noch nicht reservierten Frame zurück, ansonsten wird -1 zurückgegeben. Das bedeutet für den Sender, dass der Frame den er beim Senden über getFreeSlotNextFrame() bekommen hat, der gültige Frame ist.

Uhrensynchronisation

Der Empfänger bekommt bei seiner Initialisierung mitgeteilt, welcher Typ (A oder B) er ist. Je nachdem wird dann die Uhr mit jeder eintreffenden Nachricht synchronisiert (siehe Zeitsynchronisation).

Slotverarbeitung

Die Slotverarbeitung wird immer am Ende eines Slots gemacht, da erst dann sichergestellt werden kann ob Kollisionen aufgetreten sind oder nicht. Sind keine Kollisionen aufgetreten, werden die Informationen verarbeitet, ansonsten verworfen.

Kollisionserkennung

Eine Kollision ist aufgetreten, sobald innerhalb eines Slots zwei oder mehr Nachrichten eingetroffen sind. Im eigenen Senderslot reicht bereits eine eingegangen Nachricht um eine Kollision vorauszusehen. Dann muss der Sender benachrichtigt werden, keine Nachricht zu senden (siehe Sender Laufzeitsicht).

Datensenke

Die Datensenke ist für die Entgegennahme der Nutzdaten und das Logging zuständig.

Initialisierung

Die Datensenke wird mit dem Start des Programms initialisiert und ist für keine weitere Komponenteninitialisierung zuständig.

Datenausgabe

Das Logging und Ausgabe der Nutzdaten findet über den Stdout Kanal statt. Folgende Ereignisse sind zu Loggen:

- Erkannte Kollisionen
    - Zusätzlich vermerk wenn selbst beteiligt
- Anzahl der Frames in denen gesendet wurde
- Anzahl der Frames in denen nicht gesendet wurde
- Eingegangene und Ausgegangene Nutzdaten