Technische Universität München
Lehrstuhl für Kommunikationsnetze
Prof. Dr.-Ing. W. Kellerer
Betreuer - Andreas Blenk, Christian Sieber

Internetkommunikation - Class Project

Programmieraufgabe 2

UDP-Picture Cast

Das verbindungslose Übertragungsprotokoll UDP ist auf Grund seines geringen Overheads und seiner geringen Latenz vor allem bei der Live-Übertragung von Multimedia-Inhalten, wie Sprache und Videos, beliebt. In dieser Programmieraufgabe werden wir uns auf die Übertragung von Bildern beschränken. Ziel dieser Programmieraufgabe ist es, die Besonderheiten von verbindungsloser Übertragung kennenzulernen.

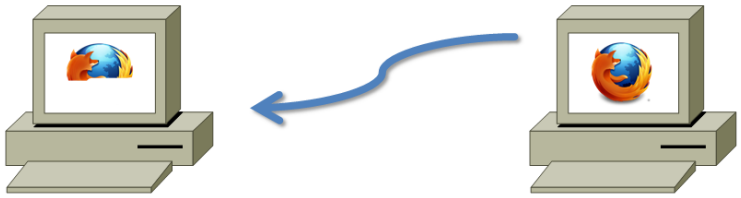


Abbildung 1: UDP-PictureCast

Wir werden uns in dieser Aufgabe mit den folgenden Punkten auseinandersetzen:

- Wie kann man die Bildinformationen auf einzelne Datenpakete aufteilen?
- Wie kann eine Überlastung der Übertragungsstrecke vermieden werden?
- Wie können wir Paketvertauschungen (*Packet Reordering*), Paketverluste (*Packet Loss*) und Paketduplizierungen (*Packet Duplication*) erkennen und deren Auswirkung auf die Qualität des übertragenen Bildes minimieren?

Folgende Dateien werden für diese Aufgabe in einer zip-Datei bereit gestellt. Es dürfen keine weiteren Dateien/Module hinzugefügt werden. Außerdem dürfen die Namen der Funktionen sowie ihre Argumente nicht verändert werden. Sie dürfen ihren Programmcode um eigene Funktionen, soweit notwendig, erweitern. Der bereitgestellte Komponententest prüft auch einzelne Teilfunktionen. Sie sollten diesen nach jeder Teilaufgabe ausführen um Fehler in Ihrer Implementierung frühzeitig zu erkennen. Für die Abgabe relevant sind nur die zwei Dateien **picture_cast_todo.py** und **picture_receiver_todo.py**! Diese zwei Dateien müssen bei der Abgabe zusammen mit den unmodifizierten bereitgestellten Programmteilen funktionieren.

Datei	**Abgabe**	**Beschreibung**
picture_cast.py	nein	Sender
picture_receiver.py	nein	Empfänger
picture_cast_todo.py	**ja**	Der Teil des Senders der selbst zu implementieren ist.
picture_receiver_todo.py	**ja**	Der Teil des Empfängers der selbst zu implementieren ist.
aufghelper.py	nein	Helferfunktionen für das Dekodieren des Bildes
aufgsettings.py	nein	Einstellungen (z.B. Ziel-Host der Übertragung)
picture_test.py	nein	Komponententest der Aufgabe
f.300x300.png	nein	Das Beispielbild

Für ein Bestehen der Aufgabe muss der Komponententest für alle drei Aufgaben erfolgreich sein! Der Empfänger zeigt, sobald er ausgeführt wurde, ein Fenster mit dem Originalbild an. Sollte sich dieses Fenster nicht öffnen, besuche Sie die Moodle Seite für Lösungsvorschläge.

Aufgabe 1: Aufteilung der Bildinformationen

Im Gegensatz zu verbindungsorientierten Protokollen wie TCP können wir einem UDP Socket nicht einfach größere Datenmengen übergeben. Wir müssen uns Gedanken machen, wie wir die Bildinformationen in einzelne Pakete verpacken. In dem bereitgestelltem Aufgabentemplate sind die Funktionen für das Dekodieren und das Anzeigen des Bildes schon implementiert. Unser Beispielbild enthält pro Bildpunkt vier Farbkomponenten (drei Farben + Transparenz). Jeder dieser vier Komponenten ist als 4-Byte Gleitkommazahl (float) gespeichert. Für die Darstellung eines Bildes mit einer Breite und Höhe von jeweils 300 Pixeln ergibt dies einen Speicherverbrauch von 1.44×10^6 Bytes (1.37 Megabytes). Kompressionstechniken werden nicht verwendet.

- Implementieren Sie *getNrOfPxlPerPkt(..)*. Die Funktion soll die Anzahl der vollen Pixel, die in ein Paket (unter Beachtung der maximalen Nutzdatenlänge (*Payload size*) passen, zurückgeben. Beachten Sie, dass für diese Berechnung möglicherweise nicht alle Parameter der Funktion gebraucht werden.
- Implementieren Sie *getReqPktCnt(..)*. Die Funktion soll die Gesamtzahl der Pakete, die benötigt werden um das gesamte Bild zu übertragen, zurückgeben.
- Implementieren Sie *genPayload(..)*. Die Funktion soll aus den Bilddaten (*pixelData*, ein 3-dimensionales Array mit Höhe, Breite und Farbwerte) und der Anzahl der Pixel pro Paket die Nutzdaten für das Paket mit der Nummer *pktNr* generieren. Die Nutzdaten sollen im Moment keine weiteren Daten außer den Pixeldaten aus *pixelData* enthalten. Informationen wie die Höhe und Breite des gesendeten Bildes sind auf dem Empfänger schon vor der Übertragung vorhanden. Tipp: Generieren Sie sich erst eine Liste aus der in diesem Paket zu sendenden Farbwerten und anschließend serialisieren Sie diese mittels *struct.pack('%sf' % len(fltLst), *fltLst)*. Siehe Abbildung 2 für ein Beispiel. Auf welche Weise Sie die Pixeldaten auf die einzelnen Pakete verteilen, ist Ihnen überlassen. Die Gesamtzahl der gesendeten Pakete muss genau dem Rückgabewert von *getReqPktCnt(..)* entsprechen. Hinweis: Der Komponententest für diesen Aufgabenteil bewertet *genPayload(..)* zusammen mit der nun folgenden Implementierung der Funktion *onRcvPkt(..)* und gibt daher erst OKAY zurück wenn beide Funktionen korrekt implementiert worden sind.
- Implementieren Sie *onRcvPkt(..)*. Die ankommenden Pakete sollen wieder in das Daten Array (*rcvdPixelData*) eingefügt werden, basierend auf der Ankunftsreihenfolge der Pakete.
- Führen Sie den mitgelieferten Empfänger aus und senden Sie anschließend mittels Ihrer modifizierten Version des Senders das Beispielbild an den Empfänger. Sollte der Empfänger und der Sender sich auf unterschiedlichen Computern befinden, kann der Zielport und die Ziel-IP ggf. über die *aufgsettings.py* Datei konfiguriert werden. Was können Sie feststellen?

		Width			
		0	1	2	3
Heigth	0	{0.5},{0.1},{0.6},{0.2}	{0.5},{0.1},{0.6},{0.2}	{0.5},{0.1},{0.6},{0.2}	{0.5},{0.1},{0.6},{0.2}
	1	{0.1},{0.2},{0.2},{0.8}	{0.2},{0.1},{0.6},{0.2}	{0.1},{0.7},{0.9},{0.3}	{0.3},{0.2},{0.1},{0.9}
	2	{0,8},{0.1},{0.2},{0.1}	{0.5},{0.1},{0.6},{0.2}	{0.5},{0.1},{0.6},{0.2}	{0.3},{0.2},{0.1},{0.9}
	3	{0.5},{0.1},{0.6},{0.2}	{0.3},{0.2},{0.1},{0.9}	{0.5},{0.3},{0.6},{0.8}	{0.3},{0.2},{0.1},{0.9}
	4	{0.1},{0.5},{0.2},{0.9}	{0.5},{0.1},{0.6},{0.2}	{0.5},{0.7},{0.2},{0.4}	{0.1},{0.5},{0.2},{0.2}

Abbildung 2: Beispiel für pixelData mit einer Höhe und Breite von 5 Pixel. In diesem Beispiel haben wir für die Übertragung in einem Paket 4 Pixel ausgewählt, angefangen von Breite 2 und Höhe 1. Die Liste der Floats für ein Datenpaket wäre in diesem Beispiel {0.1, 0.7, 0.9, 0.3, 0.3, 0.2, 0.1, 0.9, 0.8, 0.1, 0.2, 0.1, 0.5, 0.1, 0.6, 0.2}

Führen Sie den bereitgestellten Komponententest aus um Ihre Implementierung der Aufgabe zu prüfen.

Aufgabe 2: Überlastung der Übertragungsstrecke vermeiden

Eine einfache Möglichkeit eine Überlastung der Übertragungsstrecke zu vermeiden ist die Datenrate zu limitieren. Wir können dies dadurch realisieren, dass wir nach jedem gesendetem Paket eine kurze Pause einfügen.

- Modifizieren Sie dazu *sendPkt(..)* und implementieren Sie den *sleepTime* Parameter.
- Variieren sie die AUFGABE2_DELAY Variable im Kopf der Datei (*picture_cast_todo.py*). Je nach Übertragungsstrecke und verwendeter Hardware für den Empfänger werden Sie bemerken, dass unterschiedlich lange Pausen zwischen den Paketen notwendig sind um eine fehlerfreie Übertragung zu gewährleisten. Tipp: Eine Pause von fünf Millisekunden zwischen den Paketen ist ein guter Ausgangspunkt für eigene Experimente.

Führen Sie den bereitgestellten Komponententest aus um Ihre Implementierung der Aufgabe zu prüfen.

Aufgabe 3: Paketverlust, Paketvertauschungen, Paketduplizierungen

Die Limitierung der Datenrate aus Aufgabe 2 hat geholfen, Paketverluste zu reduzieren. Dennoch können wir bei dem verbindungslosen Protokollen nie ausschließen, dass Pakete verloren gehen, in falscher Reihenfolge ankommen oder auf der Übertragungsstrecke dupliziert werden. Angelehnt an dem verbindungsorientierten TCP Protokoll werden wir nun eine Sequenznummer für jedes Paket einführen.

- Setzen Sie die AUFGABE variable im Kopf der Dateien *picture_cast_todo.py* und *picture_receiver_todo.py* auf 3.
- Implementieren Sie *genPayloadEx(..)* und fügen Sie eine Sequenznummer (*seqNr*) in das Paket ein. Benutzen sie die struct.pack Funktion mit dem *fmt* Parameter '>I' um die Sequenznummer in einen 4 Byte Wert umzuwandeln und fügen Sie diese 4 Bytes vor den Nutzdaten ein. Die mögliche Überschreitung der (vorgegebenen) maximalen Nutzdatenlänge durch die Sequenznummer ignorieren wir hier.
- Implementieren Sie *onRecvPktEx(..)*. Extrahieren Sie erst die Sequenznummer aus den empfangenen Paketdaten und verwenden Sie diese um die duplizierten (*duplPkts*), verlorenen Pakete (*lostPkts*) und Paket in falscher Reihenfolge (*reorderedPkts*) zu zählen. Verwenden Sie außerdem die Sequenznummer um die empfangenen Pixeldaten an der richtigen Stelle in *rcvdPixelData* einsortieren zu können. Hinweis: Auf der Seite des Empfängers wird die Funktion *onEndMsg(..)* aufgerufen sobald die Übertragung des Bildes abgeschlossen wurde. Sie werden diese brauchen um zusammen mit selbstdefinierten globalen Variablen die Anzahl der verlorenen Pakete zu berechnen.

Führen Sie den bereitgestellten Komponententest aus um Ihre Implementierung der Aufgabe zu prüfen. Für die Abgabe folgenden Sie bitte den Anweisungen auf Moodle. Die Abgabe soll nur die Dateien *picture_cast_todo.py* und *picture_receiver_todo.py* enthalten.

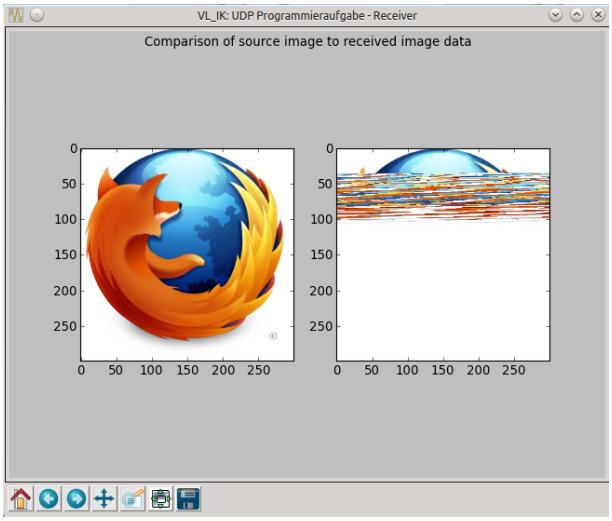
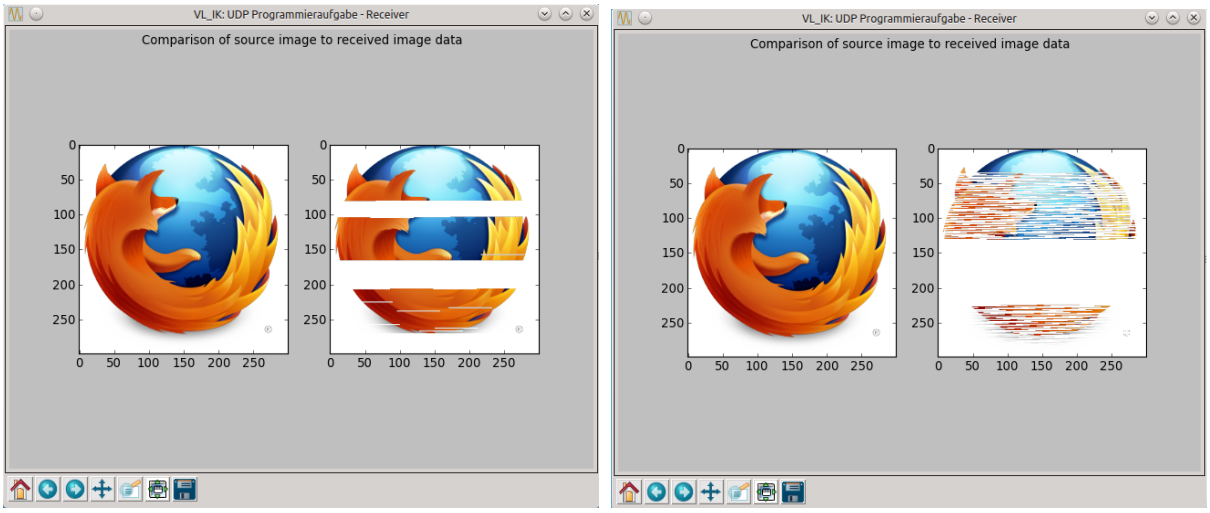


Abbildung 3: Fehlerhafte Übertragung des Beispielbildes ohne Sequenznummer



(a) Beispiel 1 **(b)** Beispiel 2

Abbildung 4: Fehlerhafte Übertragung des Beispielbildes mit Sequenznummer

```
========================
 UDPPictureCast Unit Test
========================

## Aufgabe 1 ##

getNrOfPxlPerPkt: OKAY
getReqPktCnt: OKAY
genPayload + OnRcvPkt: OKAY

## Aufgabe 2 ##

sendPkt: OKAY

## Aufgabe 3 ##

genPayloadEx + onRcvPktEx: OKAY

EVERYTHING OKAY. ALL TESTS PASSED.
```

Abbildung 5: Ausgabe wenn alle Tests bestanden wurden.