


TCP-Kommunikation mit Sockets

Installation von NEWUDPL

Installation von newudpl unter:

<https://www.cs.columbia.edu/~hgs/research/projects/newudpl/>

 [newudpl-1.4.tar.gz](#) 2003-03-25 13:45 63K

Anschließend extrahieren & im Terminal öffnen

```
./configure
```

```
make
```

Dann kommt ein warning, dass `string.h` nicht inkludiert wurde im `host2ip.c` file. Dieser warning sollte nicht ignoriert werden.

```
sudo vim host2ip.c
```

```
include „string.h“
```

Kommunikation zwischen Sender und Empfänger

Zuerst muss der Receiver gestartet werden:

Outbound Receiver:

```
./newudpl -vv -p 5001 -i "localhost/*" -o localhost:5003
```

```
-o Zielhost [ [ : | / ] Port ]
```

Legt einen Zielhost für ausgehende Pakete fest.

Standardwert: Host - localhost, Port 41194

Receiver Argument

```
python3 receiver.py --host localhost --port 5000 --dest_host localhost --dest_port 5001
```

Anschließend kann der Sender gestartet werden:

Outbound Sender:

```
./newudpl -vv -p 5002 -i "localhost/*" -o localhost:5000 -O 80
```

```
-O Verlust der Paketreihenfolge
```

Legt eine Rate für die zufällige Änderung der Reihenfolge von Paketen fest. Der Zielhost wird einige Pakete in einer bestimmten Rate in unordentlicher Reihenfolge erhalten. Die Rate wird in Prozent angegeben.

Verfügbarer Range: 1 - 99

Standardwert: 0

Senden der Daten mit 50% Paketverlust

```
./newudpl -vv -p 5002 -i "localhost/*" -o localhost:5000 -L 50
```

```
-L Zufällige Paketverlustrate
```

Legt eine Rate für die zufällige Erzeugung von Paketverlusten für ausgehende Pakete fest. Die Rate wird in Prozent angegeben.
Verfügbarer Range: 1 - 99
Standardwert: 0

Sender Argument:

```
python3 receiver.py --host localhost --port 5000 --dest_host localhost --dest_port 5001
```

Versenden von Bildern

Modifikation des Receivers

```
import base64
```

Das **base64**-Modul wird importiert, um die Base64-Codierung und -Decodierung zu ermöglichen.

```
self.output_file = open(OUTPUT_FILE, mode='wb')
```

Der Modus zum Öffnen der Ausgabedatei wurde von **'w'** (Textmodus) auf **'wb'** (Binärmodus) geändert, was darauf hindeutet, dass die Daten nun binär (Base64-codiert) geschrieben werden.

```
data = base64.b64decode(load_json["data"])
```

Die empfangenen Daten werden hier mit Base64 dekodiert. Das ursprüngliche Programm ging davon aus, dass die Daten im Klartext empfangen werden, während die modifizierte Version Base64-codierte Daten erwartet.

```
packet = {  
    "acknowledgement_number": recv_seq,  
    "internet_checksum": self.get_checksum(recv_seq)  
}
```

Die Daten im Paket werden nun als Base64-codierte Daten gespeichert.

```
sndpkt = pkt.encode()
```

Das Paket wird vor dem Senden nun in Bytes (binär) codiert.

Modifikation des Senders

```
import base64
```

Das **base64**-Modul wird importiert, um die Base64-Codierung und -Decodierung zu ermöglichen.

```
self.image_file = args.image_file
```

Ein neues Attribut namens **image_file** wird eingeführt, um den Dateipfad zum Bild zu speichern, das gesendet werden soll.

```
with open(self.image_file, 'rb') as inf: # OPEN THE FILE
    debug_index = 0
    current_payload = inf.read(MAX_PAYLOAD)
    while len(current_payload) > 0:
        is_fin = len(current_payload) < MAX_PAYLOAD
        image_data = base64.b64encode(current_payload).decode('utf-8') #
read the image data
        self.buffer.append(self.make_packet(image_data, self.seq,
is_fin=is_fin, index=debug_index))
        current_payload = inf.read(MAX_PAYLOAD)
        self.seq = 1 - self.seq
        debug_index += 1
```

In diesem Abschnitt wird die Bilddatei im Binärmodus geöffnet. Dann wird in einer Schleife die Datei blockweise gelesen, in Base64 codiert und die codierten Daten zusammen mit anderen Informationen in eine Liste eingefügt. Die Sequenznummer wird alternierend aktualisiert, und ein Debug-Index wird für jede Datenpaketübertragung erhöht. Dieser Prozess wiederholt sich, bis das Ende der Datei erreicht ist.

```
payload, addr = self.socket.recvfrom(2048)
```

In dieser Methode werden die empfangenen Daten jetzt in Base64 dekodiert, bevor sie weiterverarbeitet werden.

```
sndpkt = pkt
```

Hier wird das Paket vor dem Senden in Bytes (binär) codiert. **Wichtig:** Das **sndpkt** ist das nicht-codierte Paket, da **image_data** bereits Base64-codiert wurde.

Aufruf

Das Receiver Argument bleibt gleich, aber das Sender Argument muss jetzt mit einem Image-Pfad erweitert werden.

```
python3 sender.py --host localhost --port 5003 --dest_host localhost --dest_port
5002 --timeout 2 --image_file image.jpg
```