IT2 Beleg

`Autor: Sebastian Mischke`

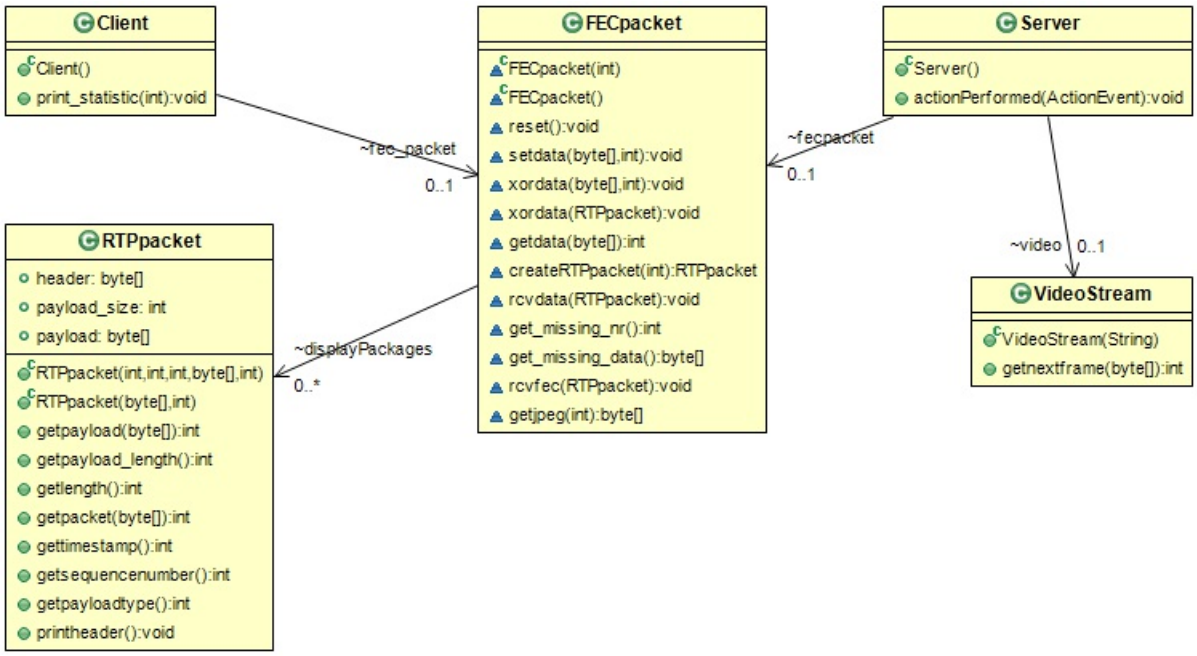
`s-Nummer: s72682`

Aufgabenstellung

Project

Bei dem Projekt handelt es sich um eine Client-Server-Anwendung, die mittels des Real-Time-Streaming-Protokolls (RTSP) einen Videostream überträgt. Dabei werden die eigentlichen Videodaten mittels des Real-Time-Protokolls (RTP) übertragen. Um den Einfluss von verloren gegangenen Paketen zu verringern, wird als Ausfallschutz eine Forward-Error-Correction (FEC) eingesetzt.

Die Klasse `Server` stellt dabei das serverseitig und die Klasse `Client` das clientseitig auszuführende Java-Programm dar.



Server

Die Aufgabe des Servers ist es, aus den Frames einer mjpeg-Datei sowohl RTP-Pakete als auch FEC-Pakete zu erzeugen und diese mittels eines Datagram-Sockets an den Client zu schicken.

Das Auslesen der Frames erfolgt über die Klasse `VideoStream`, welche die JPEG-Bilder als Byte-Array an den Server zurückgibt. Aus diesem Byte-Array wird danach ein RTP-Paket erzeugt, welches anschließend zu einem DatagramPacket umgeformt und verschickt wird.

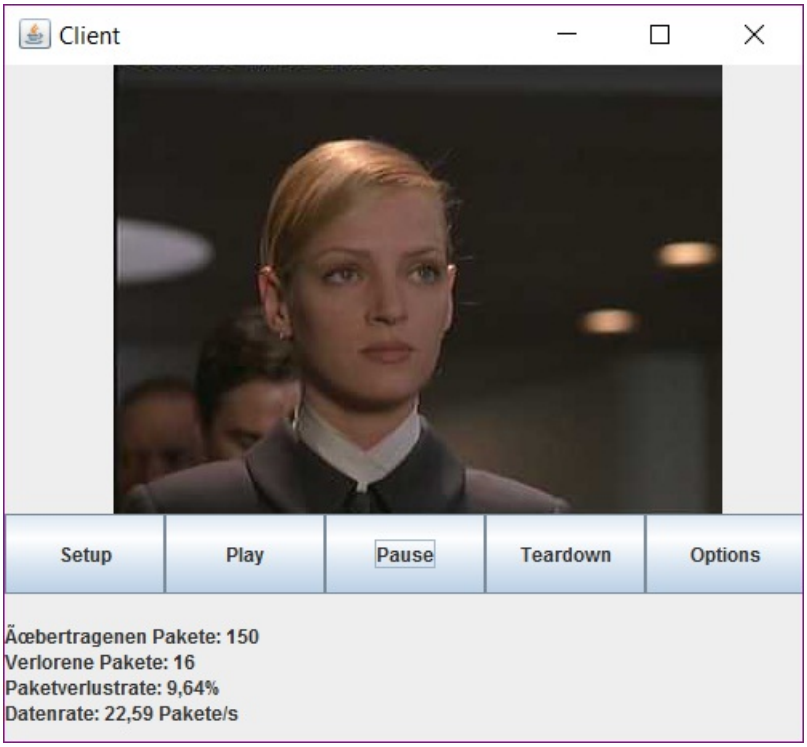
```
int image_length = video.getnextframe(buf);

// Builds an RTPpacket object containing the frame
RTPpacket rtp_packet = new RTPpacket(MJPEG_TYPE, imagenb, imagenb * FRAME_PERIOD, buf, image_length);

// get to total length of the full rtp packet to send
int packet_length = rtp_packet.getlength();

// retrieve the packet bitstream and store it in an array of bytes
byte[] packet_bits = new byte[packet_length];
```

```
rtp_packet.getpacket(packet_bits);

// send the packet as a DatagramPacket over the UDP socket
senddp = new DatagramPacket(packet_bits, packet_length, ClientIPAddr, RTP_dest_port);

RTPsocket.send(senddp);
```

Client

Der Client ist die grafische Bedienoberfläche des Nutzers. Er kann eine Verbindung mit dem Server aufbauen, diesem Befehle wie z.B. `Play` oder `Pause` schicken und empfangene JPEG-Bilder anzeigen. Zusätzlich zeigt er dem Nutzer Statistiken über die bisherigen Übertragungen an.

Im Client existieren zwei Timer. Zum einen der Listener-Timer, der die Pakete des Servers empfängt und auswertet. Zum anderen der Display-Timer, der die empfangenen JPEG-Bilder auf der Benutzeroberfläche anzeigt und die Statistik aktualisiert.



RTPpacket

Das RTPpacket beinhaltet die Daten eines Frames des Videos und besteht aus einem Byte-Array für den Header, welches über die META-Daten des Frames verfügt, und einem Byte-Array für den Payload, welches die Bytes für das JPEG-Bild beinhaltet.

Der Header setzt sich dabei wie folgt zusammen:

```
header[0] = (byte) ((Version << 6) | (Padding << 5) | (Extension << 4) | CC);
header[1] = (byte) ((Marker << 7) | (PayloadType & 0x7F));
header[2] = (byte) (SequenceNumber >> 8);
header[3] = (byte) (SequenceNumber & 0x00FF);
header[4] = (byte) (TimeStamp >> 24);
header[5] = (byte) ((TimeStamp >> 16) & 0x000000FF);
header[6] = (byte) ((TimeStamp >> 8) & 0x000000FF);
header[7] = (byte) (TimeStamp & 0x000000FF);
header[8] = (byte) (Ssrc >> 24);
header[9] = (byte) ((Ssrc >> 16) & 0x000000FF);
header[10] = (byte) ((Ssrc >> 8) & 0x000000FF);
header[11] = (byte) (Ssrc & 0x000000FF);
```

FECpacket

Das FECpacket beinhaltet die Funktionen für die Forward-Error-Correction (FEC). Zum einen erlaubt es dem Server ein UDP-Paket zu erstellen, welches die Informationen eines FEC-Paketes besitzt, zum anderen bietet es dem Client die Möglichkeit, ein verloren gegangenes RTP-Paket wiederherzustellen.

Serverseite

Jedes mal, wenn der Server einen neuen Frame zum versenden vorbereitet, werden diese Bytes auch an das FECpacket gegeben. Das FECpacket verlängert zuerst, falls notwendig, die Länge des Byte-Puffers und XOR-verknüpft die bekommen Bytes des nächsten Frames und die bereits im FECpacket vorhandenen Bytes.

```
void xordata(byte[] data, int data_length) {
  if (data_length > this.data.length) {

    // Create new data-array
    byte[] newdata = new byte[data_length];

    // Fill the new data-array with the old data
    for (int i = 0; i < this.data.length; i++) {
      newdata[i] = this.data[i];
    }

    // Set newdata as this.data
    this.data = newdata;
    this.data_size = data_length;
  }

  // XOR param-data-array with new data-array
  for (int i = 0; i < data_length; i++) {
    this.data[i] = (byte) (this.data[i] ^ data[i]);
  }
  packages++;
}
```

Wenn das FECpacket soviele Frames bekommen hat, wie vorher in der FEC_group festgelegt worden, dann wird ein neues UDP-Paket erstellt, welches die Daten des FECpacket enthält. Dabei wird die FEC_group als erstes Byte vor das Daten-Array gestellt. Anschließend wird das FECpacket resetet.

```
if (fecpacket.packages == k) {
  // Create UDPpacket from FECpacket
  RTPpacket fec_packet = fecpacket.createRTPpacket(imagenb);

  // Get data from UDPpacket
  int fec_length = fec_packet.getlength();
  byte[] fec_bits = new byte[fec_length];
  fec_packet.getpacket(fec_bits);

  // Send UDPpacket
  senddp = new DatagramPacket(fec_bits, fec_length, ClientIPAddr, RTP_dest_port);
  RTPsocket.send(senddp);

  // Create new FECpacket
  fecpacket = new FECpacket(k);
}
```

Clientseite

Wenn im Client ein RTP-Paket ankommt, wird zunächst geprüft, ob seine Sequenznummer größer ist, als die Sequenznummer

des letzten empfangenen Pakets. Ist dies der Fall, wird es im FECpacket in einer ArrayList gespeichert. Zusätzlich wird dessen Sequenznummer in einer anderen ArrayList hinzugefügt und seine `Payload`-Daten werden mit den bereits im FECpacket vorhandenen Daten XOR-verknüpft.

```
void rcvdata(RTPpacket rtppacket) {
  if (rtppacket.getsequencenumber() > lastSqNr) {
    rtp_nrs.add(rtppacket.getsequencenumber());
    displayPackages.add(rtppacket);
    packages++;
    xordata(rtppacket);
    lastSqNr = rtppacket.getsequencenumber();
  }
}
```

Erhält der Client ein FEC-Paket, so wird zunächst das erste Byte des Daten-Arrays als Größe der FEC_group genommen. Die restlichen Bytes werden mit den bereits im FECpacket vorhandenen Daten XOR-verknüpt. Anschließend wird geprüft, ob alle RTP-Pakete, die im "Zuständigkeitsbereich" des empfangenen FEC-Pakets liegen, vorhanden sind. Sollte genau ein RTP-Paket fehlen, so wird dieses aus den Daten, die sich momentan im FECpacket befinden, rekonstruiert und an die richtige Stelle in der ArrayList eingesetzt. Zum Schluss wird das FECpacket auf seine Startwerte zurückgesetzt.

```
void rcvfec(RTPpacket rtp) {
  // get FEC_group from first data element
  FEC_group = rtp.payload[0];

  // data is payload without first element
  byte[] newdata = Arrays.copyOfRange(rtp.payload, 1, rtp.getpayload_length());
  data_size = rtp.getpayload_length() - 1;
  xordata(newdata, data_size);

  to_frame = rtp.getsequencenumber();

  // check if last packages are complete
  checkDisplaylist();

  // reset data
  reset();
}
```