Praktikum RTSP-Streaming

1. Projektrahmen

Im Praktikum werden Sie einen Client und Server für Videostreaming unter Nutzung des Real-Time-Streaming-Protokolls (RTSP) implementieren. Die eigentlichen Videodaten werden mittels Real-Time-Protokoll (RTP) übertragen. Ein großer Teil der Funktionalität ist bereits als Quelltext vorhanden, so das RTSP-Protokoll im Server, das RTP-Handling im Client sowie die Videoanzeige.
Ihre Aufgabe besteht im Wesentlichen aus der Ergänzung der Quellcodes in den Punkten:

- RTSP-Protokoll im Client
- RTP-Protokoll im Server
- FEC in Client und Server

1.1. Javaklassen

Das Projekt besteht aus folgenden Java-Klassen:

Client Funktionalität des Clients mit Benutzerschnittstelle zum Senden der RTSP-Kommandos und Anzeige des Videos.

Server Funktionalität des Servers zur Antwort auf die RTSP-Clientanfragen und Streaming des Videos.

RTPpacket Funktionalität zur Unterstützung von RTP-Paketen.

VideoStream Funktionalität zum Einlesen einer MJPEG-Datei auf der Serverseite

1.2. Programmstart

Der Start des Servers erfolgt mittels `java Server RTSP-Port` Der Standard-RTSP-Port ist 554, Sie werden aber im Praktikum einen Port > 1024 nutzen.
Der Start des Clients erfolgt mittels `java Client server_name server_port video_file`
Am Client können RTSP-Kommandos angefordert werden. Eine Kommunikation läuft in der Regel folgendermaßen ab:

1. Client sendet SETUP: Erzeugung der Session und der Transportparameter
2. Client sendet PLAY
3. Client sendet u.U. PAUSE
4. Client sendet TEARDOWN: Terminierung der Session.

Der Server antwortet auf alle Clientrequests. Die Antwortcodes sind ähnlich zu HTTP. Der Code 200 bedeutet z. B. Erfolg. Die Codes finden Sie in [3].

1.3. Client

Als ersten Schritt sollte das RTSP-Protokoll in den Handlern der Buttons der Benutzerinterfaces vervollständigt werden. Für die RTSP-Kommunikation mit dem Server wird der bereits geöffnete Socket verwendet. In jeden Request muss ein CSeq-Header eingefügt werden. Der Wert von CSeq erhöht sich bei jedem Senderequest.

1.3.1. SETUP

- Erzeugen eines Sockets für den Empfang der RTP-Pakete und setzen des Timeouts (5 ms)
- Senden des SETUP-Requests an den Server, Ergänzung des Transportheaders mit dem geöffneten RTP-Port.
- Einlesen der RTSP-Antwort vom Server und parsen des Sessionheaders für die Session-ID

1.3.2. PLAY

- Senden des PLAY-Requests mit Sessionheader und Session-ID (kein Transportheader).
- Einlesen der RTSP-Antwort

1.3.3. PAUSE

- Senden des PAUSE-Requests mit Sessionheader und Session-ID
- Einlesen der RTSP-Antwort

1.3.4. TEARDOWN

- Senden des TEARDOWN-Requests mit Sessionheader und Session-ID
- Einlesen der RTSP-Antwort

1.3.5. Beispiel

Im Praktikum wird ein sehr **einfacher Parser** im Server verwendet, welcher die Daten in einer bestimmten Reihenfolge erwartet. Bitte orientieren Sie sich an den folgenden Beispiel (C-Client, S-Server). Insbesondere sind die Leerzeichen zu beachten (client_port) und die Request-URL darf nur relativ sein (ohne rtsp://host).

```
C: OPTIONS movie.Mjpeg RTSP/1.0
 : CSeq: 1

S: RTSP/1.0 200 OK
 : CSeq: 1
 : Public: DESCRIBE, SETUP, TEARDOWN, PLAY, PAUSE

C: SETUP movie.Mjpeg RTSP/1.0
 : CSeq: 1
 : Transport: RTP/UDP; client_port= 25000

S: RTSP/1.0 200 OK
 : CSeq: 1
 : Session: 123456

C: PLAY movie.Mjpeg RTSP/1.0
 : CSeq: 2
 : Session: 123456

S: RTSP/1.0 200 OK
 : CSeq: 2
 : Session: 123456
```

```
C: PAUSE movie.Mjpeg RTSP/1.0
 : CSeq: 3
 : Session: 123456

S: RTSP/1.0 200 OK
 : CSeq: 3
 : Session: 123456

C: TEARDOWN movie.Mjpeg RTSP/1.0
 : CSeq: 4
 : Session: 123456
```

1.3.6. Zustände des Clients

Im RTSP-Protokoll hat jede Session einen bestimmten Zustand. Sie müssen den Zustand des Clients entsprechend aktualisieren.

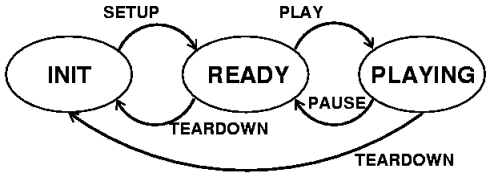

Abbildung 1: RTSP-Zustände

1.4. Server

Auf Serverseite muss das Einbetten der Videodaten in die RTP-Pakete erfolgen. Die beinhaltet das Erzeugen des Paketes, Setzen der Headerfelder und setzen der Payload.
Für Informationen zur Bitmanipulation in Java siehe Vorlesungsfolien zu RTP.

Byte 0								Byte 1								Byte 2								Byte 3							
Bit 0	1	2	3	4	5	6	7	Bit 0	1	2	3	4	5	6	7	Bit 0	1	2	3	4	5	6	7	Bit 0	1	2	3	4	5	6	7
V=2			P	X	CC			M		PT						Sequence Number																
Timestamp (in sample rate units)																															
Synchronization Source (SSRC) identifier																															
Contributing Source (CSRC) identifiers (optional)																															
Header Extension (optional)																															

Abbildung 2: RTP-Header

2. Teilaufgaben

2.1. RTSP-Protokoll im Client

Ergänzen Sie die Quelltexte entsprechend der in der Aufgabenbeschreibung und im Quelltext vorhandenen Hinweise.

2.2. RTP-Protokoll im Server

Ergänzen Sie die Quelltexte der Klasse RTPpacket entsprechend der in der Aufgabenbeschreibung und im Quelltext vorhandenen Hinweise.

2.3. RTSP-Methode Options

Ergänzen Sie die Methode entsprechend der Beispiele aus RFC 2326 und RFC 5109.

2.4. Simulation von Paketverlusten

Testen Sie den Client mit dem beigefügten Server (Klasse FunkyServer). Was stellen Sie fest und wie kann das Problem prinzipiell gelöst werden? Simulieren Sie Paketverluste und eine variable Verzögerung im Netz, indem Sie am Sender eine wahlweise Unterdrückung von zu sendenden Paketen vornehmen. Diese Unterdrückung von Paketen sollte zufällig und mit einstellbarer Paketverlustwahrscheinlichkeit über das GUI erfolgen. Beispiel: der Wert 0,1 bedeutet, es werden im Mittel 10% der zu übertragenen Pakete unterdrückt.

2.5. Anzeige von Statistiken am Client

Anzeige von Statistiken am Client (Zeigen Sie die aktuelle Gesamtzahl an übertragenen Paketen und Paketverlusten sowie die Paketverlustrate und die Datenrate, aktualisieren Sie die Ergebnisse jede Sekunde). Weitere Statistiken können entsprechend Ihren Vorstellungen hinzugefügt werden.

2.6. Implementierung eines FEC-Schutzes

Implementieren Sie einen FEC-Schutz mittels Parity-Check-Code (XOR mit $k = 2 \ldots 20, p = 1$). Der Parameter sollte am Server einstellbar sein (GUI / Kommandozeile) Der Server mit FEC-Schutz soll kompatibel zu Clients ohne FEC-Verfahren sein! Nutzen Sie dazu das Feld Payloadtype des RTP-Headers (PT=127 für FEC-Pakete).
Sie können sich bei der Implementierung an RFC 5109 orientieren, dies ist aber keine Pflicht. Sie sollten aber das Dokument zumindest lesen.
Implementierung Sie FEC über nachfolgende Schritte:

1. Nutzung einer separaten Klasse FECpacket für das FEC-Handling für Sender und Empfänger, siehe Ausführungen weiter unten
2. Serverseitige Implementierung des XOR-FEC. Nach Auswertung des PT (26) sollte der Client nach wir vor regulär funktionieren.
3. Entwurf der Architektur der Paket- und Bildverarbeitung im Client
4. Eingangspuffer im Client implementieren (Größe ca. 1-2 s)
5. FEC-Korrektur im Client implementieren

Ändern Sie die vorhandenen Klassen nur soweit **nötig**. Halten Sie die Struktur einfach und überschaubar. Nutzen Sie Threads nur wenn dies notwendig ist und Sie die Nebenwirkungen (Blockierungen, Race-Condition) kennen. Bei einer durchdachten Implementierung benötigen Sie keine Threads.

2.6.1. Architekturvorschlag

Die nachfolgenden Ausführung dienen nur zur Orientierung. Sie können auch einen anderen Ansatz wählen.

```
Klasse FECpacket
  int FEC_group  // Anzahl an Medienpaketen für eine Gruppe

  // Sender
  void setdata(byte[] data, int data_length) // nimmt Nutzerdaten entgegen
  int  getdata(byte[], data) // holt FEC-Paket (Länge -> längstes Medienpaket)

  // Empfänger
  // getrennte Puffer für Mediendaten und FEC
  // Puffergröße sollte Vielfaches der Gruppengröße sein
  void rcvdata(int nr, byte[] data) // UDP-Payload, Nr. des Bildes bzw. RTP-SN
  void rcvfec(int nr, byte[] data) // FEC-Daten
  byte[] getjpeg(int nr)  // übergibt korrigiertes Paket oder Fehler (null)
```

Verarbeitung im Server:

- der Server kann die gesamte Verarbeitung im vorhandenen Timer-Handler vornehmen
- Nutzdaten speichern (`FECpacket.setdata()`)
- Nutzdaten senden
- nach Ablauf des Gruppenzählers berechnetes FEC-Paket entgegennehmen und senden (`FECpacket.getdata()`)
- Kanalemulator jeweils für Medien- und FEC-Pakete aufrufen

Verarbeitung im Client:

- Der Client könnte z.B. mit der doppelten Timerrate laufen und dann alle Verarbeitung im Timer-Handler vornehmen. Damit kann auf Threads verzichtet werden.
- Pakete empfangen und speichern (`FECpacket.rcvdata()` bzw. `FECpacket.rcvfec()`)
- Statistiken aktualisieren
- zur richtigen Zeit (Timeraufruf) das nächste Bild anzeigen (`FECpacket.getjpeg()`), Verzögerung des Starts (1-2s), um den Puffer zu füllen

2.6.2. Parameterwahl

Finden Sie den optimalen Wert für k bei einer Kanalverlustrate von 10%. Optimal bedeutet in diesem Fall eine zufriedenstellende Bildqualität bei geringstmöglicher Redundanz.

2.6.3. Dokumentation

Dokumentieren Sie Ihr Projekt. Beschreiben Sie die Architektur Ihrer Implementierung anhand sinnvoller Softwarebeschreibungsmethoden (Klassendiagramm, Zustandsdiagramm, etc.). Eine Quellcodekommentierung ist dazu nicht ausreichend!

2.7. RTSP-Methode Describe (Optional)

Unterstützen Sie die o.g. Methode für das Beispielvideo.

3. Literatur

1 Kurose, Ross "Computernetzwerke", Pearson

2 http://openbook.galileocomputing.de/javainsel/

3 www.ietf.org/rfc/rfc2326.txt (RTSP)

4 www.ietf.org/rfc/rfc3550.txt (RTP)

5 www.ietf.org/rfc/rfc2435.txt (RTP - MJPEG)

6 www.ietf.org/rfc/rfc2327.txt (SDP)

7 www.ietf.org/rfc/rfc5109.txt (RTP for Generic FEC)