Dokumentation des IT2 Belegs

Einleitung

Im Rahmen des Moduls IT2 habe ich mich intensiv mit der Entwicklung eines RTSP-Streaming-Systems beschäftigt. Der Beleg war in mehrere Aufgaben unterteilt, und jede hat mir geholfen, Neues zu lernen und meine Programmierkenntnisse zu vertiefen. Im Folgenden beschreibe ich, was ich gemacht habe und welche Erkenntnisse ich dabei gewonnen habe.

Aufgabenbearbeitung

0. Vorarbeiten

Zu Beginn habe ich das Projekt aus dem Repository heruntergeladen und die notwendigen Abhängigkeiten in meiner Entwicklungsumgebung eingerichtet. Danach erstellte ich die Klassenrümpfe der von den abstrakten Klassen abgeleiteten leeren Klassen, was kein Problem darstellte. Zu guter Letzt habe ich noch die Konfigurationen für den Server und den Client eingerichtet und das Testvideo in das Projekt eingefügt.

1. RTSP-Protokoll: Client-Methoden

Die erste große Herausforderung war die Implementierung der RTSP-Funktionalität. Besonders die Methode send_RTSP_request() war spannend, weil ich hier genau verstehen musste, wie das Protokoll funktioniert und wie die Nachrichten aufgebaut sind. Um sicherzugehen, dass alles korrekt ist, habe ich die Konsolenausgaben überprüft.

Mein Fazit: Protokolle wie RTSP sind präzise aufgebaut, und jeder kleine Fehler könnte zum Problem werden.

2. SDP-Protokoll

Hier ging es darum, die Methode DESCRIBE für das SDP-Protokoll umzusetzen. Ich habe die relevanten Parameter, wie die Framerate und die Range, auf der Serverseite statisch hinterlegt.

Mein Fazit: Obwohl diese Aufgabe relativ einfach und schnell erledigt war, hat es mir gezeigt, wie Metadaten zwischen Server und Client übertragen werden.

Phileas Granow 85301

3. RTP-Protokoll

Für das RTP-Protokoll habe ich die Methode setRtpHeader() programmiert. Dabei musste ich sicherstellen, dass der Header korrekt aufgebaut ist, damit das Video abgespielt werden kann. Da ich anfangs Probleme hatte, den Aufbau des Headers zu verstehen, habe ich mir von ChatGPT helfen lassen, die ersten Bytes zu füllen, worauf ich dann keine Probleme mehr hatte, die restlichen Bytes selbstständig zu vervollständigen.

Mein Fazit: Nachdem ich die Header-Struktur verstanden hatte, war diese Aufgabe relativ intuitiv zu lösen. Hier hatte ich das erste Erfolgserlebnis, als das Video vollständig abgespielt wurde.

4. Fehlerstatistiken ohne Fehlerkorrektur

Bei dieser Aufgabe habe ich getestet, wie sich unterschiedliche Fehlerwahrscheinlichkeiten auf die Videoqualität auswirken. Dabei habe ich festgestellt, dass die Bildqualität bereits bei einer Kanalverlustrate von etwa 2% spürbar schlechter wird und ab ca. 10% sehr stark nachlässt. Außerdem habe ich die Wahrscheinlichkeit für den Verlust eines Bildes in Abhängigkeit von der Kanalverlustrate, wenn pro Bild 1, 2, 5, 10 oder 20 RTPs versendet werden berechnet und mittels Gnuplot visualisiert (siehe " RTSP-Streaming\statistics\rtp-fec_Aufgabe_4_und_6-2.gp")

Mein Fazit: Je mehr Pakete ein Bild benötigt, desto anfälliger wird es für Verluste – das macht Fehlerkorrektur umso wichtiger.

5. Implementierung des FEC-Schutzes

Die Implementierung des FEC-Schutzes war eine der anspruchsvollsten Aufgaben. Ich musste verstehen, wie die Datenpakete gruppiert und geschützt werden, und habe die fehlenden Funktionen im FecHandler ergänzt. Dabei haben mir für das Verständnis die FEC-Diagramme und -Tabellen auf der IT2-Website geholfen. Der FEC-Schutz hat am Ende gut funktioniert und konnte verlorene Pakete wiederherstellen.

Mein Fazit: Die Arbeit mit FEC war eine Herausforderung, aber auch sehr spannend, weil ich gesehen habe, wie man Netzwerkfehler ausgleichen kann.

6. Analyse der Leistungsfähigkeit des FEC-Verfahrens

Ich habe getestet, wie gut FEC in verschiedenen Szenarien funktioniert, und die Ergebnisse mit Gnuplot grafisch dargestellt. Dabei konnte ich erkennen, wie die Gruppengröße und die Kanalfehlerrate die Effektivität von FEC beeinflussen. (siehe "RTSP-Streaming/statistics")

Mein Fazit: Mit FEC lassen sich Verluste deutlich reduzieren, aber die richtigen Einstellungen sind entscheidend.

7. Generierung von Restart-Markern

Ich habe im Praktikum mit Restart-Markern in einem JPEG-Bild experimentiert. Durch die Bearbeitung mit jpegtran und einem Hexeditor konnte ich sehen, wie diese Marker Fehler in Bildern auffangen und stabiler machen. Ohne Marker hat es schon gereicht, nur ein paar Bits eines Bildes im Hexeditor zu verändern, um das gesamte Bild unbrauchbar zu machen. Nach dem Erzeugen von Restart-Markern war das Bild deutlich robuster und fehlerunanfälliger.

Mein Fazit: Restart-Marker sind eine wirkungsvolle Möglichkeit, Fehler in Bildern zu reduzieren.

8. Fehlerkaschierung

Zum Schluss habe ich eine Fehlerkaschierung umgesetzt, um beschädigte Bilder zu verbessern. Dabei wurde das aktuelle Bild mithilfe des vorherigen Bildes und Transparenz repariert. Diese Methode hat sich besonders bei höheren Fehlerwahrscheinlichkeiten bewährt, bei denen das Video sonst schon nicht mehr brauchbar gewesen wäre.

Mein Fazit: Selbst wenn nicht alle Fehler korrigiert werden können, lässt sich die sichtbare Qualität mit Fehlerkaschierung stark verbessern.

Fazit

Die Bearbeitung dieses Belegs hat mir gezeigt, wie vielschichtig die Entwicklung von Streaming-Systemen ist. Ich habe nicht nur viel über die Protokolle RTSP und RTP gelernt, sondern auch, wie man Netzwerkfehler analysiert und kompensiert. Besonders spannend war für mich zu sehen, wie Theorie und Praxis ineinandergreifen. Der Beleg war insgesamt relativ herausfordernd, aber durch den gezielten Einsatz von ChatGPT und das Diskutieren und Vergleichen von Aufgaben mit Kommilitonen, konnte ich diesen Meilenstein gut meistern.

Phileas Granow 85301