Ansätze zum Echtzeit-Video-Streaming im Web

Maximilian Schulke *(Matrikel-Nr. 20215853)*
Technische Hochschule Brandenburg
B.Sc. Medieninformatik
Computergrafik

betreut durch Prof. Dr. rer. nat. Reiner Creutzburg
Wintersemester 2021
Abgabetermin 24. November 2021

***Zusammenfassung*—Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Das abstract schreibe ich zu letzt!**

Inhaltsverzeichnis

I Einleitung / Motivation 1

II Historie der Echzeit-Übertragung 1
II-A 1996: RTP & RTCP – RFC 1889 . 2
II-B 2004: SRTP (RFC 3711) 2
II-C 2005: RTMP von Adobe 2
II-D 2010: WebRTC (RFC 8825) 2

III Architekturmuster 2
III-A Peer-To-Peer 2
III-A1 Signaling Server 3
III-A2 Holepunching 3
III-A3 IP-Multicast 3
III-B Relay 3

IV Protokolle 3
IV-A RTP 3
IV-B RTCP 3
IV-C RTSP 3
IV-D SDP 3
IV-E SID 3
IV-F STUN 3
IV-G TURN 3
IV-H WebRTC 3

V Implementierung eines Live-Streams 3
V-A Ziel 3
V-B Inhalt des MVP 3
V-C Architektur 3
V-D Signaling Server 4

VI Benchmarks 4

VII Auswertung 4

VIII Nächste Schritte 4

IX Abbildungsverzeichnis 4

Literatur 4

I. Einleitung / Motivation

Mit der zunehmenden Vernetzung der Arbeitswelt in den letzten Jahren quelle suchen corona zoom werden auch digitale Meeting-Systeme immer relevanter und müssen immer mehr Nutzer in Echtzeit mit einander verbinden um einen reibungslosen Arbeitsalltag zu gewährleisten. Dies impliziert natürlich auch dass eine quelle suchen ab wann gute Qualität Verbindungsqualität gegeben sein muss, damit die Systeme nutzbar bleiben.

& sicherheitsaspekt Aber wie können wir skalierbare Meeting-Systeme realisieren ohne große Datenmengen über einen Streaming-Server zu schicken der diese an alle anderen broadcasted? Die Entwicklung der letzten Jahre deuten immer mehr darauf hin, dass *Peer-To-Peer* basierte Lösungensansätze aufgrund der besseren Performance und Skalierbarkeit, in der Regel die bessere Wahl darstellen quelle suchen. Natürlich spielen zur Auswahl der Architektur noch weitere Parameter eine wichtige Rolle (z. B. die maximale Bandbreite und Rechenleistung der Endgeräte), aber mit immer großer werdenden Heimnetz-Leitungen und zunehmender Rechenleistung der Endgeräte stellt dies meistens kein Problem mehr da. quelle suchen.

Die Problematik der Echtzeit-Kommunikation im Web beschäftigt auch das *W3C* seit 2011 im Zuge der Standardisierung des seit diesem Jahr zum Web-Standard erklärten Protokoll *WebRTC*. quelle suchen.

Es ist also (immer noch) eine sehr aktuelle Thematik in der Informatik Echtzeit- oder Nahe-Zu-Echtzeit-Kommunikation zuverlässig zu bewältigen. Die Aufgabe dieser Arbeit soll sein, einen Überblick über den Stand der Architekturmuster, Protokolle und möglicher Problematiken bei der Implementierung von eigenen Echtzeit-Video-Streaming-Diensten geben, diese anhand eines Experiments durchsprechen und auswerten usw.

II. Historie der Echzeit-Übertragung

Um zu verstehen wie sich die Protokolle hin zum heutigen Stand entwickelt haben, ist es besonders interessant nachzuvollziehen wie die ersten Schritte der IEFT oder auch *Internet Engineering Task Force* bezüglich der Echtzeit-Kommunikation aussahen, welche Probleme erkannt und behoben wurden und welche Protokolle heute der Standard sind.

A. 1996: RTP & RTCP – RFC 1889

Am weitesten reicht das *Real-Time Transport Protocol* oder auch *RTP* zurück. Es wurde erstmals 1996 von der IEFT standardisiert und stellt seit dem einen Grundbaustein der datenformatsaganostischen Echtzeit-Übertragung da. Es kann für diverse Echtzeit-Übertragungs-Problematiken dienlich sein, da jegliche Binärdaten verschickt werden können; Somit gibt es keinen "Lock-In" auf bestimmte Audio- oder Video-Codecs. aussage prüfen und quelle.

RTCP ist das mit *RTP* einhergehende Kontrollprotokoll. Es wird primär dazu verwendet, die Übertragungsparameter der Sender zu beeinflussen – z. B. durch ein Feedback zur Übertragungsqualität oder ein Abmelden der Session. Desweiteren bietet es eine persistente ID für die *RTP*-Mitglieder, die über Programm-Neustarts hinweg zur Identifikation von Mitgliedern und der zuordnung von Datenströmen verwendet werden können. cite rfc1889 kapitel 6 / 6.1 bzw. S. 15-17

Das Protokoll siedelt sich im TCP/IP-Stack über *UDP* an cite rfc1889 introduction. Es fügt wichtige Informationen zu UDP-Datagrammen hinzu: Im wesentlichen eine Sequenznummer um die Sende-Reihenfolge zu codieren und einen Payload-Type, der den Codec des Segments angibt. Somit kann auch bei nicht sequenziell übertragenden Datagrammen die Ursprüngliche Reihenfolge rekonstruiert werden und es können bei verlorenen Segmenten Interpolationsalgorithmen verwendet werden. quelle / beispiele suchen. Der Payload-Type ist essenziell um ohne Session-Aushandlung zu kommunizieren wie der Empfänger die Daten zu decodieren hat, um eine Sinnvolle nachricht zu erhalten.

In der ersten Version aus 1996 gab es einige Probleme bezüglich Probleme suchen. Diese wurden 2003 in dem RFC3550 überarbeitet - somit wurde das RFC1889 durch die neuere Version 3550 obsolet. In der aktuelleren Version wurden einige Änderungen eingearbeit: Im wesentlichen "RTCP Packet Send and Receive Rules" "Layered Encodings" "Congestion Control" "Security Considerations" "IANA Considerations" Dummer text.

Eine wichtige Voraussetzung zur Verwendung von *RTP* ist ein externer *Signaling-Server*, den alle Beteiligten zur Session-Aushandlung verwenden. Ein Standard hierfür ist im *RTP-Framework* selbst nicht definiert, eine beliebte Wahl für ein Protokoll zur Session-Aushandlung ist allerdings das *Session Initiation Protokoll* (oder kurz (SIP)).

B. 2004: SRTP (RFC 3711)

Im März 2004 wurden *SRTP* und *SRTCP* vorgestellt – die verschlüsselten Version von *RTP* bzw. *RTCP*. Diese bauen weitestgehend auf dem *Advanced Encryption Standard* (kurz. *AES*) auf (siehe RFC3711 Seite 19).. Der RFC beschäftigt sich weitestgehend mit den kryptografischen Aspekten des Protokolls.

Eine weitverbreitete und offene Implementierung für *SRTP / SRTCP* wird von der US-Amerikanischen Telekomunikations-Gesellschaft Cisco bereitgestellt. Diese hat den Namen *libsrtp* und ist öffentlich auf der Entwickler-Plattform GitHub einsehbar (siehe github.com/cisco/libsrtp).

C. 2005: RTMP von Adobe

RTMP ist ein von der Firma *Adobe Systems Incorporated* spezifiziertes Protokoll zur Echtzeit-Übertragung von Multimedia-Streams. *RTMP* wurde laut Spezifikationsiehe RTMP spec 1.0, entwickelt um über dem Transport-Protokoll *TCP* verwendet zu werden.

Das Protokoll wurde dazu entwickelt um im Kontext eines Flash-Players verwendet zu werden. Dies führt dazu, dass es heutzutage nurnoch bedingt Anwendung findet, da mittlerweile viele Browser ihren Flash-Player-Support eingestellt haben (siehe Chrome und Firefox)

Außerdem spricht die hohe Latenz von bis zu 30 Sekunden, die durch die Verwendung von *TCP* zustandekommt (siehe restram.io/streaming-protocls), gegen den Einsatz von *RTMP* in einem Echtzeit-Übertragungs-Kontext.

Desweiteren gibt es noch Protokollvarianten die HTTP bzw. HTTPS als zugrundeliegendes Protokoll verwenden.Siehe XYZ Quelle suchen

D. 2010: WebRTC (RFC 8825)

WebRTC ist eine Peer-To-Peer-Technologie die über mehrere Protokolle und Audio- und Video-Codecs performante und generische Echtzeit-Kommunikations-Kanäle zwischen Nutzern (oder auch *Peers*) realisiert. Sie

WebRTC stellt eine direkte Verbindung zwischen zwei Endgeräten her und verwendet einen mix aus RTP, RTCP, SDP, (ICE Candidates) und einem signaling server. Es unterstützt beliebte video codecs wie V8 / V9 und ist in der zukunft auf AV1 angelegt. Der standard audio codec ist opus.

Durch die direkte verbindung wird eine sogennante sub second latency erreicht. Dies beschreibt im grunde XYZ.

Die ersten *Requests for Comments* oder auch *RFC* zu den grundlegenden Protokollen *RTP* und *Protokoll X*, auf denen die heutigen Protokolle weitestgehend aufbauen, wurden bereits in den späten 1990er Jahren veröffentlicht und seit dem immer weiterentwickelt. quelle anhängen RFC35XX.

III. Architekturmuster

In diesem Kapitel werden zwei der bekanntesten Architekturmuster in der Echtzeit-Übertragung vorgestellt und verglichen. Es geht primär um die verteilte Peer-To-Peer Architektur und die zentralisierte Relay / Broadcast Architektur.

A. Peer-To-Peer

Auf Grund der hohen Anforderungen an möglichst niedrige Übertragungslatenzen bietet eine Peer-To-Peer-Architektur klare Vorteile durch den stark verkürzten Weg, den die Pakete zurücklegen müssen bis sie bei dem Empfänger ankommen.

Deutlich komplizierter wird allerdings die aushandlung bzw. initialisierung einer Verbindung zwischen zwei peers,

da diese sich nicht wie bei der typischen client-server-architektur eine fixe adresse zur verbindung haben. Dieses problem wird typischerweise über einen signaling server gelöst.

1) Signaling Server: Ein signaling server ist allen peers bekannt und dient als kommunikationsplattform, damit sich die beiden (sich gegenseitig initial unbekannten) peers gegenseitig vorstellen können.

Signaling server sind interessanterweise keine feste anforderung für peer to peer muster. Wenn alle peers immer statische addressen hätten, könnten sie auch offline ihre ips / ice candidates / sdp offers etc. austauschen und so eine session aufbauen. Wichtig ist leidlgich eine initiale out of band kommunikation.

Der signaling server kann außerdem noch nach verbindungsaufbau dazu verwendet werden um die bestehende verbindung zu optimieren. Peers können neue ICE candidates vorschlagen um die Übertragung an neue gegebenheiten im internet (z.B. ausfall eines routers) anzupassen. (siehe mdn)

2) Holepunching: Holepunching beschreibt den prozess lokale firewalls und nats zu durchbrechen um eine direkte verbindung zwischen zwei endgeräten herzustellen.

3) IP-Multicast: Je mehr Nutzer / Endgeräte an einer peer-to-peer-übertragung teilnehmen, desto größer wird die Belastung der Bandbreite bei den einzelnen Teilnehmern. Jedes Paket muss nicht einmal, sondern n-mal verschickt werden (wobei n die anzahl der teilnehmer ist). Dies kann bei labilen oder einfach schwachen Netzwerken entweder zur verminderung der übertragungsqualität führen, oder in besonders schlimmen fällen sogar das restliche netzwerk eines einzelnen Teilnehmers negativ beeinflussen, da dieser die meiste Bandbreite für die wiederholte übertragung gleicher pakete verwendet.

Eine theoretische Lösung für dieses Problem, ist die Verwendung von IP-Multicast-Adressen. Diese erlauben es einem Gerät ein IP-Paket einmalig zu übertragen, und dieses von Multicast-Routern im internet multiplizieren zu lassen. (Siehe grafik)

Vergleichs grafik einfügen..

Dieser ansatz hat klare vorteile: Das gesamte - lokale & globale – netzwerk wird geschont. So würde die Anzahl möglicher Teilnehmer deutlich steigen, da ein Netzwerk-Bottleneck erst bei einem zu großen Download-Volumen des empfangenen Contents eintreten würde.

B. Relay

IV. Protokolle

A. RTP

B. RTCP

C. RTSP

D. SDP

E. SID

F. WebRTC

V. Implementierung eines Live-Streams

In diesem Kapitel wird ein experimentelles Kamera-System entwickelt, das die oben erläuterten Protokolle / Technologien verwendet.

A. Ziel

Ziel dieses Experiments ist es eine möglichst einfach ein funktionierendes System zu entwickeln und mögliche Probleme, benötigten Zeitaufwand usw. zu ermitteln. Desweiteren ist ein “Performance” Vergleich mit anderen Live-Stream systemen interessant. Es gilt also die frage zu klären, wie viel arbeit benötigt ein grundsätzlich funktionierendes Software-Produkt das auf Echtzeit-Übertragung beruht.

B. Umfang

Das zu implementierende Kamera-System wird lediglich eine einzige Kernfunktion aufweisen: Sobald die Kamera an ist, streamt sie via WebRTC, mit ausreichend FPS (mindestens 15 im durchschnitt) in ausreichender Qualität (720 x 480), ihre Aufnahmen an ein User Interface. Dieses wird im browser laufen, muss aber ausreichend Alternativen für andere Plattformen aufweisen (Nativ, Smartphone usw.). Das Interface und die Kamera starten die Aushandlung des WebRTC-Streams über den Signaling Server. D.h. sie versenden SDP (Offer / Answer) und tauschen ihre ICE-Candidates aus.

C. Architektur

Die Architektur ist sehr simpel. Sie besteht lediglich aus 3 Kernelementen:

1) Hardware mit einer Kamera: Die Hauptanforderung an die Hardware auf der die Kamera-Software läuft ist eine Linux-Installation (mit den entsprechenden installierten Paketen für die Abhangigkeiten) und eine angeschlossene Kamera die von Linux erkannt wird.

Für dieses Experiment wurde ein Einplatinencomputer der Marke Raspberry PI in der 4. Version mit 8GB RAM – erweitert durch ein Drittanbieter-Kamera-Modul – verwendet. Dieser ist allerdings absolut austauschbar, da jeder Rechner der die o.g. Hauptanforderung erfüllt eine geeignete Umgebung darstellt – so kann die Kamera-Software auch auf einem Laptop ausgeführt werden, falls keine Hardware verfügbar ist. Dies wurde zur veranschaulichung mit einem ThinkPad T490 und einer aktuellen NixOS Installation getestet.

Auf der Einplatinencomputer ist das mitgelieferte Betriebssystem “Raspberry PI OS” und die entsprechenden Software-Abhängigkeiten installiert. (Siehe GitHub für Dependencies.)

2) Signaling-Server: Der Signaling-Server ist ein Secure-WebSocket-Server, der zwei WebSocket-Verbindungen miteinander verknüpft. Er verwaltet sogenannte Räume. Ein Raum besteht aus 0 bis 2 Clients und dient dazu diese miteinander zu verbinden. Auf dem Server kann es mehrere Räume gleichzeitig geben – dadurch könnte dieser Signaling-Server theoretisch auch noch für weitere WebRTC-Anwendungen verwendet werden. Ein Raum hat eine ID, diese ist eine 256-Bit-Entropie. Clients werden durch eine 128-Bit-Entropie identifiziert.

So können sich zwei Clients durch ein Out-Of-Band kommuniziertes Secret (die Raum-ID) über diesen in Kontakt treten. Diese könnte beispielsweise, würde es sich bei der entwickelten Kamera um ein echtes Produkt handeln, bei der Herstellung generiert werden, ausgedruckt und neben die Kamera in die Verpackung gelegt werden.

3) Interface: Das Interface verbindet sich mit dem Signaling Server und nimmt eine Raum-ID entgegen. Mit dieser Raum-ID wird dann auf dem Signaling Server entweder ein neuer Raum erstellt, falls noch keiner mit der ID existiert, oder es wird dem bestehenden Raum beigetreten. Nach dem ein weiterer User beigetreten ist, fängt der in Kapitel XY erklärte Aufbau eines WebRTC-Streams zu dem Nutzer an.

D. Signaling Server

Als erstes wurde der Signaling Server entwickelt, da dieser keine Abhängigkeiten an seine Clients hat. Die Kamera-Software und Interface setzen beide jeweils den laufenden Signaling Server vorraus um korrekt zu funktioneren.

Asynchronität: Eine logische Anforderungen an den Server ist das gleichzeitige verarbeiten mehrerer Verbindungen – ansonsten könnte immer nur ein Client alleine mit dem Server Verbunden sein. Dies würde die Signaling-Funktionalität eines Raumes unbrauchbar machen.

Die asynchrone Programmierung ist ein Konzept zur Lösung dieser Problemklasse. Da langlebige Verbindungen in der Regel einen großteil der Zeit ungenutzt sind, ist es naheliegend Verbindungen ohne neue Ereignisse keine CPU-Zeit zu geben um andere Verbindungen in der Zeit abzuarbeiten, in der auf Ereignisse gewartet wird.

HIER ZIETIEREN WAS DAS ZEUG HÄLT

1) Verbindungsaufbau: Um eine Secure-WebSocket-Verbindung aufzubauen, muss zu erst eine TCP- und dann darüber eine TLS-Verbindung zu dem Client aufgebaut werden. Über diese wird dann zu erst in HTTP kommuniziert (siehe WebSocket verbindungsaufbau Quelle suchen) und nach einer erfolgreichen Nachricht des Servers mit dem Status-Code 101 (Switching Protocols) wird die über den gleichen Transport-Weg (TLS) nach dem WebSocket-Protokoll kommuniziert.

Grafik einfügen.

VI. Benchmarks

VII. Auswertung

VIII. Nächste Schritte

IPV6 only => NAT holepunching entfällt

IX. Abbildungsverzeichnis

abc def

Literatur

[1] N. M. Edan, A. Al-Sherbaz, and S. Turner, “Design and evaluation of browser-to-browser video conferencing in webrtc,” in *2017 Global Information Infrastructure and Networking Symposium (GIIS)*, 2017, pp. 75–78.