HAW HAMBURG

BACHELORTHESIS
Dennis Dmitriev

Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten durch ein browserbasiertes Peer-to-Peer-Netzwerk

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Dennis Dmitriev

Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten durch ein browserbasiertes Peer-to-Peer-Netzwerk

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Lars Hamann
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 16. Juni 2022

Dennis Dmitriev

Thema der Arbeit

Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten durch ein browserbasiertes Peer-to-Peer-Netzwerk

Stichworte

Peer-to-Peer, Browser, WebRTC, CDN, BitTorrent, statische Inhalte

Kurzzusammenfassung

In dieser Thesis wird ein Konzept zur Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten durch ein browserbasiertes Peer-to-Peer-Netzwerk entworfen. Das System wird in Form einer Beispielanwendung implementiert. Abschließend wird das System anhand Messungen geprüft.

Dennis Dmitriev

Title of Thesis

Reduction of Bandwidth when Delivering Static Content using a Browser-based Peer-to-Peer Network

Keywords

Peer-to-Peer, Browser, WebRTC, CDN, BitTorrent, static content

Abstract

In this thesis, a concept for reducing bandwidth when serving static content using a browser-based peer-to-peer network is designed. The system is implemented in the form of a sample application. Finally, the system is tested through measurements.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In den letzten Jahren haben die Fortschritte in der Speicherung, der Kompression und der Kommunikation die Ausbreitung von Multimedia Daten im Web gefördert. Immer mehr statische Inhalte, wie Videos, Bilder und Songs, werden von Usern konsumiert. Laut Ciscos Visual Networking Index [2] sollen im Jahr 2020 ungefähr 82% des kompletten Internetverkehrs der Welt aus dem Teilen und Anschauen von Videos bestehen. Die Datengröße von statischen Inhalten erhöht sich durch neuen Technologien. Bilder und Videos mit einer Auflösung von 4k oder höher sind weitverbreitet. Als Resultat müssen Content Provider ständig ihre Bandbreite erweitern und deren Systeme verbessern, um einen Standard, den die User erwarten, zu halten.

Wenn Unternehmen statische Inhalte, wie Bilder und Videos, anbieten wollen, müssen diese die ganzen Kosten für die Bandbreite und Infrastruktur übernehmen, um es den Nutzern bereitzustellen. Kann sich das Unternehmen dies nicht leisten oder kommt es zu einem plötzlichen Anstieg an Benutzern, ist oft mit einem Ausfall des Dienstes zu rechnen. Mit dem Ausbau von Glasfaser- und 5G-Netzen könnten Peer-to-Peer-Netzwerke hier Abhilfe schaffen, indem ein Teil der Bandbreite zum Bereitstellen von Medien von den Nutzern getragen wird. Während sich die Nutzer statische Inhalte anschauen, werden diese auch anderen Nutzern bereitgestellt. Man hat eine dynamische Skalierung: Je mehr Nutzer sich z.B. ein Bild angucken, desto besser ist die Verfügbarkeit. Inhalte können von mehreren Quellen bezogen werden. Unternehmen, aber auch Privatpersonen, könnten so Ressourcen und damit Geld sparen.

Schon 2013 hat das Unternehmen *Yahoo* Interesse an einem browserbasierten Peer-to-Peer-Netzwerk zur Reduzierung der Bandbreite der eigenen Server gezeigt, indem *Yahoo* das Startup-Unternehmen *PeerCDN* erworben hat. Das Startup benutzte die experimentelle WebRTC API um einen Teil der Bandbreite auf die Besucher der Webseite

zu verlegen. Seit dem Erwerb des Startups wurde die Technologie jedoch nie bei Yahoo eingesetzt [5].

1.2 Ziel der Arbeit

In dieser Arbeit soll ein Konzept zur Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten mithilfe von browserbasierten Peer-to-Peer-Netzwerken erarbeitet und implementiert werden. Eine unbestimmte Anzahl an Usern soll während oder nach Beziehen eines statischen Inhalts auf einer Webseite, Teile dieses Inhalts an andere User übertragen können. So soll ein Teil der Last vom Webserver beim Bereitstellen dieses Inhalts von den Usern getragen werden. Dazu wird eine allgemeine Systemarchitektur ermittelt und die Hauptkomponenten herausgearbeitet.

Um die Umsetzbarkeit zu demonstrieren, soll die allgemeine Systemarchitektur in Form einer Beispielanwendung implementiert werden. Die grundlegenden Funktionen zum Anschauen von statischen Inhalten, wie Bilder und Videos, werden umgesetzt. Die Hauptkomponenten der Systemarchitektur sollen wiedererkennbar sein.

In Messungen sollen die Effektivität und die Performance des implementierten Systems ermittelt werden. Dies hilft die Vor- und Nachteile eines solchen Systems zu bestimmen.

Es wird in dieser Arbeit nicht auf das Teilen von privaten statischen Inhalten und deren Sicherheit in Peer-to-Peer-Netzwerken eingegangen, da dies ein Sonderfall ist. Auch würde alles bezüglich DRM und Copyright die Komplexität dieser Arbeit überschreiten.

1.3 Zielgruppe

Die Arbeit richtet sich an Content Provider, die mit immer höher steigenden Kosten und Probleme in der wachsenden Infrastruktur zur Bereitstellung von statischen Inhalten auf ihrer Plattform zu kämpfen haben. Dazu gehören nicht nur große Betreiber, wie *Facebook* und *YouTube*, auch Privatpersonen, können aus den Erkenntnissen dieser Arbeit profitieren.

Ebenso Personen im Bereich der Forschung von verteilten Systemen, besonders im Bezug auf Browsern, werden angesprochen. Für das Verständnis der Arbeit werden grundlegende Kenntnisse aus dem Bereich der verteilten Systeme vorausgesetzt.

1.4 Aufbau der Arbeit

Im zweiten Kapitel werden die grundlegenden Konzepte und Technologien zur Verständnis dieser Arbeit erläutert. Dazu wird ach einer kurzen Erläuterung von Content-Delivery-Networks auf verschiedene Architekturen zur Bereitstellung von Daten über das Internet eingegangen. Da ein Browser ein Softwaresystem mit vielen Restriktionen ist, werden die verfügbaren Technologien und Schnittstellen zur direkten Kommunikation zwischen Browsern vorgestellt. Im Anschluss wird BitTorrent, ein verbreitetes Peer-to-Peer Protokoll zur Distribution von Dateien, vorgestellt.

Das dritten Kapitel befasst sich mit der Konzipierung eines Systems zur Lösung des Problems. Es werden Ziele an das System definiert und anhand dieser eine allgemeine Systemarchitektur eingeführt.

Im vierten Kapitel wird die allgemeine Systemarchitektur implementiert. Es wird auf die einzelnen Komponenten des Systems eingegangen und deren Implementierung vorgestellt. Ein konkreter Verlauf der Anwendung wird präsentiert. Des Weiteren wird kurz ein System zum Messen der Beispielanwendung aufgezeigt.

Im darauffolgenden fünften Kapitel werden Messungen zur Effektivität und Performance des Systems durchgeführt. Es werden die Ergebnisse bewertet und weitere Probleme der Beispielanwendung aufgezählt.

Im sechsten und letzen Kapitel wird das Fazit dieser Arbeit gezogen und ein Ausblick auf die Zukunft gegeben.

2 Grundlagen

2.1 Content Delivery Network

Ein Content Delivery Network (CDN) ist ein weltweit verteiltes Netzwerk aus Caches, die Inhalte von einem Hauptserver replizieren. Hohe Serverbelastungen und Latenz-probleme werden bewältigt, indem Inhalte geographisch näher an die User gebracht werden. Wenn ein Client Inhalte einer Webseite anfragt, die ein CDN benutzt, werden die Anfragen an den nächsten Netzwerkknoten weitergeleitet. Besitzt dieser Knoten keine aktuelle lokale Kopie des Inhalts, holt sich der Knoten erst eine Kopie vom Hauptserver, spei-chert diese und überträgt diese anschließen an den Client. CDNs verwenden DNS um die Clients an die geeigneten Caches weiterzuleiten. Die Funktionsweise wird in Abbildung 2.1 näher erläutert.

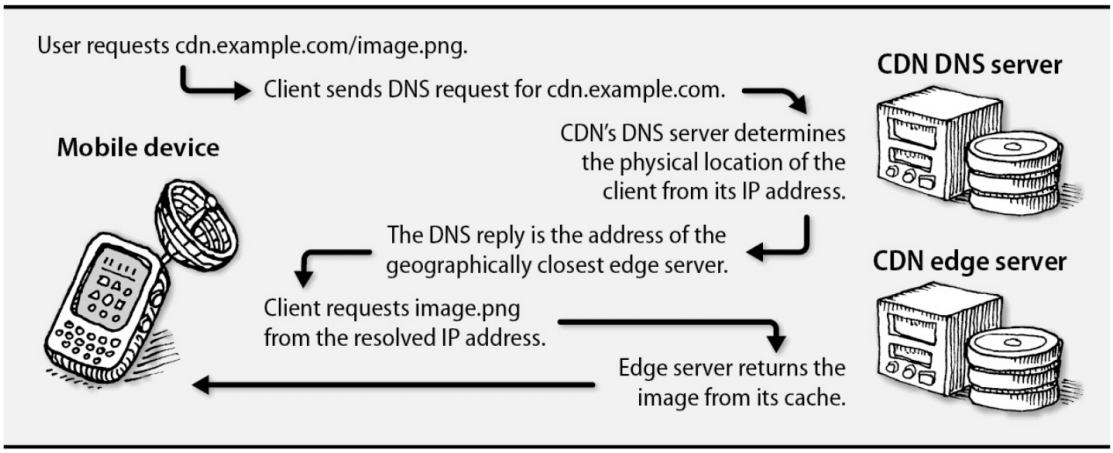

Abbildung 2.1: Rolle eines DNS in einem CDN [17]

Traditionell werden statische Inhalte, wie Bilder, Videos, HTML-Dateien, JavaScript-Bibliotheken, CSS-Dateien sowie weitere herunterladbare Objekte von Content Delivery Networks übertragen. Streaming Dienste wie *Netflix* und *Youtube* benutzen CDNs um

große Mediendateien global bereitzustellen. CDNs besitzen noch weitere Vorteile außer Performance-Verbesserungen. Die meisten CDNs bieten Sicherheitsdienste, wie Schutz vor DDoS-Attacken und Firewalls für Webapplikationen, an. Ein erheblicher Anteil von Inhalten im Web werden von CDNs bereitgestellt. Wenn man eine größere Webseite mit vielen Benutzern betreibt, ist das Investieren in ein CDN fast unausweichlich [17].

2.2 Architekturen

In diesem Abschnitt werden verschiedene Architekturen für das Bereitstellen von statischen Inhalten vorgestellt.

2.2.1 Client-Server-Architektur

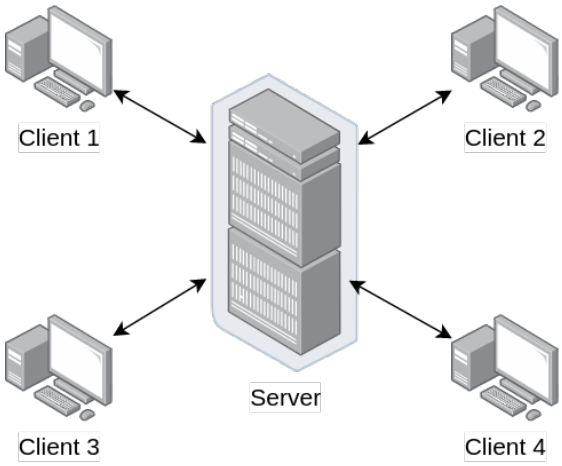

Abbildung 2.2: Aufbau einer Client-Server-Architektur

Populäre Webseiten, wie *Facebook* und *Youtube*, benutzen eine übliche Client-Server-Architektur, wie in Abbildung 2.2 abgebildet, um statische Inhalte den Usern bereitzustellen. Möchte ein User ein Bild oder ein Video angucken, so wird eine Anfrage vom Client, z.B. eine HTTP-Anfrage von einem Browser, an einen dedizierten Server geschickt. Als Antwort überträgt der Server dann die angefragte Datei. Auch kann der Server über einen Stream Teile der Datei verschicken, um das Warten auf das komplette Übertragen einer großen Datei zu vermeiden. Es existieren also dedizierte Server, die durchgängig verfügbar sein müssen, um auf die Anfragen von Clients einzugehen. Es werden keine direkten Verbindungen zwischen den Clients aufgebaut. Stattdessen kommunizieren die

Clients indirekt über einen Server. Obwohl ein Content Delivery Network die Auslastung der Band-breite sowie Anzahl an Anfragen an den Hauptserver reduzieren kann, so wird keine Last von den Clients getragen, die dazu in der Lage wären. Jedoch können die traditionellen Client-Server Systeme die Integrität der übertragenden Inhalte sicherstellen und so unberechtigte Manipulationen der angefragten Dateien vermeiden [12].

2.2.2 Peer-to-Peer-Architektur

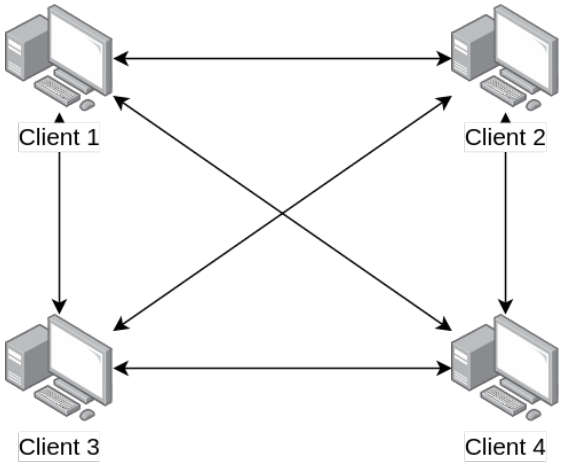

Abbildung 2.3: Aufbau einer Peer-to-Peer-Architektur

In einer Peer-to-Peer-Architektur, wie in Abbildung 2.3 zu sehen, werden kaum bis gar keine Verbindungen zu dedizierten Servern aufgebaut. Stattdessen kommuniziert die Anwendung direkt mit anderen Clients, die Peers genannt werden. Die Peers sind meistens nicht vom Provider, sondern Desktops, Laptops und Smartphones von Usern, die sich Zuhause, in Universitäten, Büros oder Unterwegs aufhalten. Durch diese direkte Kommunikation zwischen den Peers ohne Server, wird diese Architektur Peer-to-Peer, kurz P2P, genannt. Einer der populärsten Peer-to-Peer-Anwendungen ist die Distribution von Dateien mithilfe des BitTorrent-Protokolls. Einer der größten Vorteile einer P2P-Architektur ist die Skalierbarkeit. Wenn ein Peer eine Datei Peer-to-Peer bezieht, kriegt dieser Teile der Datei von mehreren Peers. Die Bandbreite wird in diesem Fall geteilt und man spart sich die Belastung eines einzelnen Servers. Der Peer stellt dann auch seine Bandbreite zu Verfügung, d.h. je mehr Peers es in dem Netzwerk gibt, desto mehr Bandbreite steht zur Verfügung. Die P2P-Architekturen sind dadurch oft kosteneffizienter, da teure Server-Infrastrukturen wegfallen und die Last dezentralisiert von

den Clients getragen wird. Jedoch hat die P2P-Architektur mit anderen Problemen zu kämpfen:

1. **Sicherheit:** Es werden direkte Verbindungen zu unbekannten Peers aufgebaut. Dies birgt potentielle Gefahren. Man muss sicherstellen, dass die unbekannten Peers keinen Schaden anrichten können.

2. **Performance:** Eine Datei wird nicht über einen dedizierten Server mit ausreichender Ausstattung angefragt. Ss müssen erstmal Verbindungen zu anderen Clients ausgehandelt werden, die dann auch die benötigte Bandbreite zur schnellen Übertragung besitzen müssen. Folglich kann sich die Zeit zum Erhalt des ersten Bytes drastisch erhöhen.

3. **Zuverlässigkeit:** Ein CDN in einer Client-Server-Architektur kann die Verfügbarkeit von statischen Inhalten so gut wie garantieren. Möchte man eine Datei Peer-to-Peer herunterladen und kein Peer mit der Datei ist verfügbar, so kann man die Datei nicht beziehen. Auch kann ein Peer während des Austausches auf eine unbestimmte Zeit verschwinden [12].

2.2.3 Hybride Architektur

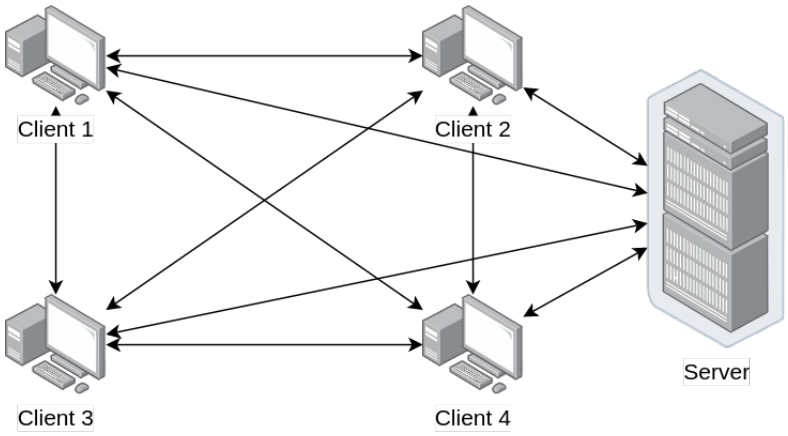

Abbildung 2.4: Aufbau einer hybriden Architektur

Die Abbildung 2.4 zeigt die hybride Architektur. Diese Architektur versucht die hohe Verfügbarkeit einer Client-Server-Architektur mit der dezentralisierten Lastverteilung eines Peer-to-Peer Systems zu vereinen. Wenn ein Peer nicht in der Lage ist ein gewünschtes Teil des statischen Inhalts von einem Peer-to-Peer-Netzwerk rechtzeitig zu erhalten,

wird der Teil von einem dedizierten Server heruntergeladen. Zusätzlich wird die initiale Verbindungszeit sowie die Zeit zum Erhalt des ersten Bytes in einem Peer-to-Peer Systems verringert, da die ersten Teile des statischen Inhalts von einem CDN oder Webserver angefragt werden können. Dies garantiert die Zuverlässigkeit. Selbst wenn ein Peer bei der Übertragung einer Datei verschwindet, kann ein Server das Übertragen übernehmen [6].

2.3 Technologien im Browser

In diesem Kapitel werden die relevanten Technologien und Schnittstellen in Browsern vorgestellt, die benutzt werden um ein browserbasiertes P2P-Netz aufzubauen.

2.3.1 WebSocket

Die Websocket API und das Websocket Protokoll, definiert in RFC6455, ermöglichen einen asynchronen Vollduplex für Webanwendungen, d.h. Daten können im Kommunikationskanal gleichzeitig in beide Richtungen übertragen werden. Es basiert auf TCP und ermöglicht Echtzeit-Updates zwischen Client und Server [20].

Für den Verbindungsaufbau wird zwar am Anfang HTTP benutzt, jedoch endet die Verbindung nach Erhalt einer Antwort nicht. Die Websocket API befreit Webanwendungen von der Einschränkung des typischen HTTP Anfrage-Antwort-Kreislaufs. Solange die Verbindung bestehen bleibt, können Nachrichten asynchron ohne Nachfrage ausgetauscht werden.

Es wird immer ein Header mit einer Nachricht verschickt, jedoch ist dieser Header meist unter acht Byte groß und somit viel kleiner als bei HTTP. Dies bringt den großen Vorteil mit sich, dass die Belastung des Clients und - viel wichtiger - des Servers reduziert wird. Websockets sind besonders wertvoll für zwei Verwendungszwecke, die sonst von keinen anderen Technologien ausreichend gedeckt werden:

1. Kleine und häufige Kommunikation zwischen Client und Server mit kleinem Overhead pro Nachricht
2. Server-initiierte Kommunikation mit einem Client, in der der Client beim Server nicht ständig nachfragen muss [16].

Abbildung 2.5 zeigt ein Sequenzdiagramm einer Websocket-Session. Eine Websocket-Verbindung wird mit einer HTTP-Anfrage an den Server initiiert. Wenn der Server in der Lage ist eine Websocket-Verbindung bereitzustellen, bestätigt dieser dies mit einer HTTP-Antwort und es wird zu einer Websocket-basierten Verbindung gewechselt. Danach können die Kommunikationspartner nach Belieben Daten übertragen. Der Browser beendet die Websocket-Verbindung automatisch, wenn der User die Seite schließt oder wegnavigiert. Alle populären Browser unterstützen Websocket seit 2013 [20].

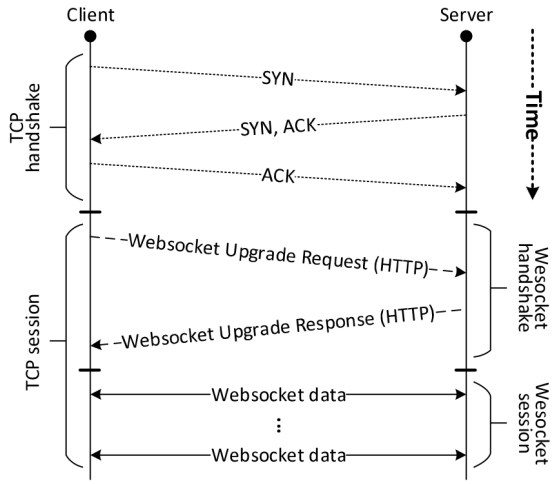

Abbildung 2.5: Websocket über TCP Sequenzdiagramm [20]

2.3.2 WebRTC

Web Real-Time Communication (WebRTC) ist ein neuer Standard, der eine Echtzeit-Kommunikation zwischen zwei oder mehr Browsern ermöglicht, es können also direkte Peer-to-Peer Verbindungen aufgebaut werden. Es ist zum Stand dieser Arbeit die einzige Möglichkeit eine direkte Verbindung zwischen Browsern herzustellen. Typische Peer-to-Peer Anwendungen wie Audio- und Videotelefonie sind ganz ohne Plugins von Drittanbietern möglich. Die Hauptkomponenten der WebRTC API sind:

1. **RTCPeerConnection** - WebRTC-Verbindung zwischen zwei Peers.
2. **MediaStream** - Stream von Media Content, die aus Video oder Audio Tracks bestehen, z.B. Webcam oder Mikrophon vom User.

3. **RTCDataChannel** - bidirektionaler Datenkanal zwischen zwei Peers einer WebRTC-Verbindung.

Jedoch ist WebRTC nicht nur eine einzelne API, sondern eine Sammlung von APIs und Protokollen. WebRTC wird von den meisten Browsern unterstützt, jedoch kann die Unterstützung sich zwischen den Browsern leicht unterscheiden [8]. Da nur RTCPeer-Connection und RTCDataChannel relevant für diese Arbeit sind, wird nicht näher auf die MediaStream-Komponente eingegangen.

RTCPeerConnection

Die RTCPeerConnection-Komponente repräsentiert eine Verbindung mit einem anderen Peer, der meist eine weitere Instanz der gleichen JavaScript-Anwendung ist. RTCPeer-Connection vereint den Verbindungsaufbau, das Management und die Informationen zum Zustand jeder Peer-To-Peer Verbindung in einem Interface.

Eine RTCPeerConnection kann auch ohne einen Server aufgebaut werden, jedoch braucht WebRTC im tatsächlichen Gebrauch oft die folgenden vier serverseitigen Funktionalitäten:

- Findung von User und deren Kommunikation
- Signaling zwischen den Usern
- NAT- und Firewall-Traversal
- Relay-Server, falls die Peer-to-Peer Kommunkation fehl schlägt

Um eine Verbindung aufzubauen, müssen Informationen zwischen den Peers ausgetauscht werden, dieser Mechanismus wird Signaling genannt. Zur Beschreibung der Signaling-Daten wird das Session Description Protocol(SDP, RFC 4566) benutzt. Diese Daten enthalten die öffentlichen IP-Adressen beider Peers sowie die lokalen IP-Adressen, falls sich die Peers im selben LAN befinden. Der Austausch dieser Daten passiert üblicherweise mithilfe eines sogenannten Signaling-Servers, der von beiden Peers erreichbar sein muss. Es verbinden sich beide Peers zum Signaling-Server und benutzen diesen als einen Relay um deren Signaling-Daten jeweils zum anderen Peer zu transferieren. Das Protokoll zum Austausch der Daten ist zwar nicht genau vorgeschrieben, jedoch wird meist Websocket benutzt.

WebRTC benutzt Interactive Connectivity Establishment(ICE, RFC 5245) als Methode zum Aufbauen und Aufrechterhalten einer Peer-to-Peer-Verbindung über UDP. ICE versucht am Anfang die Peers direkt über UDP zu verbinden. In diesem Prozess werden STUN-Server benutzt. Das Session Traversal Utilities for NAT(STUN, RFC 5389) Protokoll ermöglicht einer Applikation die Präsenz einer NAT im Netzwerk zu erkennen und in dem Fall die öffentliche IP-Adresse und Port für eine Verbindung zu bekommen. Der STUN-Server wird im öffentlichen Internet bereitgestellt [4].

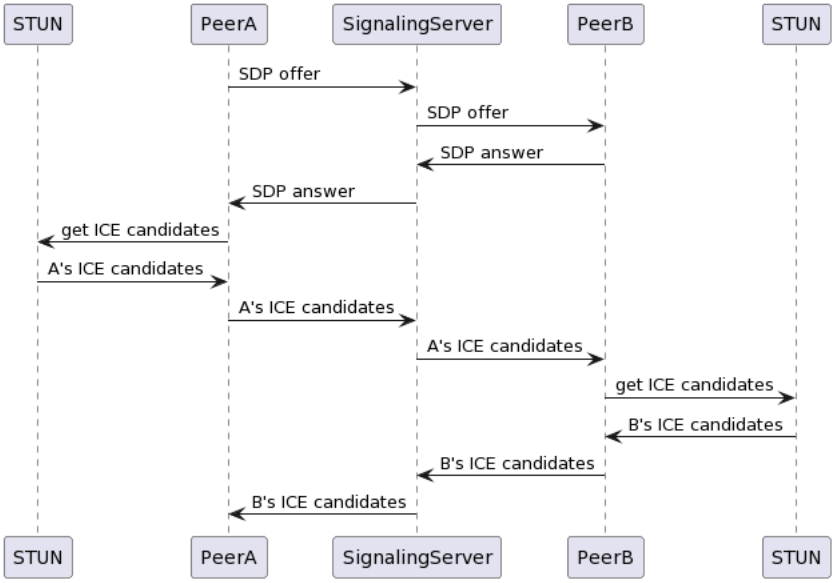

Abbildung 2.6: Sequenzdiagramm eines WebRTC-Handshakes vgl. [15]

In Abbildung 2.6 ist ein Sequenzdiagramm eines typischen WebRTC-Handshakes zum Aufbau einer RTCPeerConnection zu sehen. PeerA schickt über den Signaling-Server ein SDP-Angebot an PeerB. PeerB antwortet über den Signaling-Server zurück. PeerA holt sich ICE-Kandidaten von einem STUN-Server. Die ICE-Kandidaten sind die möglichen Adressen und Ports zum Aufbauen einer Verbindung. Diese werden dann über den Signaling-Server an PeerB verschickt. PeerB holt sich eigene ICE-Kandidaten von einem STUN-Server. Die ICE-Kandidaten von PeerB werden danach über den Signaling-Server an PeerA geschickt. Beide Peers besitzen nun die ICE-Kandidaten und somit Tupel von IP-Adressen und Ports des jeweils anderen und können Informationen direkt austauschen.

Wenn ein direkter Verbindungsaufbau über UDP fehl schlägt, wird ein TURN-Server benutzt. Das Traversal Using Relays around NAT (TURN, RFC 5766) Protokoll ermöglicht

einer Applikation hinter einem NAT eine IP-Adresse und Port von einem Relay-Server vom öffentlichen Internet zu bekommen. Trifft dies sein, gibt es keine direkte Peer-to-Peer Verbindung und alle Daten werden dann über einen Relay-Server ausgetauscht [4].

RTCDataChannel

Die RTCDataChannel API erlaubt das Übertragen von beliebigen Binärdaten zwischen Browsern und beschreibt den Datenkanal von WebRTC. Eine Webseite braucht keine Erlaubnis vom User um einen WebRTC-Datenkanal aufzubauen, die Webapplikation muss nur über einen sicheren HTTPS-Kanal übertragen werden. Die folgenden Anwendungen werden direkt im Browser mit dem Einsatz der API möglich gemacht:

- Mehrspieler-Spiele
- Echtzeit-Chats
- Datentransfer
- Dezentralisierte Netzwerke [4]

2.4 BitTorrent

BitTorrent ist ein populäres Peer-to-Peer-Protokoll für die Distribution von Dateien. Im Jahr 2019 war BitTorrent für 27% des Upload-Datenverkehrs des ganzen Internets verantwortlich [3]. Die Ansammlung an Peers, die an der Distribution einer bestimmten Datei teilnehmen, wird als Torrent bezeichnet. Die Peers in einem Torrent laden Teile einer Datei, Pieces genannt, voneinander herunter. Die Größe der Pieces ist variabel, beträgt jedoch meist 256KB. Am Anfang, wenn sich ein Peer einem Torrent anschließt, besitzt dieser keine Pieces. Über die Zeit erhält der Peer immer mehr Pieces von den anderen Peers im Torrent. Diese Pieces werden schon beim Herunterladen an andere Peers bereitgestellt. Hat ein Peer alle Pieces und somit die komplette Datei erhalten, kann dieser den Torrent verlassen oder weiter im Torrent bleiben und zur Distribution der Datei beitragen. Auch kann ein Peer einen Torrent jederzeit verlassen und später wieder Beitreten.

Für das Finden von anderen Peers verwendet das BitTorrent-Protokoll Trackers. Über eine .torrent-Datei erhalten Peers alle nötigen Metadaten, um sich einem Torrent anzuschließen und den Austausch zu beginnen. Dazu gehören IP-Adressen der Tracker und Informationen über die Pieces. Wenn ein Peer einem Torrent beitritt, registriert sich der Peer beim Tracker und informiert diesen periodisch über die Teilnahme. Auf diese Weise hat der Tracker einer Übersicht von allen Peers in einem Torrent. Ein Torrent kann Tausende von Peers zu jederzeit haben. Auch erhält der Peer eine Liste aus zufälligen IP-Adressen von Peers, die am Torrent teilnehmen, vom Tracker. Der Peer versucht parallele TCP-Verbindungen zu den Peers in der Liste aufzubauen. Der Datenaustausch kann dann über die erfolgreichen Verbindungen beginnen. Über die Zeit können Peers die Verbindung abbrechen und andere Peers können versuchen eine Verbindung aufzunehmen. Also verändern sich über die Zeit die verbundenen Peers.

Üblicherweise benutzt das BitTorrent-Protokoll eine Technik namens „rarest first“, um zu entscheiden, welche Pieces angefragt werden. Unter den verbundenen Peers wird überprüft, welche Pieces am seltensten vorkommen. Diese Pieces werden dann als erstes heruntergeladen. Das Ziel ist die Anzahl der Kopien jedes Pieces im Torrent gleich zu halten.

BitTorrent benutzt einen Algorithmus, der „tit-for-tat“ gennant wird, um zu entscheiden, welche der verbundenen Peers die Pieces bekommen. Ein Peer misst die Datenübertragungsrate von den verbundenen Peers und priorisiert vier Peers mit den höchsten Werten. Zu den vier priorisierten Peers wird noch ein zufälliger Peer ausgewählt. Alle paar Sekunden werden die Datenübertragungsraten neu berechnet und die vier Peers angepasst. Das Ziel ist die Peers mit den höchsten Datenübertragungsraten zu paaren und so den Datenaustausch zu optimieren. Die Auswahl eines zufälligen Peers erlaubt auch neuen Peers Pieces zum Austauschen zu bekommen. Die fünf Peers bezeichnet man als „unchocked“, alle anderen verbundenen Peers werden „chocked“ genannt, da diese keine Pieces vom Peer bekommen.

BitTorrent hat noch viele weitere Mechanismen zur effizienten Distribution von Dateien, jedoch sind die beiden vorgestellten Mechanismen die Hauptgründe, die zum Erfolg von BitTorrent geführt haben[13].

Das BitTorrent-Protokoll hat seit der Veröffentlichung mehrere Erweiterungen über die BitTorrent Enhancement Proposals(BEP) erhalten.

Dazu gehören Magnet-Links, die in BEP 9 vorgestellt wurden. Dabei handelt es sich um eine URI, die aus Informationen zu den Dateien und Trackern besteht. Die Erweiterung

erlaubt das Übertragen der restlichen Metadaten von anderen Peers. So kann man einem Torrent ohne einer .torrent-Datei beitreten. In den .torrent-Dateien ist der Infohash des Torrents beschrieben. Es ist ein SHA1-Hash, den die Peers zum Identifizieren des Torrents benutzen. Der Magnet-Link fokussiert sich auf den Infohash und lässt die Metadaten im Binärformat außen vor. Informationen wie Tracker können in einen Magnet-Link mit aufgenommen werden [9].

Eine weitere wichtige Erweiterung für das Protokoll ist die Einführung einer verteilten Hashtabelle, beschrieben in BEP 5. Dies ermöglicht das Finden und Paaren von Peers ohne einen Tracker [14]. Da Browser für das Finden und Paaren von anderen Browsern einen Signaling-Server brauchen, ist auch die Implementierung einer DHT im Browserkontext zum Stand dieser Arbeit nicht möglich.

BEP 17 und 19 erweitern das Protokoll um "Webseeds". Peers können Pieces von HTTP-Servern übertragen [10][1]. Webseeds ermöglichen eine hybride Architektur zur Bereitstellung von statischen Inhalten.

3 Konzept

3.1 Ziele des Systems

Das primäre Ziel des Systems ist die Reduzierung der Bandbreite mithilfe eines browserbasierten Peer-to-Peer-Netzwerks. Wenn sich ein User einen statischen Inhalt anschaut oder angeschaut hat, spendet der User etwas seiner Bandbreite und trägt so zur dezentralisierten Lastenverteilung bei. Man kann es sich wie ein automatisch skalierendes Peer-to-Peer CDN vorstellen, je mehr User teilnehmen, desto besser funktioniert das CDN. Dafür müssen direkte Verbindungen zwischen Browsern hergestellt werden.

Das System muss zusätzlich noch gewisse Erwartungen, die User beim Anschauen von statischen Inhalten auf populären Plattformen wie Facebook und YouTube haben, erfüllen. Man nehme die erfolgreiche Video-Plattform YouTube. Besucht ein User die Webseite, kann dieser einfach auf ein Video klicken und es anschauen. Dies passiert ohne die Installation von extra Addons oder Plugins. Die Prozesse geschehen im Hintergrund und tragen zu einer reibungslosen Erfahrung vom Nutzer bei. Dies muss auch das System erfüllen. Auch soll das Video schnell starten, jedoch wird etwas Ladezeit zum Buffern hingenommen. Laut einer Studie[11] werden einige User nach einer initialen Bufferzeit von zwei Sekunden ungeduldig und schließen das Video. In einem Video muss es möglich sein, zu beliebigen Zeitpunkten zu springen, auch hier wird eine kleine Ladezeit von zwei Sekunden zum Buffern hingenommen.

Auch wenn Internetverträge für Zuhause unbegrenztes Datenvolumen anbieten, haben viele Mobilfunkverträge immer noch eine limitierte Datenkapazität. Ein User soll beim Anschauen und Teilen eines statischen Inhalts nicht das ganze Datenvolumen seines Vertrags ausschöpfen. Es muss also möglich sein, das Teilen zu begrenzen, wie z.B. durch das Einstellen einer Grenze für die Upload-Rate.

3.2 System-Architektur

Das System muss zum Stand dieser Arbeit WebRTC verwenden, um direkte Peer-to-Peer Verbindungen zwischen den Browsern zu ermöglichen. Damit die Browser sich gegenseitig finden und verbinden können, wird ein Signaling-Server im System benötigt. Die Browser sprechen über Websocket mit dem Signaling-Server und kriegen so die nötigen Signaling-Daten. Zur effizienten Peer-to-Peer Datendistribution wird das verbreitete Peer-to-Peer Protokoll BitTorrent verwendet. Das BitTorrent-Protokoll eignet sich für das System, da die Rolle eines Signaling-Servers von einem Tracker übernommen werden kann. So kann ein effizientes Finden und Paaren von Peers gewährleistet werden. Da RTCDataChannels keine explizite Einwilligung des Users brauchen, kann ein reibungsloses Austauschen der Daten erfolgen.

Der „rarest first“ Mechanismus des BitTorrent-Protokolls hilft zwar die Distribution von Dateien zu optimieren, jedoch muss dies nicht immer sinnvoll sein. Möchte ein Peer ein Video über das Protokoll schauen, müssen die Pieces in sequentieller Reihenfolge angefragt werden. Möchte ein Peer zu einer bestimmten Stelle im Video springen, müssen auch die nötigen Pieces angefragt werden. Der Mechanismus ist empfohlen, jedoch optional und viele BitTorrent-Clients bieten die Möglichkeit an, die Pieces sequentiell herunterzuladen.

Ein reines Peer-to-Peer System hat mit Problemen wie Zuverlässigkeit und Zeit zum ersten Byte zu kämpfen. Um die Nachteile eines reinen Peer-to-Peer Systems zu reduzieren und so die Vorteile einer Client-Server-Architektur zu nutzen, wird auf eine hybride Architektur gesetzt. Das BitTorrent Protokoll schafft auch hier Abhilfe. Die Webseed Erweiterungen machen es möglich, Teile der Dateien von einem HTTP-Server zu beziehen. Um unter einer initialen Ladezeit von zwei Sekunden zu kommen, werden die ersten Bytes von einem Webserver heruntergeladen. Sobald ein browserbasiertes Peer-to-Peer-Netzwerk aufgestellt ist, wird ein Teil der angeforderten Datei von anderen Peers bezogen. Der Webserver muss somit nicht mehr die ganze Bandbreite zum bereitstellen der Datei auf sich nehmen.

Das BitTorrent-Protokoll bietet eine Metrik für die Upload-Rate an. Über diese Metrik ist es in vielen BitTorrent-Clients möglich das Hochladen von Daten an Peers zu begrenzen. Über den Browser ist es möglich ein mobiles Gerät zu erkennen, erkennt das System ein mobiles Gerät, so wird die Upload-Rate limitiert. Das mobile Gerät nimmt immer noch

am Peer-to-Peer-Netzwerk teil, jedoch kann so die Auswirkung auf das Datenvolumen reduziert werden.

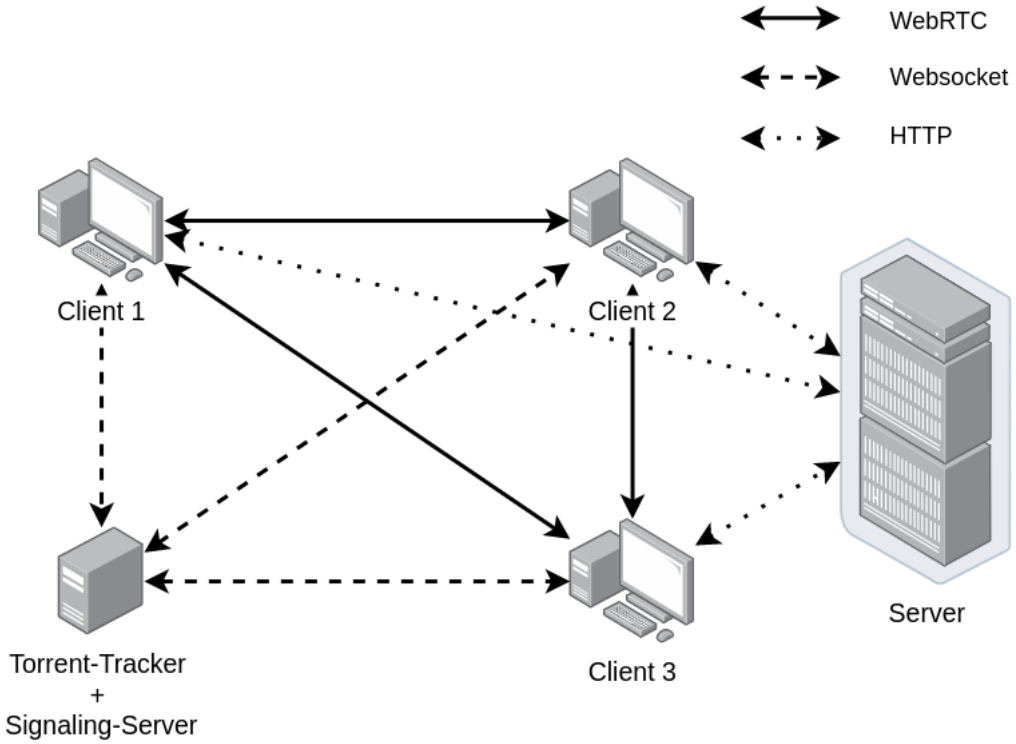

Abbildung 3.1: Konzept einer hybriden Architektur zur Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten

In der Abbildung 3.1 sind die drei Hauptkomponenten und deren Zusammenhänge im System zu sehen. Ein Webserver stellt die statischen Inhalte, deren .torrent-Dateien und die Webapplikation bereit. Besucht ein User den Webserver, wird die Webapplikation übertragen und eine BitTorrent-Client Instanz im Browser gestartet. Der Torrent-Tracker kümmert sich um das Finden von Peers um den Austausch von Dateien über das BitTorrent-Protokoll zu ermöglichen. Der Torrent-Tracker dient auch als Signaling-Server für die WebRTC Verbindungen zwischen den Browsern. In der Abbildung 5.1 wird zwar nur eine Torrent-Tracker Instanz gezeigt, es können jedoch mehrere Torrent-Tracker Instanzen verwendet werden, um einen Single Point of Failure zu vermeiden.

4 Implementierung

In diesem Kapitel wird die Architektur und Implementierung einer Beispielanwendung mit WebTorrent zur Reduzierung der Bandbreite vorgestellt.

4.1 Verwendete Tools

4.1.1 WebTorrent

Bei WebTorrent handelt es sich um einen BitTorrent-Client, der für moderne Browser geschrieben wurde. Es wird WebRTC statt TCP verwendet, um den direkten Datenaustausch zwischen den Browsern zu ermöglichen. Der vorgegebene Torrent-Tracker wird über Websocket angesprochen und übernimmt auch die Rolle des Signaling-Servers. Zum Stand dieser Arbeit ist das Paaren von Peers ohne einen Signaling-Server nicht möglich, deswegen ist auch die Benutzung von verteilten Hashtabellen(DHT) nicht umsetzbar. Es wird ein öffentlicher STUN-Server von Google für die ICE Kandidaten verwendet.

Der WebTorrent-Client kann automatisch zwischen sequentieller Reihenfolge und dem „rarest first“-Mechanismus wechseln. Auch sind die Webseeds BEP implementiert, um Pieces über HTTP herunterzuladen. Somit ist es möglich, die vorgestellte Systemarchitektur aufzubauen.

4.1.2 Puppeteer

Puppeteer ist eine Node.js Bibliothek zum programmatischen Starten, Navigieren und Schließen von Browser-Instanzen. Als Browser wird Chromium benutzt, eine Open-Source Variante von Googles Chrome Browser.

4.2 Komponenten des Systems

Das System wurde wie in Abbildung 4.1 implementiert. Die Hauptkomponenten aus der Architektur aus der Abbildung 3.1 sind wiederzuerkennen. WebTorrent wurde als eine schon existierende BitTorrent-Implementierung für Browser verwendet. Die Webseite in der Webanwendung kann ohne einen Webserver keine Informationen anzeigen. Der Client ist eine Instanz des WebTorrent-Clients. Die Webseite und der Client stehen eng in Verbindung und kommunizieren miteinander. Der WebTorrent-Client braucht einen Torrent-Tracker um Verbindungen zu anderen WebTorrent-Client Instanzen aufzubauen. Die verbundenen Clients tauschen die Pieces untereinander aus.

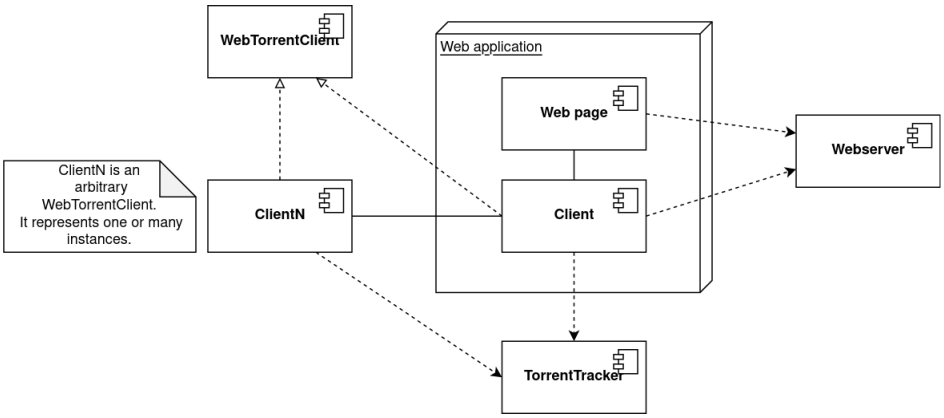

Abbildung 4.1: Komponentendiagramm des implementieren Systems

Im folgenden werden die Komponenten näher vorgestellt und auf technische Details eingegangen.

4.2.1 Webserver

Der Webserver ist in der Programmiersprache Go geschrieben und überträgt die Webapplikation. In dieser Beispielanwendung stellt der Webserver ein Bilderalbum und ein Video als statische Inhalte zur Verfügung. Damit Webseeds in WebTorrent auch ohne anderen Peers richtig funktionieren können, wurden für alle Inhalte jeweils eine .torrent-Datei mit den nötigen Metadaten erzeugt. Über eine HTTP-Anfragen kriegt man eine Liste der statischen Inhalte. Auch erhält man nähere Informationen, Magnet-Links und Aufenthaltsorte der passenden .torrent-Dateien. Der Webserver unterstützt HTTPS über ein Self-Signed Certificate.

4.2.2 Webapplikation mit WebTorrent

Bei der Webapplikation handelt es sich um eine Single-Page-Application, die mit dem Frontend-Framework Svelte geschrieben wurde. Das Bündeln der Webapplikation übernimmt WebPack. Der WebTorrent-Client wird direkt beim Öffnen der Webapplikation im Browser gestartet. Normalerweise speichert WebTorrent die Dateien im Arbeitsspeicher, dies wurde auf eine Speicherung in der IndexDB direkt im Browser geändert. Somit werden die Torrents beim Neuladen der Webapplikation nicht verworfen und können fortgesetzt werden. WebTorrent ermöglicht das Anfügen von statischen Inhalten an Container-Elementen auf der Seite. In der Beispielanwendung werden die statischen Inhalte nicht über die üblichen HTML-Tags wie <img> und <video> eingebettet, sondern dynamisch in -Containern auf der Seite hinzugefügt. Das hat den Vorteil, dass man sich nicht um die spezifischen HTML-Tags kümmern muss. WebTorrent übernimmt auch das Streamen von Audio und Video Inhalten direkt in die ausgewählten Container. Um ein mobiles Endgerät zu erkennen, wird eine Browser API benutzt. Wenn ein mobiles Endgerät erkannt wurde, wird der WebTorrent-Client mit einer definierten Upload-Rate begrenzt.

4.2.3 Torrent-Tracker

Der Torrent-Tracker ist von WebTorrent vorgegeben und in JavaScript geschrieben. Zusätzlich übernimmt der Tracker die Rolle eines Signaling-Servers für WebRTC. Die Kommunikation geschieht komplett über Websocket. Wenn sich ein WebTorrent-Client beim Tracker registriert, werden auch SDP Angebote mit verschickt. Möchte man sich zu einem anderen Client verbinden, wird das Angebot genommen und verarbeitet. Man kann eigene Instanzen starten oder schon vorgegebene Tracker benutzen. Der Vorteil von eigenen Instanzen ist die Performance.

4.2.4 Laufzeitsicht des Systems

Die Abbildung 4.2 beschreibt einen typischen Verlauf beim Anschauen von statischen Inhalten in der implementierten Beispielanwendung. Möchte sich ein User einen statischen Inhalt anschauen, kriegt die Webapplikation über eine HTTP-Anfrage an den Webserver einen Magnet-Link und den Aufenthaltsort der .torrent-Datei des statischen Inhaltes. Der WebTorrent-Client im Browser fügt den Magnet-Link hinzu und holt die nötigen Metadaten von der .torrent-Datei vom Server. Über den Magnet-Link ist es zwar möglich die Metadaten auch von anderen Peers anzufragen, wenn jedoch kein Peer verfügbar ist, können auch keine Metadaten bezogen werden. Um die Zeit zum ersten Byte zu reduzieren und die Verfügbarkeit zu gewährleisten, wird zu dem Torrent ein Webseed hinzugefügt. Die Bytes der ersten Pieces des Torrents werden so über HTTP vom Server angefragt. Das Herunterladen über den Webseed wird über die gesamte Dauer bis zum vollständigen Erhalt des statischen Inhalts fortgesetzt. Währenddessen registriert sich der WebTorrent-Client beim Torrent-Tracker für den Torrent. Befinden sich andere Peers im Torrent, so kriegt der WebTorrent-Client eine Liste und kann über den Signaling-Server WebRTC-Verbindungen zu anderen Peers aufbauen um bei der Distribution teilzunehmen. In der Abbildung verbindet sich der Client zu Client2 über einen WebRTC-Handshake. So können die Pieces des statischen Inhalts von anderen Peers heruntergeladen werden und der Webserver muss nicht mehr die ganze Datei übertragen. Dies führt zu einer Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten. In der Abbildung bezieht der WebTorrent-Client über den Server und Client2 die Pieces von der Datei. Die Bytes der Pieces werden dann nach und nach vom WebTorrent-Client an die Webseite angehängt. Mit dem vollständigen Beziehen der Datei, wird die Verbindung zum anderen Client geschlossen. Während der User die Seite offen hat und der WebTorrent-Client noch aktiv am Torrent teilnimmt, können andere WebTorrent-Clients über den Torrent-Tracker Verbindungen aufnehmen. Im Sequenzdiagramm stellt der Client3 eine WebRTC Verbindung her. Client3 fragt dann nach Pieces vom Client. Eine WebRTC-Verbindung bleibt so lange bestehen bis einer der Teilnehmer die Verbindung abbricht, dies kann mehrere Gründe haben:

- ein Teilnehmer hat alle Pieces erhalten
- ein schnellerer Peer steht zur Verfügung
- die Seite wurde geschlossen

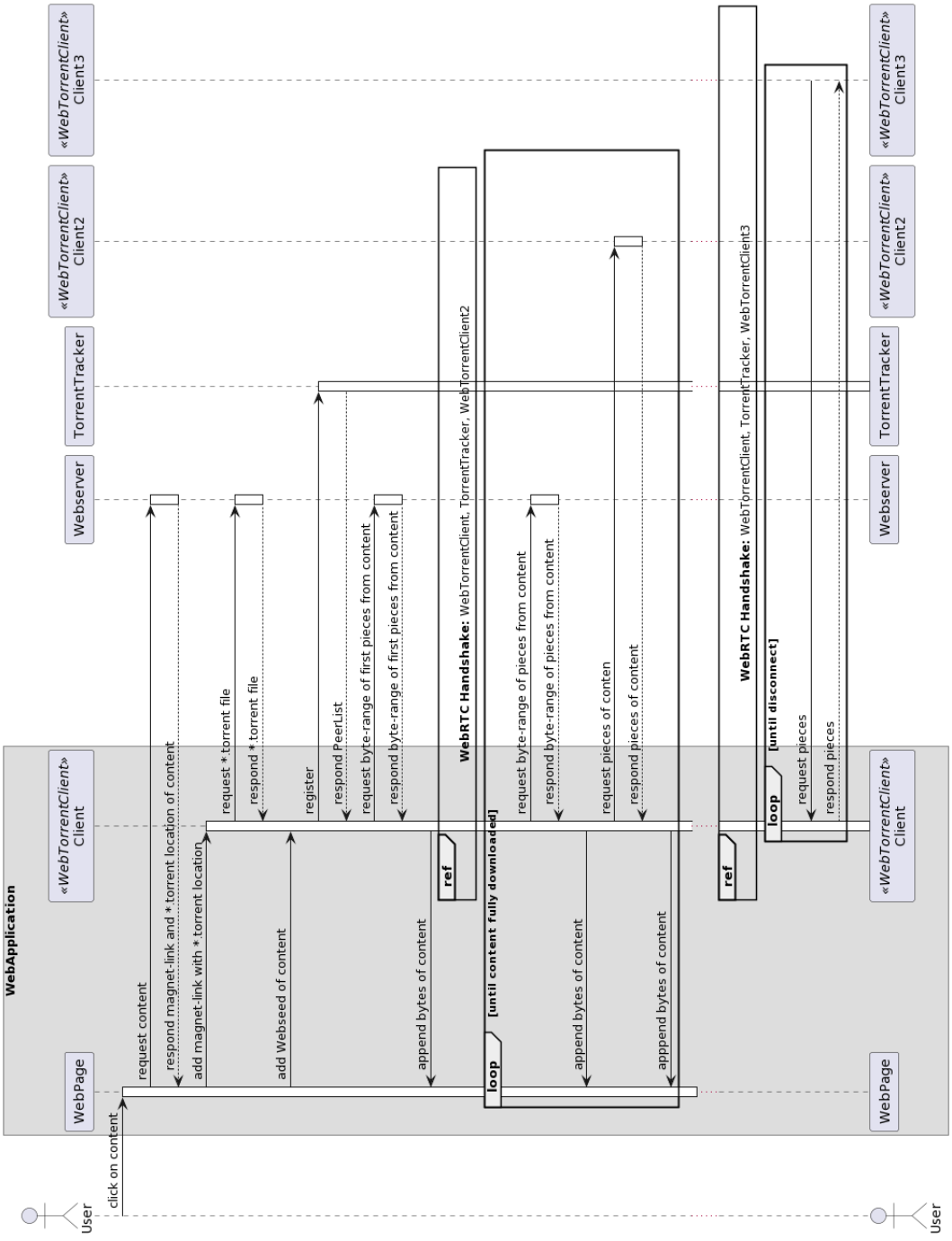

Abbildung 4.2: Sequenzdiagramm zum Anschauen und Teilen von statischen Inhalten im implementierten System

4.3 Messungen

Um zu testen wie gut das vorgestellte System funktioniert, werden einige Messungen durchgeführt. Mehrere Browser-Instanzen müssen orchestriert werden, um ein browserbasiertes Peer-to-Peer-Netzwerk aufzubauen. Ein geeignetes Werkzeug zur programmatischen Erstellung und Steuerung von Browsern ist die Node.js Bibliothek "Puppeteer". Die Tests sind in Node.js geschrieben und starten mehrere Browser-Instanzen über Puppeteer, die auf eine Testseite in der Webapplikation navigieren.

Auf der Testseite befinden sich Buttons zum Starten und Stoppen des Tests. Es gibt mehrere Gründe einen Start-Button zu benutzen und den Test nicht sofort beim Aufrufen der Seite zu Starten. Um die Peers auseinanderzuhalten, müssen diese auf irgendeine Weise gekennzeichnet werden. Über ein Textfeld auf der Testseite kann man den Peers eine clientId geben, diese kann beliebig aussehen. Über Puppeteer kann man das Textfeld ausfüllen, bevor der Test gestartet wird. Des Weiteren gibt es eine Checkbox auf der Testseite. Setzt man einen Hacken in der Checkbox, so wird der Peer im Test als mobiles Gerät angesehen und man kann unterschiedliche Internetgeschwindigkeiten haben. Außerdem, wenn der Test beim Aufruf der Seite gestartet werden würde, könnten Verzögerungen beim Starten der Browser-Instanzen zu Störungen beim Messen führen. Deswegen werden beim Ausführen der Tests, zunächst alle Browser-Instanzen gestartet und die Testseite aufgerufen, um sicher zugehen, dass alle Instanzen geladen wurden und bereit sind.

Wird der Start-Button gedrückt, wird der Ablauf im vorgestellten System simuliert, d.h. über einen Magnet-Link und einer Torrent-Datei wird im WebTorrent-Client ein Torrent gestartet. Ein Webseed wird hinzugefügt und die Verbindungen zu den Torrent-Trackern aufgebaut.

Während des Tests werden jede Sekunde Daten aus dem WebTorrent-Client Daten abgerufen und mit zusätzlichen Informationen zwischengespeichert. Dabei handelt sich um folgende Statistiken:

- Eingetragene clientId
- Heruntergeladene Bytes vom Torrent
- Hochgeladene Bytes vom Torrent
- Upload-Rate in Bytes/Sekunde

- Download-Rate in Bytes/Sekunde
- Ratio
- Progress
- Summe aller erhaltenen Daten
- Zeitstempel

Die Statistik Ratio spielt die größte Rolle beim Auswerten der Messungen. Die Ratio gibt das Verhältnis zwischen den hoch- und heruntergeladenen Daten an. Es werden die hochgeladenen Bytes durch die heruntergeladenen Bytes dividiert. Hat man eine Ratio von 1.0 in einem Torrent, so hat man die gleiche Anzahl an Bytes hoch- sowie heruntergeladen. Eine Ratio kann also schwanken, jedoch ist keine negative Ratio möglich. Eine positive Ratio bedeutet, dass der Peer einen Teil seiner Bandbreite dem Netzwerk zur Verfügung gestellt hat. Wenn ein Peer eine Ratio unter 1.0 hat, wurde dem Netzwerk weniger zurückgegeben als man genommen hat. Somit bedeutet eine Ratio über 1.0, dass man dem Netzwerk mehr Bandbreite zur Verfügung gestellt hat, als man selber verbraucht hat. In der Implementation ist eine Ratio von unter 1.0 bei vielen Peers zu erwarten, da Webseeds große Teile der statischen Inhalte an die Peers übertragen und so die Peers weniger Chance haben, ihre Bandbreite bereitstellen zu können als bei reinen Peer-to-Peer-Netzwerken. Trotzdem bedeutet eine positive Ratio bei Peers, dass diese zur dezentralisierten Lastenverteilung beigetragen haben und so die benötigte Bandbreite des Servers beim Bereitstellen von statischen Inhalten reduzieren konnten. Bezieht ein Peer einen statischen Inhalt und bekommt 20% der Dateien über einen anderen Peer, so hat der andere Peer eine Ratio von 0.2 für die Übertragen addiert bekommen. Der Peer hat somit auch 20% der Bandbreite zum Übertragen der Datei getragen.

Über den Stop-Button kann der Test beendet werden. Der WebTorrent-Client wird im Browser gestoppt und die gesammelten Statistiken werden auf der Seite ausgegeben. Auch wird angezeigt, wie viel Zeit für das vollständige Herunterladen der Datei benötigt wurde. Über Puppeteer werden die Ausgaben aller Browser-Instanzen eingesammelt und zu einer CSV-Datei zusammengefügt. Dies soll das weitere Auswerten der Daten erleichtern.

Über Python-Skripte werden die CSV-Dateien eingelesen und die Bytes in Megabytes zur besseren Darstellung umgewandelt. Danach werden aus den Daten verschiedene Diagramme erstellt und im PDF-Format gespeichert.

5 Evaluation

In diesem Kapitel wird sich das implementierte System genauer angeschaut. Das System wird aufgesetzt. Es werden Messungen zur Effektivität des browserbasierten Peer-to-Peer-Netzwerks durchgeführt und ausgewertet.

5.1 Aufbau

Zur Durchführung der Messungen wird die Beispielanwendung mit der Testseite auf einer Linux VM-Instanz auf der Google Cloud Plattform zum Laufen gebracht. Die Linux VM befindet sich in den Niederlanden in Europa und hat die folgende Systemkonfiguration:

- 2GB Arbeitsspeicher
- 2 virtuelle Kerne
- 10GB Speicher
- Ubuntu 22.04 LTS

Die Puppeteer-Tests werden auf einem Desktop-PC in Hamburg, Deutschland mit der Chromium Version 100 auf folgender Systemkonfiguration durchgeführt:

- Intel Core i5-6600 CPU mit vier CPU-Kernen
- 16GB DDR4 Arbeitsspeicher
- SSD-Speicher
- Windows 10 Pro(21h2)

	Mbit/s	MB/s	r. MB/s
Download(fest)	63.46	7.92	8
Upload(fest)	27.06	3.38	3
Download(mobil)	30.75	3.84	4
Upload(mobil)	8.78	1.10	1

Tabelle 5.1: Medianwerte globaler Bandbreiten von Usern

	fest	mobil
Download	4	2
Upload	1.5	0.5

Tabelle 5.2: Ausgewählte Internetgeschwindigkeiten für Peers in den Messungen in MB/s

Auf dem Desktop-PC steht eine Internetverbindung von je 250Mbit/s im Download sowie Upload zur Verfügung. Um Messungen mit zehn Peers durchzuführen, muss die Bandbreite gerecht aufgeteilt werden. Dazu wurden Geschwindigkeitsgrenzen für alle Peers gesetzt.

In der Tabelle 5.1 sind die Medianwerte der weltweiten Internetbandbreiten abgebildet. Die Daten stammen von einem der größten Dienste zum Messen von Internetbandbreiten [18]. Die Tabelle beinhaltet die Download- sowie Upload-Raten in Festnetzen als auch Mobilfunknetzen. WebTorrent verwendet Bytes für die Geschwindigkeitslimitierungen sowie der Metriken, deswegen wurden die Mbit/s-Werte auf MB/s umgerechnet. Auch wurden gerundete MB/s-Werte angegeben, um die folgenden Messungen zu vereinfachen und Messwerte besser darzustellen.

Eine Internetgeschwindigkeit von 250Mbit/s beträgt 30MB/s. Laut Statistiken [19] wurden im Jahr 2021 rund 58% aller Seitenaufrufe über mobile Endgeräte getätigt. Um die Bandbreite auf die Peers gerecht aufzuteilen, wurden die gerundeten Werte in Tabelle 5.1 halbiert. Für die Messungen wurden somit die in Tabelle 5.2 angegebenen Werte für die Geschwindigkeitslimitierungen eingesetzt.

Jeder zweite Peer, angefangen mit dem Peer mit einer clientId von 0, bekommt die Internetgeschwindigkeiten eines Mobilfunknetzes. Zur besseren Lesbarkeit werden im folgenden die Peers mit einer clientId von N, PeerN bezeichnet. In den Messungen laden und teilen die Peers eine Testdatei, bei der es sich um eine 129,2MB große Videodatei handelt.

Eine Torrent-Tracker Instanz wird auch auf der Linux VM gestartet. Zusätzlich befinden sich in den .torrent-Dateien noch weitere Tracker.

Die folgenden Messungen wurden min. dreimal durchgeführt um jegliche Störungen auszuschließen.

5.2 HTTP-Anfragen über Webseed

Um zu prüfen wie viele HTTP-Anfragen an den Webserver über den Webseed gestellt werden, bezieht ein Peer die vollständige Testdatei nur über den Webseed.

Zusammen mit der Anfrage nach der .torrent-Datei, wurden 988 HTTP-Anfragen für die Testdatei an den Webserver geschickt. Bezieht man die Datei wie gewohnt direkt über den HTTP-Link, werden nur drei HTTP-Anfragen an den Webserver gestellt und ein Stream vom Browser zum Webserver wird geöffnet.

Die Webseed Implementation in WebTorrent stellt somit viel mehr Anfragen an den Webserver, da die 987 Pieces der Datei einzeln über HTTP-Anfragen heruntergeladen werden. Dies wirkt sich nicht nur auf die Performance im Browser aus, da jedes Piece eine eigene Latenz hat, sondern auch auf den Server. Schon bei 1000 Usern, die auf die vollständige Testdatei zugreifen würden, müssten ca. eine Million HTTP-Anfragen vom Webserver bearbeitet werden.

5.3 Torrent Tracker Performance

Das Ziel dieses Tests ist das Einschätzen der Performance der Torrent-Tracker bezüglich der benötigten Zeit zum Paaren der Peers. Folgendes soll getestet werden: Ein WebTorrent-Client baut eine Websocket-Verbindung mit dem Torrent-Tracker auf. Dort erhält man eine Liste mit einem weiteren Peer. Da im Browser keine einfachen TCP- und UDP-Verbindungen möglich sind, wird eine WebRTC-Verbindung über den Torrent-Tracker aufgebaut. Erst nach einem erfolgreichen Handshake, können die Peers sich austauschen. Sobald das erste Byte von einem anderen Peer ankommt, wird die Messung gestoppt. Dieser Prozess geschieht bei jedem einzelnen Peer, deswegen ist ein schnelles Paaren entscheidend für den Erfolg des Systems.

In den Messungen hat der Start, das Verbinden und der Erhalt des ersten Bytes eines Peers ohne Webseed ungefähr drei bis fünf Sekunden gedauert. Die Zeit wäre bei einem kleinen Bild viel zu lang und es würde nie zu einem Austausch kommen, da der ganze statische Inhalt vom Webserver bezogen wird. Es sollten also nur Dateien über das System bereitgestellt werden, die eine große Dateigröße aufweisen. Dies ist zum Beispiel bei Videos und Audiodateien der Fall. Es müssen aber nicht unbedingt einzelne Dateien sein. In der Beispielanwendung wird auch ein Bilderalbum zur Verfügung gestellt. Es ist groß genug, dass es zu einem Austausch zwischen den Peers kommt.

5.4 Latenz bei geographischer Entfernung

Das Ziel dieses Tests war es zu messen, wie sich die geographische Entfernung auf die Latenz zwischen den Peers auswirkt. Dazu sollten zwei Tests durchgeführt werden. Im ersten Test befinden sich Peers auf einer VM in Europa und im zweiten Test befinden sich Peers in Nordamerika. Ein Peer in Deutschland Bezieht die Testdatei einmal von den Peers in Europa und einmal von den Peers in Nordamerika. Die Zeiten sollten dann verglichen werden.

Zum Stand dieser Arbeit ist es nicht möglich WebRTC-Verbindungen in der Google Cloud Plattform herzustellen. Auch wenn alle Ports und Protokolle erlaubt werden, konnten Browser auf der Linux VM von Google keine direkte Verbindung zu andern Browsern herstellen. Es wurde immer in TURN-Server als Relay benutzt.

Deswegen kann die folgende Behauptung in dieser Arbeit nicht getestet werden: Das BitTorrent-Protokoll achtet nicht darauf, wo sich ein Peer befindet. Somit wird auch nicht auf die Latenz Rücksicht genommen, es geht nur um die größeren Bandbreiten. Eine Lösung wäre es ein CDN statt einem Webserver zu benutzen. So wird die Latenz für den Webseed reduziert und das traditionelle CDN wird um ein Peer-to-Peer CDN erweitert.

5.5 Evaluation der Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten

Die folgenden Messungen sollen zeigen, wie viel das browserbasierte Peer-to-Peer Netzwerk zur Reduzierung der Bandbreite beim Bereitstellen der Testdatei beiträgt. Dazu wurde die Testdatei auf verschiedene Weisen von den Peers bezogen.

5.5.1 Sequentielles Beziehen von statischen Inhalten

Beschreibung

In dieser Messung werden zehn Browser-Instanzen erstellt. Die Instanzen laden dann die Testdatei nacheinander herunter, d.h. wenn ein Peer die Testdatei vollständig heruntergeladen hat und so in der Lage ist, die Datei komplett zu teilen, beginnt der nächste Peer mit dem Beziehen der Testdatei. Ziel ist es die Effektivität des browserbasierten Peer-to-Peer-Netzwerks in einem Idealfall zu ermitteln.

Ergebnisse

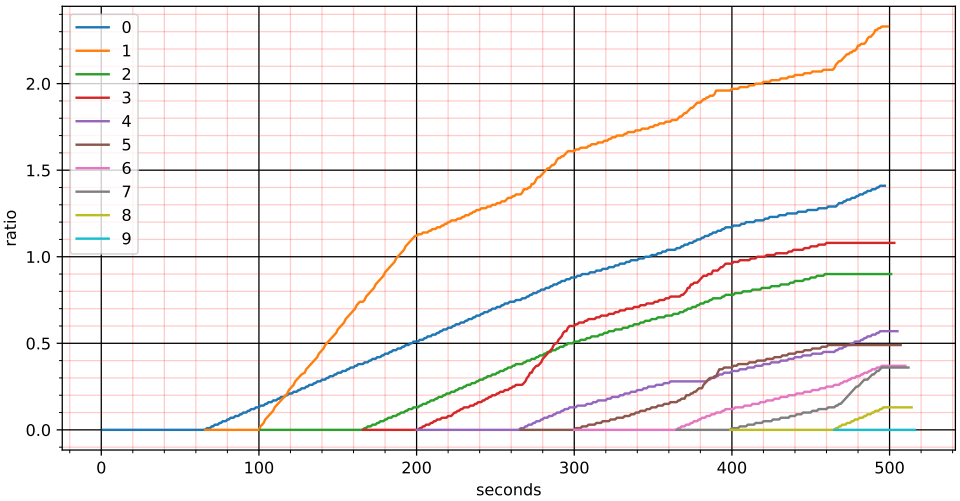

Abbildung 5.1: Verlauf der Ratios der Peers beim sequentiellen Beziehen der Testdatei

In der Abbildung 5.1 sind die Ratios der Peers über den Zeitraum der Messung zu sehen. Nach dem vollständigen Herunterladen der Testdatei, kann Peer0 dem Server beim Bereitstellen der Datei unterstützen. Dies kann man an dem Anstieg der Ratio von Peer0 erkennen. Peer1 bekommt die Pieces der Testdatei vom Server und Peer0. Da es sich bei Peer1 um einen Peer mit einer größeren Upload-Rate handelt, kann dieser die Pieces schneller teilen und überholt den mobilen Peer0 in der Ratio schon zum Download der Testdatei bei Peer2. Die Steigung der Ratio von Peer1 bleibt weiterhin stärker als bei Peer1 und Peer2, nachdem Peer3 angefangen hat, die Testdatei herunterzuladen. Peer1 und Peer2 weisen die gleiche Steigung der Ratio auf. Alle Peers benutzen die volle Bandbreite zum Hochladen der Testdatei an Peer3. Der Start des Downloads der Testdatei beim Peer4 verändert das Verhalten vom Peer1. Die Steigung der Ratio beim Peer1 ist nun nicht mehr so stark und entspricht der Steigung der anderen Peers. Peer4 benutzt die Mobilfunk-netzgeschwindigkeit zum Beziehen der Testdatei. Die Bandbreite wird unter den verfügbaren Peers gleichmäßig aufgeteilt. Der nächste Peer benutzt die Festnetzgeschwindigkeit zum Downloaden der Testdatei. Die Peers mit der Mobilfunkgeschwindigkeit benutzen die volle Upload-Rate zum Schicken der Pieces an Peer5. Peer1 und Peer3 stehen höhere Upload-Raten zu Verfügung, deswegen ist die Steigung der Ratio bei den beiden Peers auch stärker. Jedoch bleibt die Steigung nur bis zum Start des nächsten Peers bestehen. Die Bandbreite wird beim Peer6 wieder gleichmäßig aufgeteilt. Es gibt mehr Peers, die die Testdatei teilen und die Steigungen der Ratios sind sehr ähnlich. Dies bedeutet, das der Webserver weniger Bandbreite zur Verfügung stellen muss und die Peers immer mehr das Übertragen der Testdatei übernehmen.

Im weiteren Verlauf sind die Unterschiede in den Steigungen nicht mehr so drastisch wie im vorherigen Verlauf. Es stehen mehr Peers mit der Testdatei zur Verfügung, somit kann die Bandbreite zwischen den Peers gleichmäßig aufgeteilt werden. Der Web-server muss mit jedem weiteren Peer weniger Bandbreite aufbringen um die Testdatei zu Übertragen.

Je früher die Peers die vollständige Datei teilen konnten, desto mehr Ratio besitzen diese auch. Besonders Peer1 sticht wegen dem Anstieg der Ratio am Anfang der Messung heraus. Die Peers mit der höheren Upload-Rate konnten oft die vorherigen Peers mit der niedrigeren Ratio überholen.

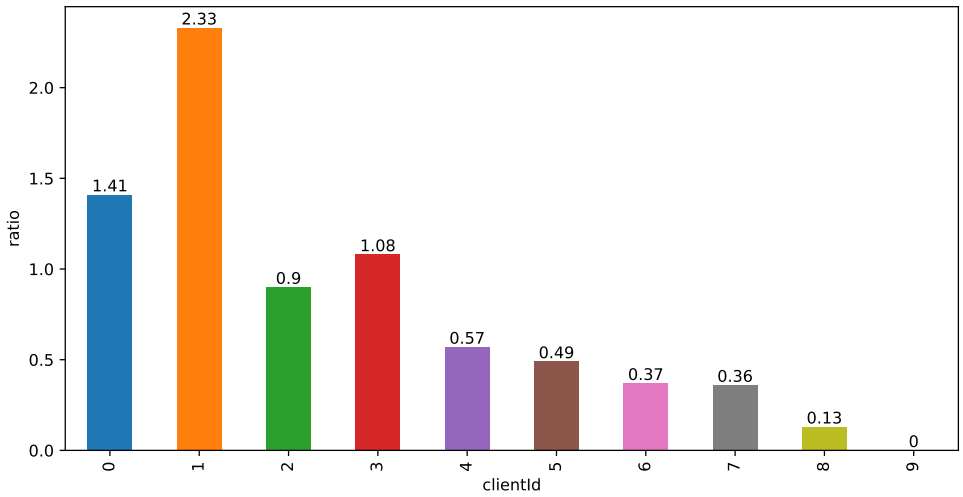

Abbildung 5.2: Finalen Ratios der Peers beim sequentiellen Beziehen

Die finalen Ratios der Peers sind in der Abbildung 5.2 zu sehen. Wie man den Werten entnehmen kann, beteiligen sich alle Peers bis auf den letzten bei der dezentralisierten Lastenverteilung. Der letzte Peer hat keine Ratio, da danach kein Peer mehr die Testdatei bezieht. Besonders auffällig ist Peer1 mit der größten Ratio. Peer1 hat mit einer Ratio von 2.33 auch 233% der Bytes der Testdatei an andere Peers verschickt. Zählt man alle Ratios zusammen erhält man den Anteil, den das browserbasierte Peer-to-Peer-Netzwerk beim Bereitstellen der Testdatei übernommen hat. Bei zehn Clients müsste der Server in einer klassischen Client-Server-Architektur die Datei auch zehn mal vollständig übertragen. Die Peers haben laut den finalen Ratios einen Anteil von 7.64/10 übernommen, d.h. 76,4% der Bandbreite wurde von den Peers getragen. Ein erheblicher Teil der Bandbreite beim Bereitstellen der Testdatei wurde durch das browserbasierte Peer-to-Peer-Netzwerk reduziert.

	Peer0	Peer1	Peer2	Peer3	Peer4	Peer5	Peer6	Peer7	Peer8	Peer9
Zeit	64s	33s	64s	32s	64s	32s	64s	32s	64s	32s

Tabelle 5.3: Zeiten der Peers zum fertigen Herunterladen der Testdatei beim sequentiellen Beziehen

In Tabelle 5.3 sieht man, dass die Peers mit der Festnetzgeschwindigkeit fast immer doppelt so schnell beim Herunterladen der Testdatei waren, wie die Peers mit der Mobil-

funknetzgeschwindigkeit. Die Verhältnisse entsprechen auch den in Tabelle 5.2 definierten Geschwindigkeiten.

5.5.2 Gleichzeitiges Beziehen

Beschreibung

In dieser Messung werden sechs Browser-Instanzen erstellt. In allen sechs Instanzen werden die Start-Buttons gleichzeitig gedrückt, d.h. alle Peers fangen an, die Testdatei gleichzeitig herunterzuladen. Dies soll die Veröffentlichung eines erwarteten statischen Inhalts simulieren, in der viele User gleichzeitig auf den Inhalt zugreifen.

Ergebnisse

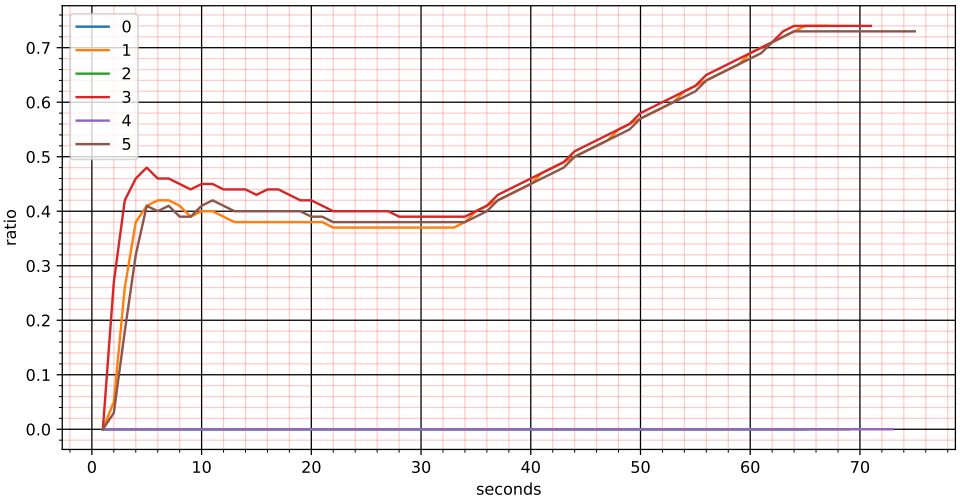

Abbildung 5.3: Verlauf der Ratios der Peers beim gleichzeitigen Beziehen

Wie man der Abbildung 5.3 entnehmen kann, bekommen die Peers mit einer höheren Download-Rate schneller die Pieces der Testdatei vom Server und können früher anfangen den Server beim Übertragen der Testdatei zu unterstützen. Die Ratio steigt in den ersten Sekunden bei Peer1, Peer3 und Peer5 stark an. Da die Download-Rate höher als die Upload-Rate ist, sinken die Ratios der Peers leicht bis diese die Testdatei vollständig heruntergeladen haben. In Tabelle 5.4 sind die Zeiten zum vollständigen Beziehen der

Testdatei dargestellt. In dem Verlauf sieht man, dass ab ca. Sekunde 34 in der Messung, die Ratios der Peers mit der Festnetzgeschwindigkeit stark ansteigen. Die Peers schicken Pieces der Testdatei an Peer0, Peer2 und Peer4. Diese brauchen aufgrund der niedrigeren Download-Rate, länger zum vollständigen Beziehen der Testdatei.

	Peer0	Peer1	Peer2	Peer3	Peer4	Peer5
Zeit	64s	33s	64s	34s	64s	35s

Tabelle 5.4: Zeiten der Peers zum fertigen Herunterladen der Testdatei beim gleichzeitigen Beziehen

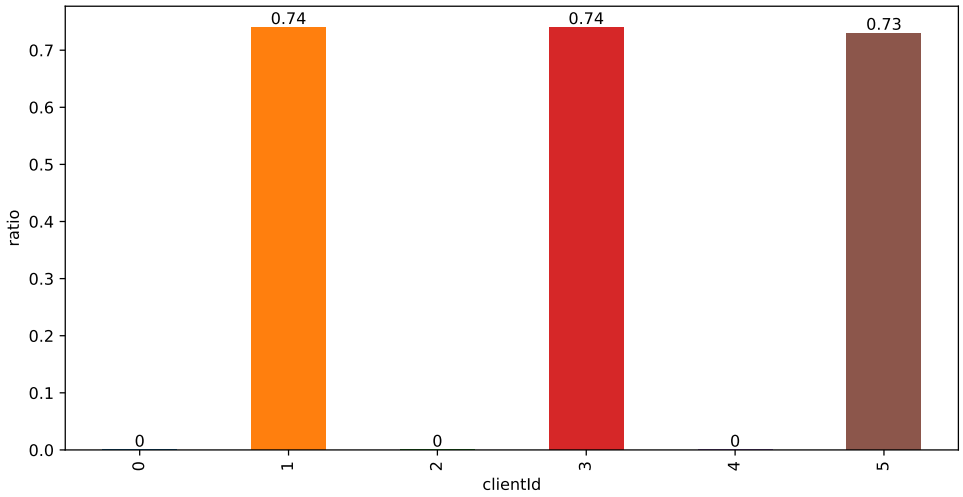

Abbildung 5.4: Finalen Ratios der Peers beim gleichzeitigen Beziehen

In Abbildung 5.4 sind die Ratios der Peers am Ende der Messung abgebildet. Die Peers mit der Mobilfunknetzgeschwindigkeit haben eine Ratio von 0, da diese keine Möglichkeit hatten, die Pieces an andere Peers zu verteilen. Es sind die Peers mit der niedrigsten Download-Rate und es gibt keine Peers nach denen mehr. Die Peers mit der höheren Download-Rate haben alle jeweils eine Ratio von ca. 0.74 erreicht. Zusammengerechnet ergibt sich eine Ratio von 2.21. Bei der Übertragung von sechs Testdateien, bedeutet dies, dass das browserbasierte Peer-to-Peer-Netzwerk einen Anteil von 2.21/6 übernommen hat. Die Festnetz-Peers haben somit 36.83% der Bandbreite zum Bereitstellen der Testdatei vom Webserver übernommen.

5.5.3 Beziehen in Gruppen

Beschreibung

In dieser Messung werden insgesamt 15 Browser-Instanzen erstellt. Es laden fünf Peers die Testdatei gleichzeitig herunter. Es gibt drei Peer mit der Mobilfunknetzgeschwindigkeit und zwei mit der spezifizierten Festnetzgeschwindigkeit. Sobald alle fünf Peers in dieser Gruppe die Testdatei vollständig heruntergeladen haben, startet die nächste Gruppe mit fünf Peers. In dieser Gruppe gibt es zwei Peers mit der Mobilfunknetzgeschwindigkeit und drei Peers mit einer Festnetzgeschwindigkeit. Nachdem auch jeder Peer in dieser Gruppe die Testdatei vollständig bezogen hat, werden die fünf Peers aus der ersten Gruppe gestoppt und die letzte Gruppe mit den letzen fünf Peers gestartet. Auch hier hat man, wie in der ersten Gruppe, drei Peers mit der Mobilfunknetzgeschwindigkeit und zwei Peers mit der Festnetzgeschwindigkeit. Es werden zwar insgesamt 15 Peers erstellt, jedoch sind zu jeder Zeit nur maximal zehn Peers aktiv, um die Performance aller Peers zu garantieren. Das Ziel dieses Test ist es, einen üblichen Fall beim Bereitstellen von statischen Inhalten zu simulieren: User greifen ungefähr gleichzeitig auf die Inhalte zu und verschwinden nach einiger Zeit und werden durch andere User ersetzt.

Ergebnisse

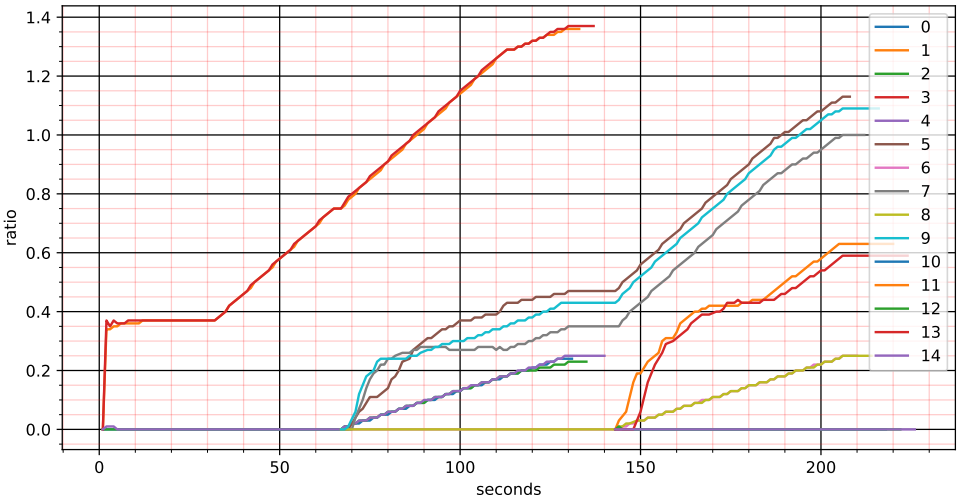

Abbildung 5.5: Verlauf der Ratios der Peers beim Beziehen in Gruppen

In Abbildung 5.5 ähnelt sich der Verlauf der ersten Gruppe stark dem Verlauf der vorherigen Messung in Abbildung 5.3. In der ersten Gruppen können Peer1 und Peer3, die eine Festnetzgeschwindigkeit haben, schon früher die ersten Pieces vom Webserver beziehen. Da diese Peers schon einige Pieces haben, können die Peers den Server entlasten und einen Teil der Bandbreite beim Übertragen der Testdatei an die Peers mit der Mobilfunknetzgeschwindigkeit übernehmen. Peer0, Peer2 und Peer4 haben keine Möglichkeit Pieces zu teilen, deswegen vermerken diese kaum bis gar keinen Anstieg in der Ratio. Sobald die Peers mit der schnelleren Download-Rate die Testdatei vollständig heruntergeladen haben, steigt die Ratio stark an.

In der Abbildung ist der Start der zweiten Gruppe gut erkennbar. Die mobilen Peers, die davor keine Möglichkeit hatten, zur dezentralisierten Lastenverteilung beizutragen, weisen einen Anstieg in ihrer Ratio auf. Diese benutzen die eigene Bandbreite zum Teilen der Testdatei. Die Ratios von Peer1 und Peer3 aus der ersten Gruppe steigen weiterhin stark an, die volle Upload-Rate wird benutzt. Peer5, Peer7 und Peer9 können die Pieces wegen der höheren Download-Rate schneller vom Server und den Peers aus der ersten Gruppe beziehen. Da diese auch eine höhere Upload-Rate als die mobilen Peers aus der ersten Gruppe haben, steigen Ratios stark an und überholen die mobilen Peers.

Mit dem Start der nächsten Gruppe, fallen die Peers aus der ersten Gruppe weg. Die Peers mit der Festnetzgeschwindigkeit tragen nun einen großen Teil beim Übertragen der Testdatei. Die Ratios steigen rasant an, die volle Upload-Rate wird benutzt. Auch die Peers mit der Mobilfunknetzgeschwindigkeit nutzen die komplette Bandbreite aus, ein Anstieg der Ratios bei Peer6 und Peer8 sind zu erkennen. Wie auch schon in den vorherigen Gruppen, beziehen die Peers mit der höheren Download-Rate die Pieces schneller von den anderen Peers und dem Webserver. Mit dem vollständigen Herunterladen der Testdatei, steigt die Ratio bei diesen Peers auch stark an.

Aus dem gesamten Verlauf ist zu erkennen, dass das Verschwinden von anderen Peers das System nicht beeinträchtigt. Die noch verfügbaren Peers können einspringen und die volle Bandbreite zum Hochladen benutzen.

In Tabelle 5.5 sind die Zeiten der Peers zum fertigen Herunterladen der Testdatei in der Messung zu erkennen. Die Peers mit der Festnetzgeschwindigkeit brauchen nach der ersten Gruppe ca. 10 Sekunden länger zum vollständigem Beziehen der Datei. Grund hierfür ist, dass der WebTorrent-Client eine Liste mit zufälligen Peers vom Tracker bekommt. Es können am Anfang Verbindungen zu langsamen Peers, wie die mit der Mobilfunkgeschwindigkeit, aufgebaut werden. Erst nach einigen Sekunden, nachdem die

	Peer0	Peer1	Peer2	Peer3	Peer4
Zeit	64s	32s	64s	32s	64s
	Peer5	Peer6	Peer7	Peer8	Peer9
Zeit	44s	64s	46s	64s	46s
	Peer10	Peer11	Peer12	Peer13	Peer14
Zeit	64s	43s	64s	44s	64s

Tabelle 5.5: Zeiten der Peers zum fertigen Herunterladen der Testdatei beim Beziehen in Gruppen

Bandbreiten der Peers gemessen und neue Peers ausgewählt wurden, werden Verbindungen zu schnelleren Peers aufgebaut. Dies hat den Vorteil, dass auch die Peers mit einer nicht so großen Bandbreite am Austausch teilnehmen können, jedoch kann es auch den Download in den ersten Sekunden verlangsamen.

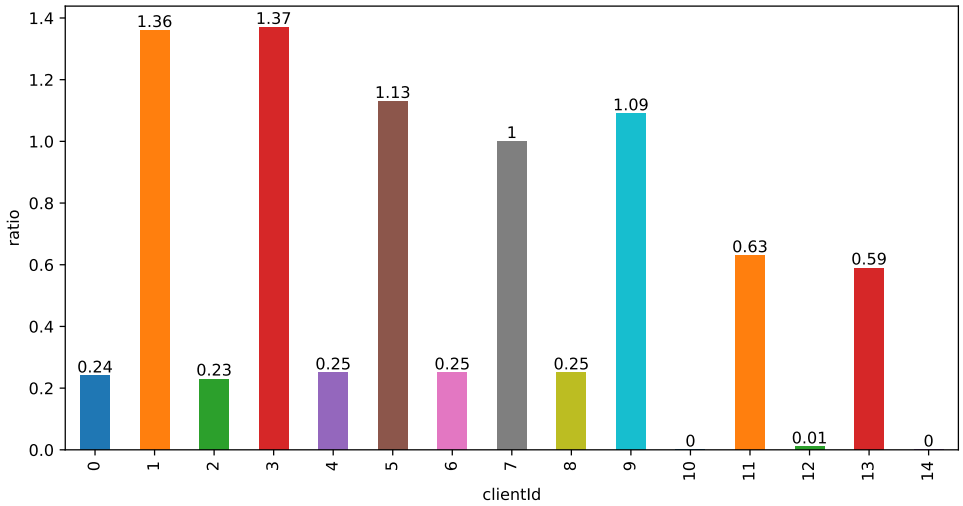

Abbildung 5.6: Finale Ratios der Peers beim Beziehen der Testdatei in Gruppen

Abbildung 5.6 zeigt die finalen Ratios der Peers in dieser Messung. Die Peers mit der Mobilfunkgeschwindigkeit, bis auf die in der letzten Gruppe, weisen jeweils eine Ratio von ca. 0.24 auf. Die Peers mit der höheren Upload-Rate haben mehr Vielfalt in der Ratio, jedoch sind alle Ratios bei diesen höher als bei den mobilen Peers. Da die erste Gruppe nur zwei Peers mit höheren Internetgeschwindigkeiten hatte, konnten die Peers mehr Pieces der Testdatei an die mobilen Peers teilen. Die schnellen Peers in der zweiten Gruppe konnten nicht die Ratios der ersten Gruppe einholen, da in der zweiten Gruppe

nur zwei mobile Peers den Download starteten. Des Weiteren unterstützten die Peers aus der ersten Gruppe die Lastenverteilung der zweiten Gruppe. Deswegen haben die schnelleren Peers in der zweiten Gruppe auch weniger Ratio am Ende.

Bei 15 Übertragungen der Testdatei in dieser Messung, ergibt insgesamt eine Ratio von 8.4 unter dem browserbasierten Peer-to-Peer-Netzwerk. Somit haben die Peers 56% der Bandbreite zum Bereitstellen der Testdatei übernommen.

6 Fazit und Ausblick

6.1 Fazit

In dieser Arbeit wurde ein mögliches Konzept zur Reduzierung der Bandbreite beim Bereit-stellen von statischen Inhalten mithilfe von browserbasierten Peer-to-Peer-Netzwerken vorgestellt und implementiert.

Die Idee ist BitTorrent, ein Peer-to-Peer Protokoll zur Distribution von Dateien, im Browser zu benutzen. Das Protokoll bietet eine Erweiterung zum Beziehen von Teilen der Datei über HTTP. Damit wäre es möglich, die Vorteile der Client-Sever-Architektur mit den Vorteilen der Peer-to-Peer-Architektur in einer hybriden Architektur zu vereinen. Damit ein Browser direkte Verbindungen zu anderen Browsern herstellen kann, musste zum Stand dieser Arbeit WebRTC verwendet werden. WebRTC wird von allen Browsern unterstützt. Zum Finden und Paaren der Browsern wird ein Signaling-Server gebraucht. Das BitTorrent-Protokoll benutzt auch Tracker-Server zum Finden und Paaren von Clients in einem Peer-to-Peer-Netzwerk. Die Rolle eines Signaling-Servers wird somit von einem Torrent-Tracker übernommen. Das entworfene System besteht somit aus drei Hauptkomponenten:

1. **Webserver:** stellt die Webapplikation und nötigen Dateien über HTTP bereit
2. **Torrent-Tracker:** übernimmt die Rolle eines Signaling-Servers zum Finden und Paaren der Clients über Websocket
3. **Webapplikation:** startet einen BitTorrent-Client im Browser, der über WebRTC zu anderen Browsern eine direkte Verbindung aufnehmen kann, um Daten auszutauschen

WebTorrent ist eine Implementierung des BitTorrent-Protokolls für Browser. Mit WebTorrent konnte die Beispielanwendung umgesetzt werden. In Messungen wurde die Effektivität und die Performance des browserbasierten Peer-to-Peer-Netzwerks geprüft. Die

Ergebnisse zeigten, dass im implementierten System, die Browser zur Reduzierung der Bandbreite beim Bereitstellen von statischen Inhalten beigetragen haben.

Jedoch wurden auch Nachteile des Systems aufgedeckt. Der WebTorrent-Client stellt für jeden kleinen Teil einer Datei eine eigene HTTP-Anfrage. Somit wurde der Server mit vielen Anfragen überhäuft. Auch sollte nicht jeder statische Inhalt über das System geteilt werden. Der Torrent-Tracker und somit auch der Signaling-Server brauchen einige Sekunden zum Finden und Paaren der Browser. Die Dateien müssen also groß genug sein, damit es überhaupt zum Austausch der Datei von den Peers kommen kann. Das Bereitstellen von großen Videos, großen Bildern und großen Audiodateien könnte von diesem System profitieren.

6.2 Ausblick

Der Konsum von Medien im Internet wird weiter steigen. Die Internetinfrastruktur wird immer weiter ausgebaut und immer mehr Ansprüche an Systeme zum Bereitstellen von Medien werden gesetzt.

Das in dieser Arbeit vorgestellte Konzept zeigt viel Potential, in einigen Messungen konnten große Anteile der Bandbreite vom browserbasierten Peer-to-Peer-Netzwerk übernommen werden. Jedoch gibt es immer noch viele Probleme, die eine Adoption des Systems verhindern.

Das System könnte von einer besseren BitTorrent Implementierung profitieren. Mit WebAssembly ist es den Browsern seit kurzer Zeit auch möglich, Sprachen wie Rust und C++ statt nur Javascript für Bibliotheken zu verwenden [21]. Jedoch könnten auch alternative Peer-to-Peer-Netzwerke in den Browser gebracht werden. Mit einem anderen Peer-to-Peer-Netzwerk könnten vermutlich auch dynamische Inhalte, besonders Live-Streams, unter den Usern geteilt werden, da BitTorrent in diesem Fall limitiert ist.

Für Videos könnte die adaptive Auflösungsänderung implementiert werden. YouTube setzt das adaptive Streamen mit DASH um [7]. Man könnte schauen, ob DASH auch in das vorgestellte System implementieren könnte.

DRM und Copyright sind auch große Probleme, die beim Austauschen von Dateien zwischen Usern entstehen könnten und für die Arbeit außer Acht gelassen wurden. Dies müsste ausführlich untersucht werden, da jedes Land diesbezüglich andere Regeln hat.

Literaturverzeichnis

[1] Burford, Michael: *WebSeed - HTTP/FTP Seeding (GetRight style).* 2008. – https://www.bittorrent.org/beps/bep_0019.html, Last accessed on 2022-04-19

[2] Cisco: *VNI Complete Forecast Highlights.* 2018. – https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Business_Highlights.pdf, Last accessed on 2022-04-19

[3] Cullen, Cam: *Sandvine releases 2019 Global Internet Phenomena Report.* 2019. – https://www.sandvine.com/press-releases/sandvine-releases-2019-global-internet-phenomena-report, Last accessed on 2022-04-20

[4] Dutton, Sam: *Get Started with WebRTC.* 2020. – https://www.html5rocks.com/en/tutorials/webrtc/basics/, Last accessed on 2022-05-30

[5] Feross ; Abi ; John: *PeerCDN Acquired by Yahoo!* 2013. – https://web.archive.org/web/20150810065820/https://peercdn.com/, Last accessed on 2022-04-20

[6] Florescu, Bogdan ; Andreica, Mugurel I.: Towards a Peer-Assisted Content Delivery Architecture. In: *Proceedings of the 18th International Conference on Control Systems and Computer Science (CSCS)*, 2011, S. 521–528. – ISSN 2066-4451

[7] Google, Youtube: *Delivering Live YouTube Content via DASH.* 2019. – https://developers.google.com/youtube/v3/live/guides/encoding-with-dash, Last accessed on 2022-06-10

[8] Grigorik, Ilya: *High Performance Browser Networking: What every web developer should know about networking and web performance.* O'Reilly Media, 2013. – 309–348 S. – ISBN 1449344763,9781449344764

[9] Hazel, Greg ; Norberg, Arvid: *Extension for Peers to Send Metadata Files*. 2008. – https://www.bittorrent.org/beps/bep_0009.html, Last accessed on 2022-04-19

[10] Hoffman, John ; DeHackEd: *HTTP Seeding*. 2008. – https://www.bittorrent.org/beps/bep_0017.html, Last accessed on 2022-04-19

[11] Krishnan, S. S. ; Sitaraman, Ramesh K.: Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. In: *Proceedings of the 2012 Internet Measurement Conference*. New York, NY, USA : Association for Computing Machinery, 2012 (IMC '12), S. 211–224. – URL https://doi.org/10.1145/2398776.2398799. – ISBN 9781450317054

[12] Kurose, James F. ; Ross, Keith W.: *Computer Networking - A Top-down Approach*. 8th. Pearson, 2020. – ISBN 9780136681557

[13] Kurose, James F. ; Ross, Keith W.: *Computer Networking - A Top-down Approach*. 8th. Pearson, 2020. – ISBN 9780136681557

[14] Loewenstern, Andrew ; Norberg, Arvid: *DHT Protocol*. 2008. – https://www.bittorrent.org/beps/bep_0005.html, Last accessed on 2022-04-19

[15] Mozilla ; contributors, MDN: *WebRTC connectivity*. 2022. – https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity#the_entire_exchange_in_a_complicated_diagram, Last accessed on 2022-05-19

[16] Murley, Paul ; Ma, Zane ; Mason, Joshua ; Bailey, Michael ; Kharraz, Amin: WebSocket Adoption and the Landscape of the Real-Time Web. In: *Proceedings of the Web Conference 2021*. New York, NY, USA : Association for Computing Machinery, 2021 (WWW '21), S. 1192–1203. – URL https://doi.org/10.1145/3442381.3450063. – ISBN 9781450383127

[17] Nemeth, E. ; Snyder, G. ; Hein, T.R. ; Mackin, D. ; Whaley, B.: *UNIX and Linux System Administration Handbook*. Addison-Wesley, 2018. – 1100–1100 S. – URL https://books.google.de/books?id=qtDBjwEACAAJ. – ISBN 9780134277554

[18] Ookla, LLC.: *Speedtest Global Index*. 2022. – data retrieved from Global Median Speeds April 2022, https://web.archive.org/web/20220601094539/https://www.speedtest.net/global-index, Last accessed on 2022-06-01

[19] Rabe, L.: *Statistiken zum Thema Mobiles Internet.* 2022. – https://de.statista.com/themen/258/mobiles-internet, Last accessed on 2022-05-30

[20] Skvorc, D. ; Horvat, M. ; Srbljic, S.: Performance evaluation of Websocket protocol for implementation of full-duplex web streams. In: *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2014, S. 1003–1008

[21] Stepanyan, Ingvar: *Using WebAssembly threads from C, C++ and Rust.* 2021. – https://web.dev/webassembly-threads/, Last accessed on 2022-06-10

A Anhang

Der Anhang befindet sich zusätzlich zur Thesis auf der beigelegten CD.

A.1 Inhalt der CD

Das Projekt ist im HAW-internen GitLab unter https://git.haw-hamburg.de/acn077/webtorrent-cdn einsehbar. Die CD enthält eine Kopie des Projekts. Die Kopie entspricht dem Commit 09d66f42a2baf67b31b524062de1373d86c92d08 des Git-Repos. Die kurze Version des Commits ist 09d66f42. Zur Verhinderung von zukünftigen Kompatibilitätsproblemen ist die gebündelte Webapplikation auf der CD mit enthalten. Es reicht also, nur den Go-Webserver zu starten. Zusätzlich wurde die Webapplikation und der Webserver in einer kompilierten Binärdatei mit dem Namen „app“ zusammengefasst. Die Beispielanwendung kann so auf einem Linux-System ohne zusätzlichen Dependencies ausgeführt werden. Ebenso befinden sich die node-modules von Puppeteer mit der Chromium Version 100 auf der CD.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original