MASTER'S THESIS

DISTRIBUTION OF LARGE DATA IN NETWORKS WITH LIMITED BANDWIDTH

WEBBASIERTE VERTEILUNG GROSSER DATENMENGEN IN LOKALEN NETZWERKEN

TIM FRIEDRICH

CHAIR
Internet Technologies and Systems

SUPERVISORS
Prof. Dr. Christoph Meinel
Dipl.-Inf. (FH). Jan Renz

October 5th, 2016

[September 7, 2019 at 13:51 –]

Tim Friedrich: *Distribution of large data in networks with limited bandwidth,* Master's Thesis, © October 5th, 2016

ABSTRACT

My english abstract

ZUSAMMENFASSUNG

Meine deutsche Zusammenfassung

ACKNOWLEDGMENTS

I would like to thank

CONTENTS

ACRONYMS

1

EINLEITUNG

1.1 MOTIVATION

In den letzten Jahren hat sich das verwendete Datenvolumen des Internets immer weiter gesteigert. Waren es 2015 noch monatlich 72 Petabyte pro Monat sind es 2016 bereits 96 Petabyte. Zwar ist die vorhandene Bandbreite bei vielen Nutzern ebenfalls gestiegen jedoch bezieht sich die vor allem auf Ballungsgebiete. In ländlicheren Regionen ist die Bandbreite in Deutschland in vielen Fällen weiterhin nicht ausreichend. Insbesondere wenn viele Nutzer sich gemeinsam eine Internet Anbindung teilen müssen ist dies ein Problem.*Studie*

Immer mehr Unternehmen halten Ihre Hauptversammlungen, Kundgebungen, und Pressemitteilungen über Live Streams im Internet ab.*Studie* Dies stellt sie vor das Problem das trotz oftmals guter Internetanbindung zu viele Mitarbeiter das Video über die Internetanbindung laden müssen, was zu einer vollständigen Auslastung des WANs führen kann. Dies wiederum kann zur Folge haben, dass ein Arbeiten für die restliche Belegschaft schwierig bis unmöglich wird. Der dadurch entstandene Schaden ist oft nur schwer zu beziffern, geht jedoch schnell in die Millionen.(klingt ohne ref nicht gut)*Studie*

Nicht nur bei Unternehmen sondern auch in Schulen ist die Digitalisierung auf dem Vormarsch. Zunehmend werden Online-Lernplattformen im Unterricht eingesetzt. Diese Entwicklung wird jedoch stark ausgebremst durch fehlende Internet Bandbreiten. Viele Schulen haben eine schlechtere Internetanbindung als viele privat Haushalte. Um statische Inhalte anzeigen zu können, muss jeder Schüler einer Klasse sich diese über das WAN aus dem Internet herunterladen. Da jedoch Schüler in derselben Klasse oft die gleichen Inhalte benötigen, kann Bandbreite gespart werden, indem diese Inhalte nur einmal über das Internet geladen und anschließend im lokalen Netzwerk verteilt werden.

Beide Anwendungsfälle haben gemeinsam das viele Nutzer die selben Inhalte zur annähernd gleichen Zeit benötigen und zum aktuellen Zeitpunkt häufig über das Internet laden müssen. Diese Zeitliche und inhaltlich Lokalität kann genutzt werden um die benötigte Bandbreite zu reduzieren, indem die Inhalte nur einmal über das WAN geladen und anschließend im lokalen Netzwerk verteilt werden.

Die folgende Arbeit betrachtet den Anwendungsfall des Live-Streaming und den Einsatz von Unterstützender Software in Schulen. Es wird betrachtet ob ein Peer to Peer Ansatz zu einer Verbesserung von Ladezeiten und Netzwerklast betragen kann.

1.2 PROJEKT SCHUL-CLOUD

Das Projekt Schul-Cloud[1] ist ein Gemeinschaftsprojekt des Hasso-Plattner-Instituts und des nationalen Execellence-Schulnetzwerkes (MINT-EC). Im Mai 2017 startete die Pilotphase des Projektes mit insgesamt 27 Schulen. Ziel des Projektes ist die Förderung der Digitalisierung in Schulen. Zu diesem Zweck wurde eine Web basierte Plattform entwickelt die Lehrer und Schüler bei der Unterrichtsvorbereitung, Durchführung und Nachbereitung unterstützen soll.

Lehrer können Kurse anlegen und diese nutzen um Materialien sowie Aufgaben zu verteilen. Schülern ist es über die Plattform möglich Lösungen für Aufgaben einzureichen und ihr Ergebnis einzusehen. Über einen Kalender können sie Ihren Stundenplan abrufen.

Das Projekt wird als Open Source Projekt zur Verfügung gestellt und basiert auf einer Microservices Architektur. Bei diesem Architekturmuster wird die Software aus unabhängigen Softwarekomponenten(Services) zusammengesetzt. Die Komponenten kommunizieren über Schnittstellen, sind aber darüber hinaus eigenständige Entitäten und können von beliebig vielen anderen Komponenten verwendet werden. Durch die Verwendung von Microservices wird eine einfachere Anbindung an bestehende Infrastrukturen ermöglicht. Des weiteren können einzelne Services ersetzt werden um die Plattform an die Anforderungen der Schulen anzupassen. Bereitgestellt wird die Plattform mit Hilfe von Cloud Hosting Ansätzen, bei dem die Infrastruktur zentral und nicht von jeder Schule bereitgestellt wird. Dies ermöglicht eine einfache Skalierung. Neben der Web Anwendung existieren native Apps für Android und IOS.

unsicher wie detailliert ich hier werden soll

1.3 SLIDESYNC

Slidesync ist eine Live Streaming Plattform des Unternehmens Media Event Services. Sie ermöglicht es Live-Streams eigenständig anzulegen und an eine Vielzahl von Nutzern zu verteilen. Die Zielgruppe der Plattform sind mittelständische bis große Unternehmen. Neben dem Self-Service, bei dem die Kunden selbst das Streaming übernehmen, wird auch ein Managed Service angeboten bei dem Media Event Services das Streaming und die Produktion vor Ort übernimmt. Die Plattform ist für eine große Anzahl von Nutzern ausgelegt und ist hochverfügbar um den Ansprüchen von Unternehmen gerecht zu

1 https://schul-cloud.org/

werden. Sie stellt unter anderem Funktionen bereit um Events mit Registrierung und Foliensätzen zu realisieren. Die Plattform wird in Ruby on Rails entwickelt. Und stellt neben den Anwendungsservern auch eigen Streaming Server bereit.

1.4 ZIELE

1.4.1 *Forschungsfrage*

- Wie können in einem Netzwerk mit geringerer Internetanbindung Datenintensiven Ressourcen ausgeliefert werden?
- Eignet sich ein Peer to Peer Ansatz um die benötigte Internetbandbreite von Schulen und Unternehmen im Rahmen von Livestreams zu verbessern?

Diese Arbeit wird versuchen die Frage: Wie können in einem Netzwerk mit geringerer Internetanbindung Datenintensiven Ressourcen ausgeliefert werden? zu beantworten.

Dabei wird ein Fokus auf Peer To Peer Technologien gesetzt. Zur Evaluation wird neben simulierten Benchmarks auch der Einsatz unter realen Bedingung getestet.

1.5

Hier muss klar sein was gemacht werden soll !!!

2

ABGRENZUNG

2.1 ANNAHMEN

2.2 TECHNOLOGISCHE ABGRENZUNG

- mehrere nutzer die gleiche ressourcen abrufen - Browserbasiertes P2P CDN -

3

GRUNDLAGEN

3.1 RESSOURCEN

3.1.1 *Statische Ressourcen*

Statische Ressourcen sind Inhalte einer Website die für alle Nutzer gleich sind. Sie sind im Gegensatz zu dynamischen Inhalten nicht nutzerspezifisch und können daher gut über ein CDN verteilt werden. Insbesondere die so genannten Assets einer Internetseite sind meist statisch. Dies sind meist Javascript, CSS aber auch Bild Dateien. Auch Videos fallen häufig in diese Kategorie.

3.1.2 *Dynamisch generierte Ressourcen*

Dynamisch generierte Ressourcen werden zur Laufzeit der Website erzeugt und werden nicht im Vorfeld festgelegt. Dabei lässt sich zwischen Nutzergenerierten Inhalten und automatisch generierten Inhalten, z.b. Statistiken, unterscheiden.

Dynamische Inhalte sowohl Nutzer spezifisch sein, in diesem Fall werden jedem Nutzer bei selber abfrage andere Inhalte angezeigt.

3.2 CDN

Detaillierter Komponenten Beschreiben plus Grafik. Siehe CDN Paper

Unter einem CDN, auch Content Delivery Network versteht man ein Netzwerk in dem sich Clients Inhalte von einer Reihe von Knoten laden. Ein CDN stellt dem Nutzer Auslieferungs- und Speicherkapazitäten zur Verfügung. Dadurch kann die Last auf dem Ursprungsserver und die Latenz auf Seiten der Nutzer reduziert werden. Die reduzierten Ladezeiten werden unter anderem durch eine bessere geographische Nähe und damit geringerer Netzlaufzeiten erreicht.

Es lassen sich drei Klassen von CDNs unterscheiden. Infrastruktur basierte CDN die auf einer geografisch verteilten Server Infrastruktur basieren, Peer To Peer basierte CDNs bei denen die Inhalte direkt zwischen den Teilnehmern verteilt werden und Hybride CDNs die aus einer Kombination aus Server Infrastruktur und Peer To Peer Verteilung beruhen.

3.2.1 *Infrastruktur basierte CDNs*

Infrastruktur basierte CDNs bestehen aus einem Ursprungsservern, der von dem Bereitsteller der Inhalte kontrolliert wird, und einem Netzwerk aus replica Servern. Die replica Server übernehmen die Verteilung der Inhalte an die Clients. Sie fungieren als ein möglichst regionaler cache in dem Inhalte des Ursprungsservers gespiegelt werden. Ein Distributionssystem ist dafür verantwortlich die Inhalte auf den replicas zu aktualisieren und übernimmt das Routing bei einer Anfrage eines Clients. Unter Zuhilfenahme verschiedener Metriken versucht das Distributionssystem einen möglichst optimalen replica Server für den Client zu finden. Diese Metriken unterscheiden sich zwischen den Anbietern. Häufig werden jedoch geographische Entfernung, Latenzzeiten und die Übertragungsrate berücksichtigt. Um eine möglichst geringe Latenz zu erreichen sind Infrastruktur basierte CDNs häufig geografisch sehr verteilt und bestehen aus mehreren tausend replica Servern. So hat Akamai, einer der größten CDN Anbietern, über 137000 Server in 87 Ländern. [10]

3.2.2 *Peer To Peer basierte CDNs*

3.2.3 *Hybrid CDNs*

Hybrid CDNs kombinieren Peer To Peer CDNs und Infrastruktur basierte CDNs. Bei hybriden CDNs wird zuerst versucht die Resource über das Peer Netzwerk zu laden. Ist dies nicht möglich wird auf ein Infrastruktur basiertes CDN zurück gegriffen. Dadurch kann die Last auf dem CDN verringert und durch die Kombination verschiedener CDNs eine bessere Ausfallsicherheit erreicht werden. Häufig kommt diese Art der CDNs zum Einsatz wenn Ressourcen für Websites mit einem Peer To Peer Ansatz verteilt werden sollen. Da in diesem Kontext nicht alle Teilnehmer die technischen Vorraussetzungen mitbringen um an dem Peer To Peer Netzwerk teilzunehmen ist eine entsprechende alternative Lösung nötig. Da die viele Websites bereits mit einem Infrastruktur basierten CDN arbeiten ist es naheliegend dieses weiter zu verwenden.

3.3 VERTEILTE HASHTABELLEN

3.4 PEER TO PEER NETZWERKE

Bei einem Peer To Peer Netzwerk handelt es sich um eine Netzwerk Struktur bei der alle Teilnehmer gleichberechtigt sind. Sie bildet damit das gegen Konzept zur klassischen Client-Server Struktur, bei der einer oder mehrere Server einen Dienst anbieten der von Clients genutzt werden kann. In einem Peer To Peer Netzwerk können die Teilnehmer

sowohl Dienste anbieten als auch nutzen. Typische wenn auch nicht notwendige Charakteristika sind laut Steinmetz[9]:

- Heterogenität der Internetbandbreite der Teilnehmer
- Verfügbarkeit und Qualität der Verbindung zwischen Teilnehmern kann nicht vorausgesetzt werden
- Dienste werden von den Teilnehmern angeboten und genutzt
- Die Teilnehmer bilden ein Netz das auf ein bestehendes Netz aufgesetzt wird(Overlay Netzwerk) und stellen Suchfunktionen bereit
- Es besteht eine Autonomie der Teilnehmer bei der Bereitstellung von Ressourcen
- Das System ist selbstorganisiert
- Die restlichen Systeme müssen nicht skaliert werden und bleiben intakt

Sie lassen sich einteilen in zentralisierte, reine und hybride Peer To Peer Netzwerke. Zentralisierte Netze haben zur Verwaltung einen Server der unter anderem die Verbindung der Teilnehmer übernimmt. Dadurch ist es möglich eine Verbindung aufzubauen ohne das die IP Adresse im Vorfeld bekannt ist. Reine Peer To Peer Netzwerke haben keinen zentralen Verwaltungsserver. Die Verwaltung des Netzwerkes wird von den Teilnehmern selber übernommen. Das hat zur Folge das eine Verbindung nur möglich ist, wenn die IP Adresse des anderen Teilnehmers bekannt ist.

Man unterscheidet zwischen unstrukturierten und strukturierten Peer To Peer Netzwerken.

3.4.1 *Unstrukturierte Peer To Peer Netzwerke*

In unstrukturierten Peer To Peer Netzwerken wird keine Zuordnung von Objekten zu Teilnehmern gespeichert. Um ein Objekt zu finden müssen alle Teilnehmer des Netzwerks gefragt werden.(Flooding) Dadurch steigt die Belastung des Netzwerks mit zunehmender Peer Anzahl.

3.4.2 *Strukturierte Peer To Peer Netzwerke*

Strukturierte Peer To Peer Netzwerke haben eine Zuordnung von Objekt und Teilnehmer. Es ist also möglich gezielt nach einem Objekt zu suchen. Dies wird häufig über verteilte Hash Tabellen, über die mit einem verteilten Index gesucht werden kann, realisiert.

3.4.3 *Topologien*

- Ring
- Fully/pariatial meshed

3.5 WEBRTC- WEB REAL-TIME COMMUNICATION

Webrtc ist ein offener Standard mit dem Echtzeit Kommunikation zwischen Browser und mobilen Anwendungen ermöglicht wird. Mit Hilfe von Webrtc ist es möglich eine Peer To Peer Verbindung zwischen Browsern aufzubauen und Daten direkt zwischen den Clients auszutauschen ohne das externe Plugins erforderlich sind. Insbesondere der Austausch von Multimedia Inhalten wird ermöglicht. Neben der Unterstützung für Video und Audio Inhalten gibt es jedoch auch die Möglichkeit Daten auszutauschen. Webrtc wird vom W3C[2] standarisiert und definiert eine Sammlung von APIs und Protokollen.

Aktuell wird Webrtc von Chrome, Firefox, Safari, Android und iOS unterstützt.[1] Webrtc implementiert drei APIs: MediaStream, RTCPeerConnection und RTCDataChannel die im folgenden genauer beschrieben werden.

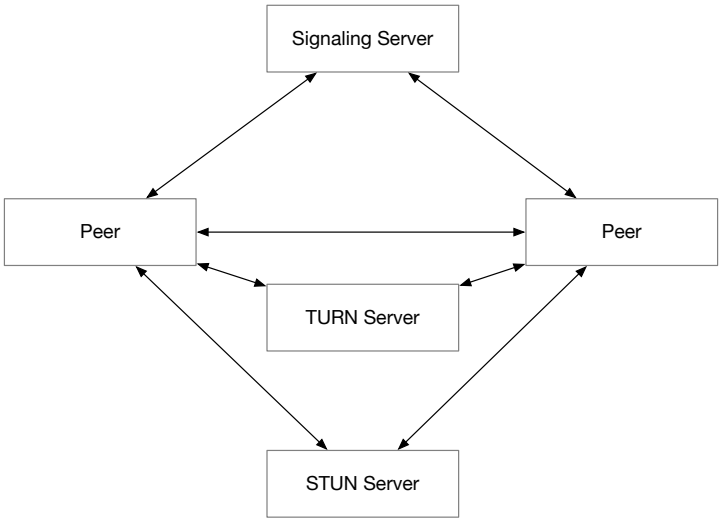

Figure 1: Überblick Server/Client Struktur Webrtc

3.5.1 *Signaling*

Damit zwei Clientsich miteinander verbinden können müssen sie voneinander wissen und Informationen über Metadaten zum Verbindungsaufbau, wie z.B. IP-Adressen und Ports austauschen.

Das Signaling koordiniert die Kommunikation der Verbindungen zwischen Peers. Mit Hilfe des Signalings werden unter anderem die

1 https://caniuse.com/#feat=rtcpeerconnection

Metadaten ausgetauscht, die benötigt werden, um eine erfolgreiche WebRtc Verbindung aufzubauen. Dazu wird das SDP Protokoll verwendet. Unter anderem werden folgende Metdaten ausgetauscht:

- Session Metadaten zum öffnen/schließen von Verbindungen
- Fehler Nachrichten
- Metadaten über die zu übertragenden Medien (z.b. Codecs)
- Schlüsseldaten für verschlüsselte Verbindungen
- Netzwerk Daten wie öffentliche IP Adressen und Ports

Der Webrtc Standart legt keine für das Signaling zu verwendende Technologie und Protokolle fest um die Integration mit bestehenden Technologien zu verbessern und es dem Entwicklern zu ermöglichen das für den Anwendungsfall beste Protokoll zu verwenden. Allerdings legt er fest das eine bidirektionale Kommunikation zwischen den Clientsotwendig ist.

3.5.2 *STUN Server - Simple Traversal of User Datagram Protocol [UDP] Through Network Address Translators*

Da die Anzahl von IPv4 Adressen begrenzt ist, verwenden die meisten Subnetze NATs. Das hat zur Folge das diese Clients nicht wissen wie über welche IP Adresse und welchen Port sie erreichbar sind. Daher ist der Einsatz von STUN Servern nötig um einen Verbindungsaufbau zu ermöglichen. Stun Server überprüfen eingehende Anfragen auf IP Adresse und Port und senden diese Informationen zurück an den Client, der somit in der Lage ist diese Information weiter zureichen und damit auch außerhalb seines lokalen Netzwerkes erreichbar ist. Das STUN-Protokoll ist im RFC 3489 [6] definiert und ist nicht auf Webrtc beschränkt.

3.5.3 *TURN Server*

Verwaltete Netzwerke, wie die von Unternehmen, haben häufig Firewalls und Port blocking Systeme installiert um die Sicherheit des Netzwerks zu gewährleisten. Das kann dazu führen das Webrtc Verbindungen nicht aufgebaut oder Daten nicht über Webrtc Verbindungen übertragen werden können.

TURN Server bieten eine Fallback Lösung für diesen Fall. Sie haben eine öffentliche IP und sind über das Internet erreichbar. Im Fehlerfall kann der Datenverkehr über einen TURN Server geleitet werden, so das die Kommunikation nicht unterbrochen wird.

3.5.4 *SDP - Session Description Protocol*

- Session Description Protocol (SDP, RFC 4566)
- beschreibt Eigenschaften von Eigenschaften von Multimediadatenströmen
- verwaltet kommunikationssitzungen z.b. SIP(IP-telefonie)
- keine aushandlungsmechaniken sondern nur beschreibungen der Datenströme
- v=0 o=Alice 1234 1234 IN IP4 host.provider1.com s=Video von 987654 c=IN IP4 host.provider2.com t=0 0 m=audio 20000 RTP/AVP 97 a=rtpmap:97 iLBC/8000 a=fmtp:97 mode=30 m=video 20001 RTP/AVP 31 a=rtpmap:31 H261/90000
- daten aus eigener anwendung einfügen
- Felder beschreiben? zumindest die wichtigesten/verwendeten

3.5.5 *ICE*

- Methode zur Überwindung von NAT
- Interactive Connectivity Establishment
- https://tools.ietf.org/html/rfc5245
-

3.5.6 *Webrtc API*

RTCPEERCONNECTION

Das RTCPeerConntection Interface repräsentiert eine Verbindung vom lokalen Clientu einem anderen Client Ein RTCPeerConnection Objekt hält den momentanen Zustand der Verbindung ebenso wie Metadaten über den verbundenen Peer. Sie stellt Funktionen zum verwalten von Verbindungen bereit.

RTCDATACHANNEL

- Übertragung von raw data (Bitstreams)
- string, blob, arraybuffer, ArrayBufferView
- Übertragung mit Stream Control Transmission Protocol (SCTP)
- kurze erklärung SCTP
- Verbindungsorientiertes Netzwerkprotololl

- RFC 4960
- Das zuständige Gremium bei der IETF ist die Arbeitsgruppe Signaling Transport, kurz SIGTRAN.xX
- Selbe stufe als Transportprokoll im stack wie TCP/UDP
- Konzept der Association:
- mehrere Nachrichten-Datenströme in sich reihenfolgenerhaltend
- zwischen den Datenströmen muss die Reihenfolge nicht erhalten bleiben
- Multistreaming - Ein host mehere Ips
- Vier wege Handschake
- Hierbei speichert der Server bei einer Verbindungsanfrage (INIT-Paket) keine Zustandsinformationen, sondern schickt diese in Form eines Cookies (INIT-ACK-Paket) an den Client. Der Client muss dieses Cookie in seine Antwort (COOKIE-ECHO-Paket) einfügen und wird damit vom Server als zum Verbindungsaufbau berechtigt erkannt, was dieser ihm bestätigt (COOKIE-ACK-Paket)x

MEDIASTREAM

Die Mediastream Api, auch getUserMedia, ermöglicht es Echtzeit Daten wie audio oder Video aufzunehmen, anzuzeigen und an andere Clients weiter zu leiten und repräsentiert Medien Streams wie z.b. Audio oder Video Streams. Sie ermöglicht unter anderem den Zugriff auf Video Kameras und Mikrofone. Durch Sie ist es möglich auf die Hardwareunterstützung für Videos mittels open GL zuzugreifen. MediaStreams lassen sich mithilfe des src Attributes von HTML 5 video Elementen in das DOM einbinden. MediaStreams wurden von vom W3C in einem eigenen Standart definiert.[3]

3.6 DATACACHE API

Die DataCache Api ermöglicht es Netzwerk Requests zu cachen. Ursprünglich wurde die Api entwickelt um Service Workern die Möglichkeit zu geben einen Cache anzulegen und selbst zu verwalten. Dadurch ist es möglich mithilfe von Service Workern und der DataCache Api Websiten auch verfügbar zu machen wenn kein internet verfügbar ist.

- https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api

3.7 INDEXEDDB

IndexedDB ist ein HTML 5 Feature um Daten im Browser zu speichern. Es wurde vom W3C standarisiert[1] und soll den veralteten Web Sql Standard ablösen. Im Gegensatz zu Web Sql hat die IndexedDB keine strukturierte Query Language und ihr liegt kein relationales Modell zu Grunde. Sie stellt einen Key-Value Store bereit der in der Lage ist auch große Datenmengen effektiv bereit zu stellen. Dabei ist der Datenzugriff auf die selbe Domain beschränkt. Die API ist überwiegend asynchron und basiert auf Promises.

3.8 SERVICE WORKER

Service Worker sind Skripte die im Browser als separate Prozesse im Hintergrund laufen, so genannte Web Worker. Sie stellen die Funktionalitäten eines programmierbaren Netzwerk Proxies bereit. Durch Service Worker ist es möglich die Anfragen einer Seite zu kontrollieren auf sie zu reagieren und in den Prozess einzugreifen.[7] Service Worker haben keinen Zugriff auf das DOM. Sie könne mehrere Browser-Tabs und mit Hilfe des PostMessage Protokolls können Nachrichten zwischen Service Worker und Browser-Tab ausgetauscht werden. Da Service Worker Zugriff auf den DataCache und die IndexDB haben werden sie häufig verwendet um Internetseiten offline verfügbar zu machen. Bei der Registrierung eines Service Workers wird ein URL-Scope fest gelegt für den der Service Worker zuständig ist. Nur Anfragen die sich innerhalb des URL-Scopes des Service Workers befinden können von diesem bearbeitet werden.

3.8.1 *Lebenszyklus*

Nach dem ein Service Worker registriert wurde befindet er sich im Zustand der Installierung. Während der Installation werden häufig Inhalte in den Cache geladen. Wurde der Service Worker erfolgreich installiert wird er aktiviert. Ab diesem Punkt kann er Requests über das fetch event abfangen. Um Arbeitsspeicher zu sparen wird der Service Worker terminiert falls er keine Fetch oder Message events empfängt. Dies hat zu folge das ein Service Worker sich nicht auf den globalen Zustand verlassen kann, sondern statt dessen seinen Zustand auf die IndexedDb ausgelagert werden muss.

3.9 WEBSOCKETS

Websockets ist ein, auf TCP basierendes, Protokoll das bidirektionale Verbindungen zwischen Server und Webanwendung ermöglicht. Nachdem der Client eine Websocket Verbindung zum Server Aufgebaut hat ist es dem Server im Gegensatz zu HTTP möglich ohne vorherige

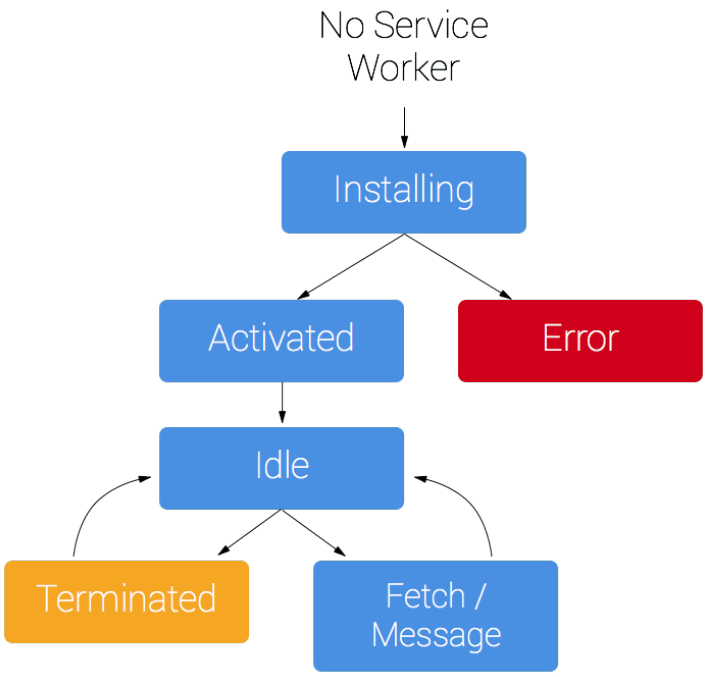

Figure 2: Lebenszyklus eines Service Workers[2]

Anfrage des Clients Daten an ihn zu senden. Zum initiieren einer Verbindung wird ein Handshake durchgeführt der vom Client angestoßen werden muss. Dazu wird wie bei HTTP der Port 80 verwendet. Der Server antwortet bei erfolgreichem Handshake mit dem HTTP Status code 101. Um eine Abwärtskompatibilität zu gewährleisten werden ähnliche Header wie bei HTTP verwendet.

Neben dem unverschlüsseltem URI-Schema ws definiert RFC6455[4] auch das verschlüsselte Schema wss. Da sehr wenig Daten overhead bei der Kommunikation besteht eigenen sich Websockets insbesondere für Anwendungen die eine geringe Latenz benötigen. Websockets werden von allen modernen Browsern unterstützt.

RFC6455[4]

3.10 DISTRIBUTED CACHES

- Hier Algorithmen erklären?
- auf hashes eingehen
- wie ein hash nur verteilt

3.11 IP ADRESSEN

Eine IP Adresse ist ein eindeutiger Identifizier für Computer in einem Netzwerk. Jedem Computer in einem Netzwerk wird eine eindeutige

Table 1: Beispiel einer IP Adresse mit Subnetzmaske

IP-Adresse	145.574.322.	5
Subnetzmaske	255.255.255.	0
	Netzanteil	Hostanteil

IP Adresse zugewiesen über die der Computer adressierbar ist. Dadurch ist der Computer für andere erreichbar. Sie wird benötigt um ein routing vom Sender zum Empfänger zu ermöglichen. [**www**]

3.11.1 *Aufbau von IP Adressen*

IPv4 wurde im Jahr 1981 durch das RFC 791[8] definiert und besteht aus einer 32 stelligen Binärzahl wodurch maximal 4.294.967.296 Adressen dargestellt werden können. Zur besseren Lesbarkeit werden IPv4 Adressen meist als vierer Blöcke in Dezimal geschrieben. IP Adressen bestehen aus einem Netz- und einem Hostanteil. Mit dem Netzanteil wird das Teilnetz indem sich das Gerät befindet beschrieben, während der Hostanteil das Gerät identifiziert. Durch eine Subnetzmaske wird festgelegt welcher Teil der IP-Adresse Host- und welcher Netzanteil ist. Alle Bits die in der Subnetzmaske 1 sind legen den Netzanteil fest - alle bits die 0 sind den Hostanteil.

Aufgrund der stark ansteigenden Zahl an Geräten die mit dem Internet verbunden sind ist der Adressbereich der IPv4 Adressen nicht mehr ausreichend. Entwickelte der ITFE 1998 einen neuen Standard. IPv6 verwendet 128 anstatt der 32 Bits zur Darstellung von IP Adressen. Dadurch ist es möglich 2^{32} Geräte abzubilden. IPv6 wird meist als vier Oktette in hexadezimal dargestellt. Zur Unterscheidung von Host- und Netzanteil werden Präfixlängen angegeben.

3.11.2 *Network Adress Translation(NAT)*

Mit Hilfe von NAT können mehrere Geräte über die selbe öffentliche IP-Adresse über das Internet verbunden werden. Dadurch ist es möglich trotz des begrenzten Adressraums von IPv4 mehr Geräte mit dem Internet zu verbinden. Dazu werden werden vom die IP-Adress header Felder der Datenpackete verändert.

3.12 RUBY ON RAILS

Ruby on Rails ist ein in Ruby geschriebenes Opensource Webframework. Zentraler Ansatz der Entwicklung des Frameworks ist es, es Software Entwicklern einfacher zu machen Webanwendungen zu

schreiben. Die Philosophie des Frameworks beinhaltet zwei Prinzipien:[3]

Don't Repeat Yourself(DRY): Jede Information und Funktionalität soll einen Repräsentation im System haben. Dadurch soll erreicht werden das die Software einfacher wartbar, übersichtlicher und fehlerfreier sein.

Convention over Configuration: Um die Entwicklung für den Programmierer zu erleichtern werden annahmen getroffen und Standard Einstellungen festgelegt. Diese lassen sich zwar durch die Entwickler ändern sind zu beginn jedes Projektes erst einmal gleich. So werden z.b. auch Vereinbarungen über die Benennung von Methoden und Klassen getroffen(Naming Conventions).

Ruby on Rails ist ein Model View Controller basiertes Framework. *MVC Beschreiben?*

3.13 TURBOLINKS

Turbolinks[4] ist eine Javascript Bibliothek zur Beschleunigung der Navigation auf Internetseiten. Klick ein Nutzer auf einen Link wird der Seitenabruf unterbrochen und stattdessen mit Ajax geladen. Anschließend überprüft Turbolinks welche Teile der Seite sich verändert haben und rendert nur diese Elemente. Dadurch entsteht ein flüssigeres Nutzererlebnis da Teile der Seite bestehen bleiben und nicht ersetzt werden. Da nachgeladene Ressourcen die sich bei der Navigation nicht ändern nicht erneut geladen werden müssen beschleunigt sich die Ladezeit. Javascript scripts müssen nicht neu geladen werden sondern bleiben in ihrem vorigen Zustand bestehen da kein kompletter Pageload vorgenommen werden muss.

3.14 SINGLE PAGE APPLICATIONS

Unter Single Page Applications(SPA) versteht man Webanwendungen bei denen der Client aus einem einzigen HTML Dokument besteht. Inhalte werden meist dynamisch mit AJAX oder Websocket nachgeladen. Dadurch entsteht ein Nutzererlebnis das flüssiger erscheint da Seiten nicht bei Navigation komplett neu geladen werden müssen. Sie bieten sich für Anwendungen mit hoher Nutzerzahl an da der Aufwand der HTML renderings com Server auf den Client verlagert wird. Da das Backend zumeist nur Daten ausliefert wird die Entwicklung nativer Clients für Mobilgeräte vereinfacht da zumeist das Backend wiederverwendet werden kann. *Woher quellen nehmen??? online??*

3 https://guides.rubyonrails.org/
4 https://github.com/turbolinks/turbolinks

3.15 REDIS

Bei Redis handelt es sich um einen In-Memory Datenstruktur Speicher der als LRU cache, Datenbank oder Message Broker verwendet werden kann.[5] Redis ist ein Opensource Projekt und wird von Redis Labs gesponsert. Jede gespeicherte Datenstruktur ist über einen Key abrufbar.

Unter anderem unterstützt Redis folgende Datentypen: Set: Bei Sets handelt es sich um Mengen die Strings beinhalten können. Wobei jeder String nur einmal pro Set vorkommt. Hinzufügen von Element, Löschen von Elementen und das prüfen ob ein Element in einem Set vorhanden ist passieren in konstanter Zeit. (O(1))

Strings: Redis Strings sind binary safe, das heißt sie können jegliche Daten, z.B. auch Bilddaten, enthalten. Strings können auch als atomare Counter verwendet werden. Dazu stellt Redis Kommandos bereit zum erhöhen oder subtrahieren von Strings.

List: Listen enthalten eine geordnete Liste von Strings, in der Reihenfolge in der sie hinzugefügt wurden. Elemente können sowohl vorne als auch hinten in die Liste eingefügt werden

5 https://redis.io/

4

KONZEPT

4.1 NETZWERK STRUKTUREN

Anwendungsfall Analyse? – evt in Einleitung schieben.

Im Folgenden wird betrachtet wie die Netzwerkstruktur der Clients im Falle von Slidesync und Schul-Cloud zusammensetzen.

4.1.1 *Schul-Cloud*

Bei der Schul-Cloud lassen sich im wesentlichen zwei Anwendungsszenarien unterscheiden. Zum einen die Anwendung im Unterricht. Der Lehrer stellt z.B. eine Aufgabe die mit Hilfe der Schul-Cloud durchgeführt werden soll. Daraufhin besuchen die Schüler die entsprechende Seite und bearbeiten die Aufgabe. In einem kurzen Zeitfenster laden also mehrer Schüler während sie sich im gleichen lokalen Netzwerk befinden, dem der Schule, die selben Inhalte herunter. Bei dem anderen Szenario wird die Schul-Cloud außerhalb des Unterrichts genutzt. Z.B. bereitet der Lehrer den Unterricht vor oder die Schüler bearbeiten gestellte Hausaufgaben. Die Nutzer befinden sich nicht zwangsläufig im selben Netzwerk. Auch Laden die Nutzer die selben Daten nicht notwendigerweise in einem kurz Zeitfenster sondern verteilt über einen längeren Zeitraum. Es findet jedoch auch keine so starke Auslastung des Netzwerks statt. Deshalb wird im Rahmen dieser Arbeit vor allem das erste Szenario betrachtet.

4.1.2 *Slidesync*

Die Verteilung der Clients auf Netzwerke kann sich bei Slidesync von Event zu Event stark unterscheiden. Da sich Slidesync jedoch hauptsächlich für Streams von mittleren bis großen Unternehmen wendet, lässt sich beobachten das viele der Nutzer sich gemeinsam in einem lokalen Netzwerk, einem Standort, befinden. Um die Last der Unternehmensnetzwerke zu reduzieren, werden bei einigen Unternehmen caching Server eingesetzt. Betrachtet man 10 Events mit caching Infrastruktur im Internen netz stellt man fest das 64% der Teilnehmer aus dem internen Netz auf das Event zugegriffen haben. *caching erklären*
In dieser Arbeit wird betrachtet wie die Last auf das interne Netz

reduziert werden kann ohne das zusätzliche caching Server eingesetzt werden müssen.

4.1.3 *Gemeinsamkeiten*

-

4.2 ARCHITEKTUR

Schul und Unternehmens Netzwerke sind meistens so aufgebaut das viele Clients über einen oder mehrere WAN Anbindungen mit dem Internet verbunden sind. Werden Ressourcen geladen müssen diese über das WAN geladen werden. Dies ist in der Regel auch der Fall wenn mehrere Clients die selben Ressourcen benötigen. Abbildung 3 zeigt den typischen Aufbau eines solchen Netzwerks. Übersteigt die benötigte Bandbreite der Clients die der durch as WAN zur Verfügung gestellten, so kommt es zu mitunter sehr teuren Netzwerk Ausfällen die ganze Unternehmensstandorte betreffen können. Durch die dadurch resultierenden langen Ladezeiten kann es zu einer starken Einschränkung des Nutzererlebnisses und der Nutzerzufriedenheit kommen.[5] Um dem entgegen zu wirken wird versucht mit caching Appliances den Datenverkehr der über das Internet geladen werden muss zu reduzieren. Slidesync z.B. bietet dazu Unternehmen ein eigenes lokales CDN an bei dem Server in dem Netzwerk betrieben werden. Dies verursacht jedoch kosten und Konfigurationsaufwand. Damit eigenen es sich nur für größere Unternehmen die den Service häufig nutzen.

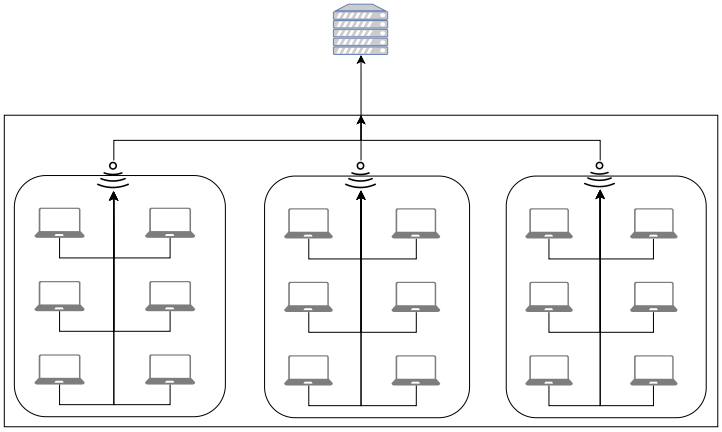

Figure 3: Netzwerkverkehr in einem herkömmlichen Netzwerk

In den betrachteten Anwendungsfällen besteht eine hohe zeitlich und inhaltliche Lokalität der Daten. Dies kann genutzt werden um die benötigte Bandbreite zu reduzieren. Dazu soll im Folgenden eine interne Verteilung mittels eines hybriden Peer To Peer CDNs untersucht werden. Abbildung 7 zeigt exemplarisch den Aufbau eines solchen

Netzwerkes. Anstatt das jeder Client sich die Ressource von einem externen Server lädt, lädt nur noch ein Nutzer je Subnetz die Resource über das WAN. Dieser verteilt die Resource dann im internen Netzwerk an andere Clients die diese dann ebenfalls wieder bereitstellen.

Benötigt ein Client eine Resource versucht er zunächst die Resource über sein Peer To Peer Mesh zu laden. Ist dies nicht möglich lädt er sie über einen externen Server. Hat ein Peer eine Resource geladen speichert er sie zwischen und stellt sie für andere Clients bereit.

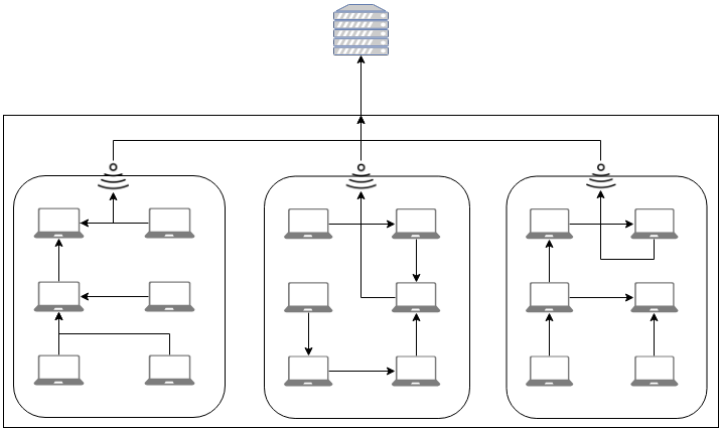

Figure 4: Netzwerkverkehr in einem Peer To Peer CDN

Da sowohl im Kontext der Schule als auch bei Unternehmen kein Wissen im Bereich der Computer Administration seitens der Nutzer vorausgesetzt werden kann, muss ein Ansatz gewählt werden der keine Installation auf Seiten der Nutzer benötigt.

Um dies zu erreichen wird eine Kombination aus Webrtc und Service Workern verwendet. Der Javascript Code lässt sich als ein Plugin einbinden und erfordert nur geringen Konfigurationsaufwand seitens der Anwendungsentwickler. Da sich die Art der Seitennutzung von Anwendung zu Anwendung jedoch stark unterscheidet muss das Peer Meshing serverseitig für jede Anwendung geschrieben werden. So kann Domainenspezifisches wissen ausgenutzt werden um eine bessere überlappung der von den Clients benötigten Ressourcen zu erreichen.

Die eingesetzte Technologie zur Übertragung von Daten zwischen Browsern ist WebRTC. WebRTC ist ein offener Standard und ermöglicht es Browser paarweise zwecks Datenaustausch zu verbinden. Der große Vorteil dieser Technologie ist, dass sie direkt von modernen Browsern unterstützt wird, wodurch keine zusätzliche Software installiert werden muss. Konkret werden Webrtc DataChannel genutzt.

Für den Datenaustausch müssen wechselseitig DataChannel zueinander aufgebaut werden. Die Ausgangslage ist, dass die Peers wissen, dass es den anderen gibt, aber nicht wie der jeweils andere zu erreichen ist. Um diese Problematik zu lösen, existiert ein Vermittlungsserver (Signaling server).

Als erstes werden Informationen, über die Verbindung die aufgebaut werden soll, an den Signaling server gesendet. Es wird ein SDP-Offer gesendet(SDP für Session Description Protocol). Dieses SDP-Offer leitet der Signaling server an die Peers in dem selben Mesh sind weiter. Geantwortet wird mit einer SDP-Answer, welche Informationen über die abgestimmte Verbindung enthält und über den Signaling server zurück geleitet wird.

Damit eine direkte Verbindung aufgebaut werden kann, müssen über den Signaling server noch weitere Informationen wie ICE-Kandidaten ausgetauscht werden. ICE steht hierbei für Interactive Connectivity Establishment und ist fester Bestandteil von WebRTC. Es ist für den Aufbau der Browser-zu-Browser-Verbindung verantwortlich. ICE-Kandidaten enthalten hauptsächlich Informationen darüber wie ein bestimmter Nutzer erreichbar ist (also z.B. private oder öffentliche IP-Adresse). Ermittelt werden diese ICE-Kandidaten mithilfe eines STUN-Servers und dem dazugehörigen Session Traversal Utilities for NAT (STUN) Protokoll. Wie der Name des Protokolls schon verrät, wird es vor allem benötigt um auch Nutzer erreichen zu können die keine eigene öffentliche IP-Adresse besitzen, bei denen also Network Address Translation (NAT) eingesetzt wird. Dies ist aufgrund der mangelnden Anzahl an IPv4-Adressen bei fast jedem Internetnutzer der Fall.

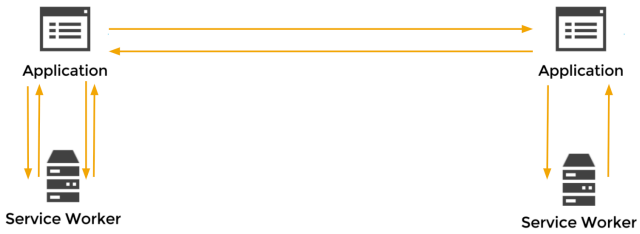

Figure 5: Service Worker - Webrtc

4.3 JAVASCRIPT PROXIES - ABFANGEN VON ANFRAGE

Damit ein Client-seitiges CDN möglich ist, ist es notwendig das die Abfragen des Browsers abgefangen und auf anderem Weg beantwortet werden können. Nachdem der Browser nach einer Anfrage URL augelöst hat(DNS-lookup) lädt er die abgefragte Seite. Ist die Seite geladen beginnt der Browser die im HTML Dokument verlinkten

Dokumente zu laden. Das sind neben Bildern auch CSS und Javascript Dateien. Ein CDN muss in der Lage sein auf all diese Anfragen reagieren zu können.

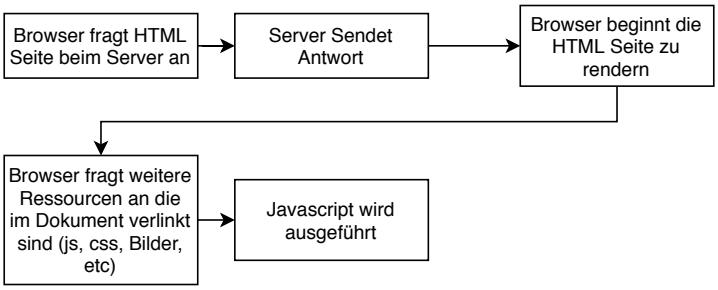

Figure 6: Ablauf einer HTML abfrage im Browser

Um dies zu realisieren gibt es verschiedene Möglichkeiten. Turbolinks unterbricht die Weiterleitung nachdem ein Link angeklickt wurde und lädt die abgefragte Seite mit Ajax. Dadurch ist es möglich die zu zeichnenden Elemente selbst auszuwählen und manuell teile der Seite zu cachen. Dieser Ansatz ließe sich auch für ein CDN verwenden. Allerdings ist es nötig die Javascript Page load Events durch eigene Events zu ersetzen und bestimmte Teile des Javascript Codes umzuschreiben. Javascript Code wird bei diesem Ansatz nach Navigation auf eine neue Seite nicht neu geladen. Auch wenn dies die Ladezeiten verringert ist eine Integration ohne Anpassung des Anwendungscodes nicht möglich. Ebenfalls ist es nicht möglich Anfragen abzufangen die beim ersten Besuch der Seite entstehen, sondern nur solche die nach weiterer Navigation entstehen.

Eine weitere Möglichkeit besteht darin eigene HTML tags einzuführen und diese nachdem die eigentliche Seite und das CDN script geladen wurde mit Ajax nachzuladen. Dadurch lässt sich mit Javascript kontrollieren woher die Ressource geladen werden soll. Allerdings können Ressourcen die über das Peer To Peer CDN geladen werden sollen erst geladen werden wenn das komplette HTML Dokument und das CDN script geladen sind. Dies kann die Ladezeiten beeinflussen und ebenfalls Anpassungen im Javascript Code der Anwendung notwendig machen. Wird in einer nachgeladenen Javascript Datei ein Eventhandler auf ein Event registriert das bereits gefeuert wurde, so wird dieser Code nicht mehr ausgeführt.

Service Worker sind eigene Prozesse die in einem anderen Kontext laufen als die eigentliche Webseite. Einmal registriert existieren sie und fungieren als proxy, unabhängig davon ob die Webseite gerade geladen ist oder nicht. Besucht ein Nutzer die Seite wird der Service Worker geladen. Kehrt er wieder so ist der Service Worker bereits aktiv und kann Anfragen des Browsers abfangen. Da einer der Anwendungsfälle für Service Worker das Offline verfügbar machen von Webanwendungen ist, verfügen sie über Unterstützung von Caching-APIs. Durch die Caching-API ist es möglich Anfragen zu speichern

und zu einem späteren Zeitpunkt wieder abzurufen. Somit ist es nicht nur möglich eigene Anfragen aus dem Cache zu beantworten, sondern ebenfalls gespeicherte Ressourcen an andere Clients auf Anfrage weiter zu leiten. Daher eigenen sie sich gut für die Verwendung als Proxy in einem Clientseitigen CDN.

4.4 VERBINDEN VON PEERS - SIGNALING

Um eine Verbindung zwischen den Peers aufzubauen ist ein Signaling Process erforderlich. Der Webrtc Standard schreibt nicht vor wie das Signaling durchgeführt werden soll, jedoch bieten sich hierzu Websockets an, da eine bidirektionale Kommunikation notwendig ist. Da das Schul-Cloud backend in Nodejs und Slidesync in Ruby on Rails programmiert sind biete es sich an eine websocket Implementierung zu wählen die für beide Backends Schnittstellen anbietet. Faye[1] bietet neben Einem Browser-Client auch Backend Clients für verschiedene Programmiersprachen, darunter auch Nodejs und Ruby an.

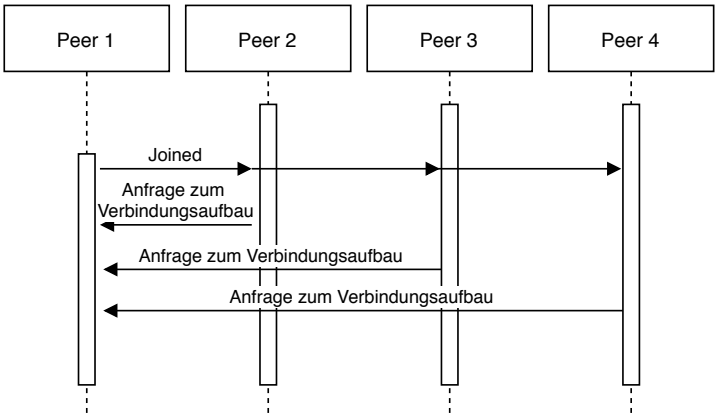

Figure 7: Signaling Ablauf

Tritt ein Client einem Peer Mesh bei, so sendet er eine Nachricht mit seiner eigenen PeerId auf einen Websocket channel(Prefix/joined). Alle Peers des Meshes sind auf diesem Websocket Channel registriert und empfangen die Nachricht. Empfängt ein Peer die Nachricht das ein neuer Peer dem Netzwerk beigetreten ist, beginnt er eine Webrtc Verbindung zu dem Peer aufzubauen.

Abbildung 8 zeigt den Verbindungsaufbau zwischen zwei Peers. Nachdem Peer1 über den Websocket Channel mitgeteilt hat das er dem Peer Mesh beitreten will sendet Peer2 ein SDP-Offer über einen Websocket Channel zu Peer1. Dazu registriert sich Peer1 auf dem Channel 'Prefix/ClientId' über den er für Peer2 erreichbar ist. Hat Peer1 das SDP-Offer erhalten sendet er ein SDP-Answer zurück. Mit den dadurch erhaltenen Informationen tauschen die beiden Clients

1 https://faye.jcoglan.com/

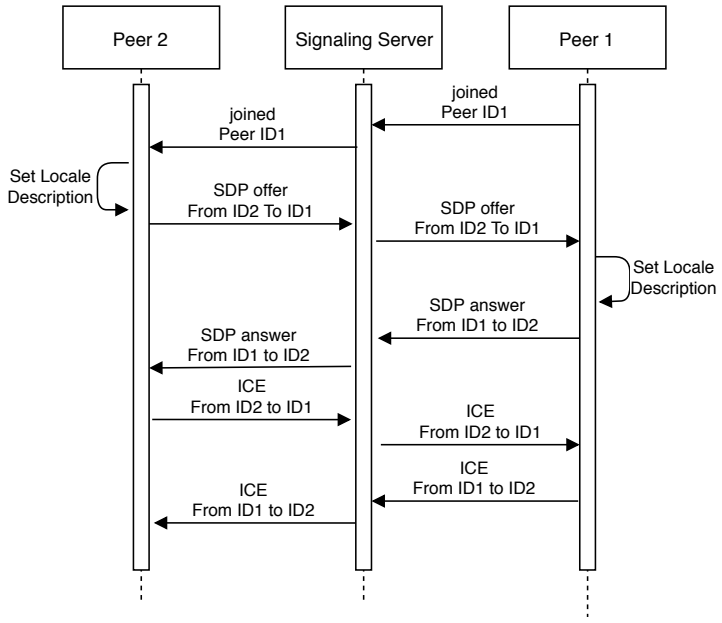

Figure 8: Ablauf Verbindungsaufbau

nun ICE Pakete über den Websocketchannel aus. Im Anschluss können beide Clients über Webrtc direkt miteinander kommunizieren.

Verlässt ein Client das Peer To Peer Netzwerk so müssen die andern Clients darauf reagieren und ihn aus ihrer Liste von Peers löschen. Zwar wäre es möglich eine Nachricht im Falle das ein Client die Seite verlässt zu senden, z.B. mittels dem onbeforeunload[2] Javascript Event, jedoch ist dies sehr unzuverlässig. Im Falle das der Client z.B. die Internetverbindung verliert oder der Computer ausgeschaltet wird kann dieses Event nicht mehr ausgelöst, und somit auch keine Nachricht mehr an die Peers gesendet werden. Daher beobachten die Peers den Status des Webrtc Datachannels. Ändert er seinen zustand zu geschlossen so wird der Peer aus dem Netzwerk entfernt. Auf diesem Weg können fehlerhafte Peers entfernt werden ohne darauf angewiesen zu sein das sie im Fehlerfall noch in der Lage sind eine Nachricht an die anderen Peers zu senden.

4.5 ROUTING - AUFFINDEN VON RESSOURCEN

Das vorgeschlagene Peer To Peer CDN ist als strukturiertes Peer To Peer Netzwerk ohne verteilte Hashtabelle implementiert. Da das auffinden von Ressourcen in einem zeitkritischen Moment erfolgt, während der Ladezeit der Seite und sich bei den betrachteten Anwendungsfällen gut vorher sagen lässt welche Ressourcen benötigt werden erscheint es als sinnvoll das Routing im Vorfeld geschehen zu lassen. Zwar scalieren algorithmen wie kademlia [**kademlia**] sehr gut für eine große Anzahl an Teilnehmern, jedoch ist zum Zeitpunkt

2 https://developer.mozilla.org/en-US/docs/Web/API/WindowEventHandlers/onbeforeunload

der Anfrage einer Ressource ein nicht unerheblicher Kommunikationsaufwand notwendig. Es müssen zwar für z.B. 1000 Teilnehmer im Netzwerk nur drei Teilnehmer gefragt werden um zu ermitteln wer die Ressource speichert, jedoch ist der Verbindungsaufbau mit Webrtc relativ aufwändig. In Tests im Rahmen dieser Arbeit wurden Verbindungsaufbauzeiten von ca. 80ms gemessen. Da im Vorfeld nicht bekannt ist müssen sämtliche Verbindungen im Moment des Routings aufgebaut werden. Für das Beispiel mit 1000 Teilnehmern hieße es das 3 Verbindungen zum auffinden der Ressource hergestellt werden müssen plus eine zu dem Teilnehmer der die Ressource speichert. Die Bearbeitung der Anfrage würde um 320ms Sekunden verzögert werden. Während dies für Filesharing systeme wie Bittorent kein großes Problem darstellt, ist diese Verzögerung für ein CDN zu groß. Daher hällt in dem vorgeschlagenen CDN jeder Teilnehmer eine Hastabelle vor in der gespeichert wird welcher Teilnehmer welche Ressourcen speichert. Diese muss aktualisiert werden was zwar zu einer höheren Last und einem erhöhten Kommunikationsaufwand führt, jedoch kann dies zu einem zeitunkritischen Moment zwischen den Anfragen geschehen. Muss ein Peer eine Ressource auffinden so hält er diese Information bereit.

4.6 MESH ZUORDNUNG

Als Netz Topologie wurde ein voll-vermaschtes Netz gewählt. Jeder Teilnehmer eines Meshes baut also Verbindungen zu jedem anderen Teilnehmer des Meshes auf. Zwar müssen mehr Verbindungen hergestellt werden als bei einem teil-vermaschten Netz oder einer Ring Topologie, jedoch kann ein Ausfall von Teilnehmern besser abgefangen werden. Da mit jedem neuen Peer ein Mehraufwand an Kommunikation entsteht ist die maximale Anzahl an Teilnehmern die sich in einem Mesh befinden können geringer. Dadurch ist es besonders wichtig die Teilnehmer sinnvoll auf die Meshes zu verteilen. Wobei sinnvoll bedeutet das sie eine möglichst große Schnittmenge an gemeinsamen Ressourcen haben. Da dies sehr Domänen spezifisch ist wurde ein Ansatz gewählt bei dem das CDN selbst keine Annahmen über Mesh Zuordnungen macht. Dies ist Aufgabe der Anwendung die das CDN verwendet, denn nur sie kennt den Kontext in dem der Nutzer die Anwendung verwendet.

4.6.1 *Slidesync*

Slidesync ist eine Plattform deren Nutzung stark durch die durchgeführten live Events dominiert wird. Ein Moderator erstellt das Event lädt die notwendigen Assets, z.B. Foliensätze, hoch. Live Events werden für eine bestimmte Zeit festgesetzt und Teilnehmer laden zum Start des Events die Seite. Ein Großteil des entstandenen Traffics *Grafik visits*

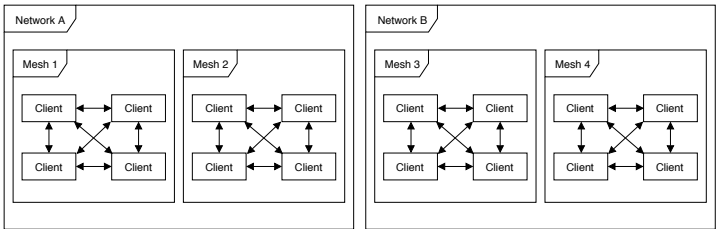

Figure 9: Peer to Peer Meshes - Slidesync

besteht aus HLS Videoseqmenten. Jeder Teilnehmer eines Events benötigt die selben Inhalte. *Grafik traffic*

Die Peer Meshes in Slidesync werden als voll vermaschte Netzte abgebildet. Da alle Teilnehmer eines Events zu großen Teilen die selben Daten benötigen können sie in dem selben Mesh untergebracht werden. Um zu gewährleisten das sich die Peers im Selben Subnetz befinden teilen sich nur solche ein Peer Mesh die sich in der Selben IP Range befinden. Ein weiterer wichtiger Factor ist der Kommunkationsmehraufwand der durch das halten von Verbindungen zu vielen Peers entsteht. Deshalb ist es nicht möglich bei größeren Events alle Peers im selben Peer Mesh unter zu bringen. Deshalb werden Sub Meshes gebildet in denen sich eine maximale Anzahl an Peers befinden können.

Abbildung 9 zeigt eine beispielhafte Aufteilung von Peer Meshes für ein Event. Für Netzwerk A und B werden jeweils zwei Meshes erzeugt und nur solche Clients werden miteinander verbunden die sich auch im Selben Subnetz befinden. Jedes Netzwerk wird wiederum in zwei Sub-Meshes unterteilt.

Um sicherzustellen das ein Peer sich auch aktiv am Mesh beteiligen kann sendet er regelmäßig ping Nachrichten an den Server. Da dieser Mechanismus in Slidesync schon zu vor zur Erhebung von Statistiken verwendet wurde, wird diese Nachricht lediglich um den Zustand des CDNs erweitert. Meldet ein Peer sich nicht innerhalb einer Minute oder meldet er das eine Verbindung zum Peer To Peer Netzwerk nicht möglich ist, so wird er als nicht mehr mit dem Peer To Peer CDN verbunden betrachtet. Sind alle aktuell verfügbaren Peer Meshes voll, so wird ein neues Peer Mesh angelegt und ein Hintergrund Job gestartet der alle Peer die als nicht verbunden betrachtet werden aus den Peer Meshes entfernt und die Meshes wieder als verfügbar markiert. Dadurch wird die Beantwortung der aktuelle Anfrage nicht verzögert und der Hintergrund Job nur bei Bedarf gestartet. Ebenso werden so Peers aussortiert deren Browser das Peer To Peer CDN nicht unterstützen, da der Anwendungsserver darüber zum Zeitpunkt der Zuordnung noch keine Kenntnis darüber hat.

- berücksichtigung des Netzwerkes
- IP range in vielen Fällen bekannt

4.6.2 *Schul-Cloud*

Bei der Schulcloud wird die Aufteilung der Nutzer anhand von Klassen gemacht. Alle Schüler die in der Selben Klasse sind werden dem selben Mesh zugeordnet. Schüler einer Klasse haben eine sehr große Überschneidung an Kursen die sie besuchen, und damit auch eine sehr große Übereinstimmung an Seiten die sie bei der Schul-Cloud aufrufen. Durch die überschaubare Klassengröße, in der Regel um die 30 Schüler, ist eine weitere Einteilung in Sub Meshes nicht nötig da diese Anzahl von einem Mesh gehandhabt werden kann. Eine mauelle Subnetzerkennung ist ebenfalls nicht notwendig da der Einsatz des Peer To Peer CDNs nur im Schulnetzwerk notwendig ist. Wird kein STUN Server spezifiziert so kann sich ein Schüler der von ausserhalb des Schulnetzwerk auf die Seite zugreift sich nicht mit Schülern innerhalb des Schulnetzwerks über WebRtc Verbinden und wird als möglicher Peer aussortiert.

4.6.3 *Updates*

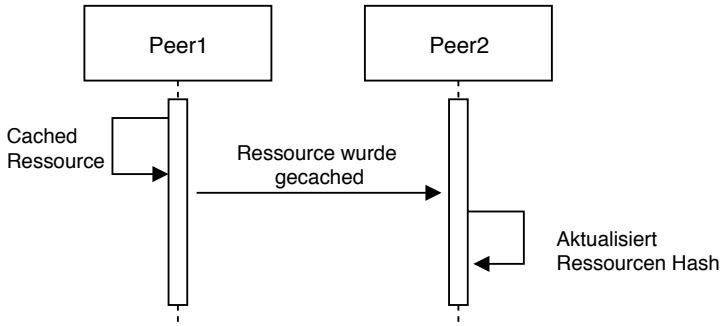

Figure 10: Flussdiagramm Ressourcen Update

Lädt ein Client eine neue Ressource in den Cache oder löscht er eine aus dem Cache so muss er seinem Peer Mesh dies mitteilen. Dazu sendet er eine Nachricht an alle Peers mit denen er verbunden ist. Die Kommunikation findet über den zuvor geöffneten Webrtc Datachannel statt. Empfängt ein Peer ein Update von einem anderen Peer so ändert er entsprechend die gespeicherte Hashmap von Ressourcen für diesen Peer.

4.6.4 *Subnetzerkennung*

Um Sicherzustellen das nur Peers aus dem gleichen lokalen Netzwerk miteinander verbunden werden muss eine Subnetzerkennung implementiert werden. Um dies zu erreichen gibt es im wesentlichen zwei Wege. Zum einen kann, wenn die IP-Range des Unternehmensstandorts/der Schule bekannt ist diese genutzt werden um nur jene Peers

in einem Mesh zu verbinden die sich im selben lokalen Netzwerk befinden. Dazu wird der Netzwerkanteil der IP Adresse mit der des Unternehmens/Schulnetzes verglichen. Stimmt der Netzwerkanteil überein so befinden sie sich im Schul- bzw. Unternehmensnetz.

Alternativ kann auch auf die Angabe eine NAT-Servers beim Webrtc Verbindungsaufbau verzichtet werden. Dadurch ist es Peers die sich hinter einem NAT oder einer Netzwerk Firewall befinden nicht mehr möglich sich mit Peers außerhalb des Netzwerkes zu verbinden, sehr wohl aber mit Peers innerhalb des selben lokalen Netzwerkes. Dies hat den Nachteil das Peers gemeinsam in Meshes sind die sich nicht miteinander verbinden können und im Anschluss aussortiert werden müssen. Jedoch muss im Vorfeld kein wissen über IP-Ranges vorhanden sein. Auch eine Konfiguration ist nicht notwendig.

Das Vergleichen von IP-Adressen hat den Vorteil das bereits zur Einteilung in die Peer meshes bekannt ist in welchem lokalen Netzwerk sich der Peer befindet. Dadurch können die Peers effizienter in die Meshes eingeteilt werden. Jedoch muss das Subnetz bekannt sein und in der Anwendung konfiguriert werden. Auch muss der Anwendung die IP-Adresse des Clients bekannt sein, was im Fall von Schul-Cloud aus Datenschutzgründen nicht möglich ist.

4.7 UMGANG MIT ÄLTEREN BROWSERN

Um sicherzustellen das das Nutzererlebnis für Teilnehmer mit nicht unterstützen Browser Versionen nicht negativ beeinträchtigt wird, überprüft das CDN bevor es initialisiert wird ob der Browser alle notwendigen Funktionalitäten unterstützt. Dazu wird der System Test verwendet. Wird der Browser nicht unterstützt so wird die Initialisierung des Scripts abgebrochen. Kann eine Verbindung zu einem Teilnehmer nicht aufgebaut werden oder bricht sie ab so wird er aus der eigenen Liste der Teilnehmer gelöscht, so das nicht versucht wird Anfragen über ihn zu beantworten. Kann das CDN nicht erfolgreich initialisiert werden, sei es weil der Browser nicht unterstützt wird oder weil die Netzwerk Einstellungen dies nicht zulassen, so werden Anfragen so beantwortet als wäre das CDN nicht vorhanden.

4.8 OFFLINE SUPPORT

- Motivation: Internet in Schulen ist nicht immer verfügbar
- lehrer lädt inhalte zuhause vor
- stellt sie schülern zur Verfügung
- wie funktionieren offline webanwendungen
- Service Worker

- prefetching
- Live Streams: Ausfallsicherheit wird erhöht.
- quasi bei design
- Resourcen werden gecached und sind offline verfügbar
- Tobias arbeit Refernzieren
- Übertragung im lokalen Netzwerk möglich wenn verbindung aufgebaut ist
- Signaling müsste in lokalem netzwerk sein z.b. Rechner vom Lehrer
- signaling könnte über webrtc erfolgen z.b. über
- kademlia routing indem Peers als vermittler fungieren
- evt. browser plugin
- Wie das routing machen?
- Offline scenario:
- Lehrer Lädt ressourcen vor und macht sie für schüler verfügbar
- Kein Internet vorhanden
- Schüler laden Ressourcen von Lehrer

4.9 SECURITY

- Owasp
- IP Leak möglich
- Stun Server kann nach IP Adresse fragen
- Über js auslesbar
- Pluckins können das blocken
- media.peerconnection.enabled ausschalten
- –> kein webrtc mehr möglich
-
- Datenintegrität integrity
- Integrität des kommunikationspartners confidentiality
- –vertrauensumgebung??

- availability
- durch hybrid cdn
- fallback lösung
- authenticity
- kann durch authority gelöst werden
- non repudiation
- accountability
- kann mit Statistiken getrackt werden
-
- Datachannel ist verschlüsselt
- https notwendig
- websocket ist verschlüsselt
- https://webrtc-security.github.io/

5

IMPLEMENTIERUNG

5.1 TECHNOLOGIEWAHL

Das Peer To Peer CDN ist eine Javascript Bibliothek, die nach erfolgreicher Einbindung und Konfiguration im Hintergrund eigenständig läuft. Es werden zwei Javascript Dateien zur Verfügung gestellt. Das Service Worker Script, und das Peer To Peer CDN Client Script. Beide Scripte müssen von der Anwendung eingebunden werden. Der Server muss einen Faye Websocket Server bereit stellen. Da das Peer Meshing über die Konfiguration des Faye Channels geschieht muss der Server gegebenefalls eine Peer Meshing Strategie implementieren. Zur einfacheren Einbindung in bestehende Projekte wird ein NodeJS Modul so wie ein Ruby Gem bereit gestellt. Das Ruby Gem bindet das ServiceWorker Script im Public Ordner der Ruby-on-Rails Anwendung ein und das Peer To Peer CDN Script im vendor/assets Ordner.

Der Javascript Code ist in ES6 geschrieben und wird mittels Babel[1] in Browser kompatiblen Javascript Code übersetzt und anschließend mit Node-minify komprimiert. [2]

5.2 ARCHITEKTUR

5.2.1 *Client Script*

Das Javascript Plugin besteht aus sieben Komponenten. Die Klasse P2pCDN bildet die Schnittstelle nach außen und stellt Funktionalitäten zum initialisieren und konfigurieren des CDNs bereit. Außerdem macht es den SystemTest nach außen verfügbar. P2pCDN nimmt die Konfiguration entgegen und initialisiert ein Peer Objekt. Das Peer Objekt repräsentiert den eigenen Clientn dem Peer To Peer Netzwerk. Es hält den eigenen Zustand und verwaltet die Verbindungen anderen Clientsnd verwaltet deren Ressourcen Hashes. Das Peer Objekt nutzt Funktionalitäten des Signaling Moduls um Verbindungen zu anderen Clientsufzunehmen. Das Signaling Modul implementiert das Webrtc Signaling Protokoll. FayeConnection abstrahiert die beim Signaling

1 https://babeljs.io/
2 https://www.npmjs.com/package/node-minify

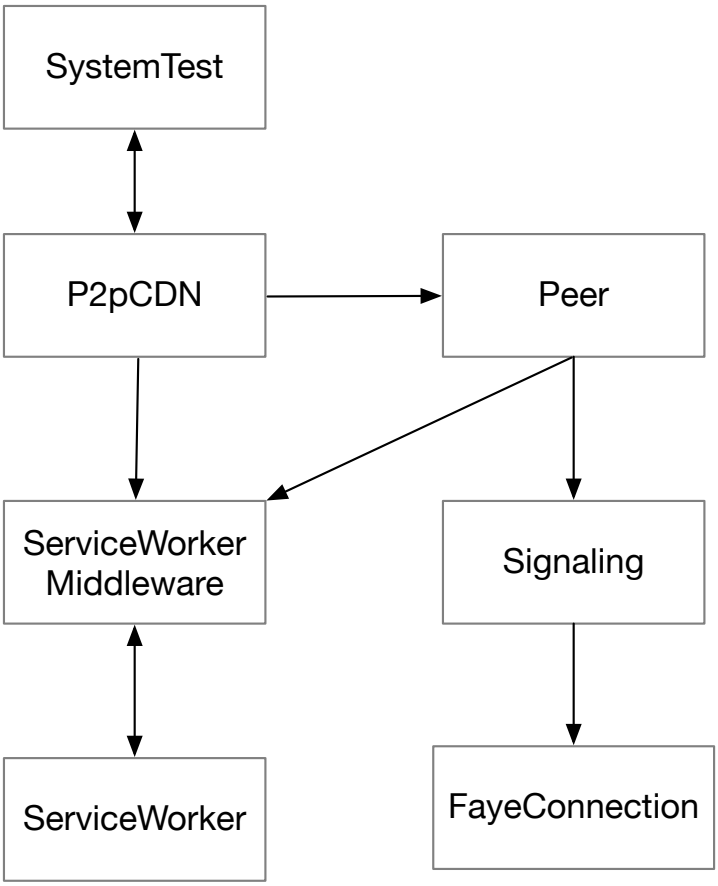

Figure 11: Klassendiagramm

verwendete Websocket Bibliothek, in diesem Fall Faye um es zu ermöglichen auch andere Websocket Bibliotheken zu verwenden. Die ServiceWorker Middleware fungiert als Vermittler zwischen Service Worker und Client Script und ist für die Initialisierung des Service Workers zuständig. Die Komponente arbeitet Event basiert und stellt folgende Events bereit:

PEER:ONREQUESTRESOURCE
: Wird von der Serviceworker Middleware ausgelöst wenn der Service Worker eine Ressource über das Peer To Peer CDN anfragt. Der Peer registriert auf dem Event und bearbeitet die Anfrage.

PEER:ONADDEDRESOURCE / PEER:ONREMOVEDRESOURCE
: Speichert der Service Worker eine Ressource so teilt er dies der ServiceWorker Middleware über einen Message Channel mit. Diese löst anschließend das peer:onAddedResource Event aus um dem Peer über die Änderung zu informieren, der die das Update an die verbundenen Clientseiter leitet.

SW:CLIENTREADY
: Noch eintscheiden ob darüber geschrieben werden soll

SW:ONREQUESTCACHE
: Stellt der Peer eine neue Verbindung zu einem anderen Clienter so sendet er eine Liste seiner gespeicherten Ressourcen an den Client. Diese Liste wird vom Service Worker verwaltet. Um an den Inhalt seines Caches zu kommen sendet der Peer das sw:onRequestCache Event mit einem Callback der bei erfolg ausgeführt wird. Die ServiceWorker Middleware leitet die anfrage über den Message Channel an den Service Worker weiter.

MESSAGE
: TODO

- heartbeats

5.2.2 *Service worker*

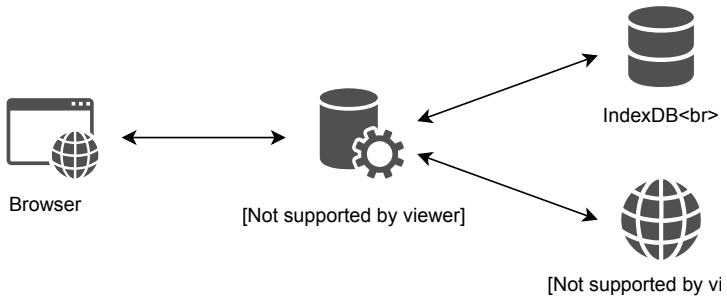

Figure 12: Service Worker

Als Proxy zwischen Client Anwendung und Browser wird ein Service Worker verwendet. Der Service Worker fängt von der Anwendung gesendete Anfragen im Fetch event ab und versucht sie als erstes durch den eigenen Cache zu beantworten. Ist die Anfrage nicht bereits im Cache vorhanden wird versucht sie über das Peer To Peer Netzwerk zu laden. Ist das nicht möglich so lädt er sie vom Server. Um eine Anfrage über das Peer To Peer Netzwerk zu beantworten muss zuvor das Client Script geladen sein, da Service Worker die Webrtc API nicht unterstützen. Die Kommunikation zwischen Service Worker und Anwendungs Script geschieht mit Hilfe der post Message API[3]. Um sicherzustellen das das Client Script geladen und bereit zur Kommunikation ist sendet der Service Worker eine nicht blockende Heartbeat Anfrage an das Script. Antwortet das Script nicht innerhalb eines bestimmten Zeitfensters so muss der Service Worker davon ausgehen, dass es noch nicht geladen ist. Die Anfrage wird an den Server weiter geleitet. Da nicht davon ausgegangen werden kann das das Script verfügbar bleibt nachdem es geladen wurde, die Seite könnte z.B. geschlossen worden sein, muss der Service Worker sich vor jeder Blockenden Anfrage vergewissern das das Script in der Lage

3 https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

ist zu antworten. Nachrichten die zwischen Anwendungs Script und Service Worker gesendet werden haben das Format: type: "Art der Nachricht", msg: "Inhalt der Nachricht" . Das Attribut "type" kann folgende werde enthalten: "resource", "cache", "hearbeat". Nachrichten vom Typ "resource" ordnen den Service Worker an eine Nachricht aus seinem Cache an das Anwendungs Script zu übergeben. Empfangt der Service Worker eine solche Nachricht lädt er die Angefragte Ressource aus dem Cache und fügt sie der Antwort bei. Nachrichten vom Typ "cache" dienen dazu beim Server den momentanen Inhalt des caches zu erfragen. Dies wird genutzt um einem neu Verbundenen Peer die Liste der aktuell gecachten Ressourcen mitzuteilen. Mit dem Typ "heartbeat" erfragt der Service Worker beim Anwendungs Script ob es momentan verfügbar ist. Ist das Script noch nicht verfügbar hat der Service Worker zwei Möglichkeiten damit umzugehen. Er kann darauf warten das das Script verfügbar ist und alle Anfragen bis dahin aufhalten oder er leitet die Anfragen direkt an den Server weiter. Leitet er die Anfragen direkt an den Server so wird die Ladezeit der Ressource nicht verzögert, jedoch ist es erst Möglich Ressourcen über das Peer To Peer CDN zu laden wenn das Script geladen wurde. Das CDN arbeitet weniger effektiv, dafür wird das Nutzererlebnis jedoch nicht negativ beeinflusst. Wartet der Service Worker bis das Script aktiv ist, so können mehr Anfragen über das CDN beantwortet werden, jedoch ist nicht garantiert wann es antwortet oder ob es überhaupt fehlerfrei geladen wird. Deshalb wurde sich bei der Implementierung des CDNs dafür entschieden das in diesem Fall Anfragen direkt an den Server weiter geleitet werden sollen. Das hat zur Folge das sich die gewählte Implementierung besonders für Single Page Applications sowie für Anwendungen die Navigationen z.B. mithilfe von Turbolinks vermeiden eignet. Bleibt beim Laden von neuen Inhalten das Anwendungsscript in Ausführung und muss nicht geladen werden, so kann der Service Worker Anfragen über das Peer To Peer Netzwerk beantworten. Vor der Registrierung des Service Workers speichert das Anwendungs Script die Konfiguration in die IndexDB des Browsers. Der Service Worker liest die Konfiguration von dort aus. Auf diesem Weg ist es möglich den Service Worker dynamisch zu konfigurieren und ihm unter anderem die zu verarbeitenden URLs zu übergeben. Die URLs werden als Regex übergeben. Fällt eine Anfrage nicht in die übergebenen URLs so unterbricht er die Verarbeitung und die Anfrage wird vom Server beantwortet. Sobald der Service Worker aktiv ist ist ruft er self.clients.claim() auf um Sicherzustellen das er schon beim ersten Seitenaufruf aktiv ist und anfragen der Clients verarbeiten kann.

- Codebeispiel für wartende messages
- Callbacks

5.2.3 *Tests*

Um das Plugin zu testen wird als test Runner Karma verwendet.[4] Als test framework wird mochajs[5] und als assertion Library wird Chai[6] eingesetzt

5.3 RESSOURCEN MANAGEMENT

5.4 CONFIGURATION

```
var config = {
  channel: 'FIXED_CLASS_1',
  clientId: '<%= peerId %>',
  idLength: 32,
  stunServer: {
    'iceServers': [
      // {
      //   'urls': 'stun:stun.l.google.com:19302',
      // },
    ]
  },
  verbose: true,
  serviceWorker: {
    urlsToShare:
      [
        '/img/',
        '/video/',
        '/testfiles/'
      ],
    path: '/p2pCDNsw.js',
    scope: '/',
    basePath: '/',
    storageQuota: '10000',
    cachingEnabled: false,
    verbose: true,
    statisticPath: '/logs'
  }
};
var cdn = new P2pCDN(config);
```

Listing 1: Beispielhafte Konfiguration

4 https://karma-runner.github.io/latest/index.html
5 https://mochajs.org/
6 https://www.chaijs.com/

Um eine gute Anpassung an verschiedene Anwendungsfälle zu ermöglichen bietet das Peer To Peer CDN eine Reihe von Konfigurationsmöglichkeiten. Nachdem das Client Script die Konfiguration geladen hat wird sie im der Indexed DB gespeichert und von dort durch den Service Worker geladen.1 zeigt eine Beispielhaft Konfiguration des CDNs im Folgenden werden die verschiedenen Konfigurationsmöglichkeiten aufgelistet und beschrieben.

CHANNEL
: Bezeichnet den für das Peer Meshing zu verwendenden Websocket channel. Alle Clients mit dem selben Channel befinden sich im selben Peer Mesh.

CLIENTID
: Eindeutiger Identifizier mit dem der Peer identifiziert werden kann. Ähnlich einer Session ID wird er verwendet um Clients wieder zuerkennen und anzusprechen.

IDLENGTH
: Bezeichnet die maximale Länge der ClientIds. Kürzere ClientIds werden bis zu dieser Länge aufgefüllt. Wird benötigt um intern bei dem verschicken von Paketen über das CDN ClientIds fester länge verwenden zu können.(siehe) *referenzieren*

STUNSERVER
: Gibt den zu verwendenden STUN Server an. Dies kann ein öffentlicher oder privat betriebener Server sein. Kann freigelassen werden falls kein STUN Server verwendet werden soll

VERBOSE
: Aktiviert/deaktiviert Debug Ausgaben.

SERVICEWORKER
: Beinhaltet alle Konfigurationen die den Service Worker betreffen.

URLSTOSHARE
: Liste aller URL die mit Hilfe des CDN bearbeitet werden sollen.

EXCLUDEDURLS
: Liste von URLs die explizit von dem CDN ausgeschlossen werden sollen

PATH
: und Text dahinter Pfad von dem das Service Worker Script geladen werden soll.

SCOPE
: Gibt den Scope an unter dem der Service Worker arbeiten soll.

BASEPATH
: Gibt den Service Worker Base Path an.

STORAGEQUOTA
: Maximal für das CDN zu verwendender Cache Speicher. Überschreitet der Cache den Wert, werden so lange Ressourcen aus dem Cache gelöscht bis der Wert unterschritten ist. (siehe) *ref*

CACHINGENABLED
: Aktiviert/Deaktiviert das Caching innerhalb des Service Workers. Nützlich zum Debuggen.

VERBOSE
: Gibt an ob der Service Worker debugging Ausgaben auf die Konsole schreiben soll.

STATISTICPATH
: URL an die die erhobenen Statistiken gesendet werden sollen. siehe 5.11

5.5 CLIENT UI EVENT

Um es der einbindenden Anwendung zu ermöglichen auf Änderungen bezüglich der verbundenen Peers sowie deren Ressourcen zu Reagieren stellt das Plugin das Event ui:onUpdate bereit das bei Änderungen ausgelöst wird. Das Event übergibt das Peer Object des Clients wodurch der Event Empfänger zugriff auf die Anzahl der verbundenen Peers so wie deren Ressourcen hat.

5.6 SIGNALING SERVER

Der Signaling Server nutzt Faye[7] als Websocket Bibliothek. Die Anwendung muss lediglich einen Faye Server anbieten um das Signaling zu ermöglichen. Der Verbindungsaufbau wird im Anschluss Client seitig gehandhabt. Um Clients einem Peer Mesh zuzuordnen werden verschiedene Websocket Channels verwendet. Wird zwei Clients der selbe Datachannel in der Konfiguration übergeben so befinden sie sich im selben Peer Mesh. So ist eine einfache Konfiguration möglich. Das CDN selbst macht keine Annahmen darüber welche Clients sich sinnvollerweise im selben Mesh befinden sollten. Die Entscheidung darüber ist der Anwendungslogik überlassen. Dadurch ist es möglich ein Domänen spezifisches Peer Meshing vorzunehmen.

- websockets faye
- gleicher data channel zur Bildung von meshs
- ID pro client
- max id length muss festgelegt werden für chunking

7 https://faye.jcoglan.com/ruby/websockets.html

- Protokoll von P2PCDN umgesetzt
- Protokoll erklären
- Diagram Protokoll
- Peer Meshing wird von anwendung übernommen

5.6.1 *Slidesync*

Um das Signaling zu realisieren implementiert Slidesync ein eigen Klasse das für die Zuordnung von Clientsu Peer Meshes verantwortlich ist. Der Klasse wird der Scope des Aufrufs so wie die maximale Mesh Größe bei der Initialisierung übergeben. Wobei der Scope die ID des aufgerufenen Events ist. Durch Aufruf der join() Methode wird der Peer einem geeigneten Mesh zugeordnet und der Faye Channel des Meshes zurückgegeben. Dabei wird immer das erste freie Mesh gewählt um möglichst viele volle Meshes zu erreichen. Die Zuordnung muss performant geschehen und in einem scalierten Multi Server Setup funktionieren. Um dies zu erreichen wird Redis als Datenspeicher verwendet. Redis bietet geignete Datenstrukturen mit geringen Zugriffszeiten und ist gut für ein multi Server Setup geeignet. Die Klasse speichert die liste der verfügbaren Submeshes in einer Liste. Jedes Mesh wird über eine Aufsteigende ID in Kombination mit dem Scope identifiziert. Mit Hilfe eines Redis Counters wird die ID des letzten Submeshes gespeichert. Dadurch lassen sich alle angelegten Meshes ermitteln ohne sie explizit speichern zu müssen. Erreicht ein Mesh die Maximale Teilnehmerzahl so wird es aus der Liste der verfügbaren Submeshes entfernt. Ist kein freies Mesh verfügbar so wird ein neues Mesh eröffnet. Dazu wird der Counter der letzten Mesh ID erhöht und ein neues Mesh mit dieser ID wird der Liste der verfügbaren Meshes hinzugefügt. Im Anschluss wird ein Hintergrund Job gestartet der überprüft ob bereits angelegte Meshes wieder frei geworden sind. Dazu wird für jedes Mesh überprüft ob sich alle Teilnehmer in der letzten Minute zurückgemeldet haben. Haben sich Teilnehmer nicht zurück gemeldet, so wird das Mesh wieder zu den verfügbaren meshes hinzugefügt. Da über sämtliche Submeshes und alle Peers iteriert werden muss macht es Sinn die Operation im Hintergrund durchzuführen. Die Zuordnung von Teilnehmern zu Meshes wird für jedes Mesh ein Redis Set angelegt in dem die ID jedes Peers der dem Mesh beigetreten ist gespeichert wird. Zusätzlich wird für jeden Peer ein temporärer Key angelegt der nach 40 Sekunden automatisch gelöscht wird. Um Statistiken über Events zu erheben meldet sich jeder Teilnehmer eines Events alle 30 Sekunden bei dem Server und übermittelt Daten von seinem Client. Im Rahmen dieser Statistischen Erhebung wird ebenfalls der temporäre key erneuert, falls er sich am Peer To Peer Netzwerk beteiligen kann. Ist der temporäre Key für einen Peer nicht verfügbar so wird angenommen das er nicht mehr

am Peer To Peer Netzwerk teilnimmt und sich demnach auch nicht mehr in dem Peer mesh befindet.

5.6.2 *Schul-Cloud*

5.7 MESSAGE PROTOCOL

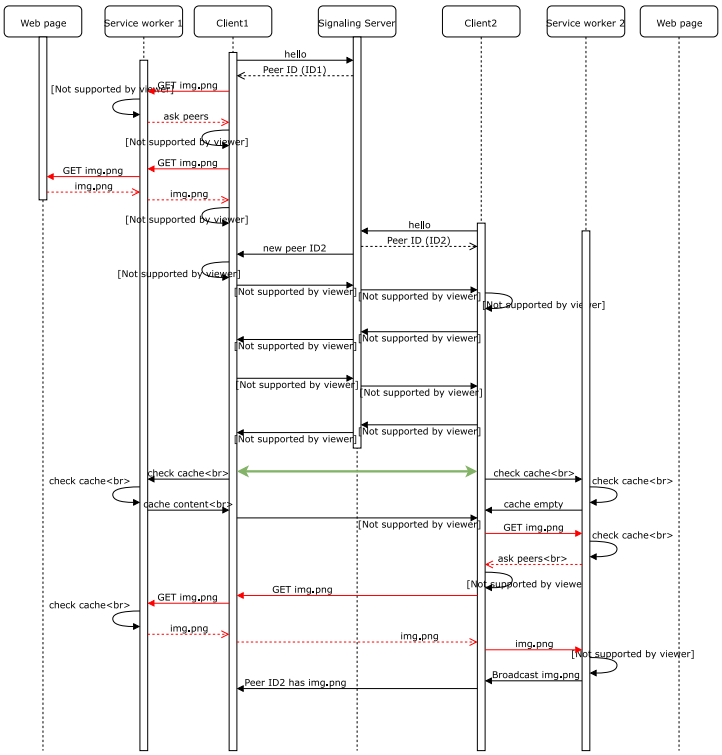

Figure 13: A Figure Title

Diagramm aktualisieren

Client 1 (C1) ist der erste der die Webseite aufruft. Er registriert sich beim Signaling server und fragt im Anschluss img.png an (rot). Da noch niemand anders auf der Seite ist von dem er die Ressource bekommen könnte und er zudem die Ressource nicht in seinem Cache hat, wird img.png über das Internet vom Webserver geladen. Client 2 (C2) ruft nun ebenfalls die Webseite auf und registriert sich beim Signaling server. Dieser benachrichtigt C1, dass ein neuer Teilnehmer registriert wurde, woraufhin C1 einen Verbindungsaufbau zu C2 einleitet. Steht die direkte Verbindung zwischen C1 und C2 (grün), teilt C1 C2 den Inhalt seines aktuellen Caches mit. Fragt C2 img.png an (rot), weiß er so, dass er diese von C1 anfragen kann. Hat er img.png erhalten, teilt er allen anderen Teilnehmern (in diesem Fall nur C1) mit, dass auch er jetzt img.png als Ressource in seinem Cache hat

5.7.1 *Updates*

- message types
- neue Resource
- gelöschte Resource
- Client tritt dem Netzwerk bei
- Client verlässt Netzwerk
- Diagram aktualisieren
- alle Clients müssen benachrichtigt werden
- Über Webrtc data channel
- Message format zeigen

5.7.2 *Client fragt Ressource an*

- Message Format
- Diagram beschreiben

5.8 REUSABILITY

5.9 SERIALISIERUNG DER DATEN

Für den Austausch von Daten werden die von WebRTC angebotene DataChannel genutzt, welche das Stream Control Transmission Protocol kurz SCTP verwendet. Problem hierbei ist, dass dieses Protokoll ursprünglich für die Übertragung von Kontrollinformationen designt wurde und deshalb für die Kompatibilität verschiedener Browser eine Paketgröße von 16kiB nicht überschritten werden sollte. Es ist jedoch notwendig auch größere Dateien zu übertragen, weshalb aktuell viele kleine Datenpakete verwendet werden müssen. Hierdurch entsteht ein nicht zu vernachlässigender Overhead.

Datachannel haben in der aktuellen Browser Implementation eine maximale Buffergröße. Der client der die Datachannel verwendet ist dafür verantwortlich sicher zu stellen, dass die maximale Buffergröße nicht überschritten ist. Ist die maximale Buffergröße erreicht, so werden weitere Chunks erst gesendet, wenn sich der Buffer wieder geleert hat.

Empfängt ein Client eine Anfrage in Form von Chunks so muss er diese wieder zu der eigentlichen Ressource zusammen setzen. Dazu wird für die Anfrage ein Array verwaltet in denen alle empfangenen

```
// maximum buffer size is 16mb
if(peer.dataChannel.bufferedAmount <= 16000000) {
  peer.dataChannel.send(msg);
  return;
}
// if maximum buffersize is reached delay sending of chunks
peer.requestQueue.push(msg);
peer.dataChannel.bufferedAmountLowThreshold = 65536;
peer.dataChannel.onbufferedamountlow = function () {
  var reqs = peer.requestQueue.slice();
  peer.requestQueue = [];
  reqs.forEach(_msg => send(_msg));
}
```

Listing 2: Buffersize Berücksichtigung

Chunks gespeichert werden. Wurden alle Chunks einer Anfrage empfangen werden sie wieder zu einem Objekt zusammengesetzt und die Anfrage wird an den Service Worker weiter geleitet

- evtl serialisierung thematisieren.

5.10 SYSTEM TEST

Das Plugin stellt ein Modul bereit um zu testen ob es einem Client möglich ist am Peer To Peer CDN teilzunehmen.

Dazu werden drei Tests bereitgestellt.

p2pCDN.systemTest.testBrowser()

Mit Hilfe von modernizr[8] testet die Funktion ob die verwendete Browserversion alle benötigten Funktionen unterstützt. Modernizr ist eine Javascript Bibliothek mit der getestet werden kann ob ein Browser bestimmte Funktionen unterstützt.

p2pCDN.systemTest.webrtcInitialized()

Gibt ein Promise zurück welches überprüft ob die webrtc Verbindung erfolgreich aufgebaut wurde. Da die Initialisierung einen Moment in Anspruch nehmen kann wird wiederholt geprüft ob die Verbindung aufgebaut wurde.

p2pCDN.systemTest.clientConnected()

Gibt ebenfalls ein Promise zurück und überprüft ob erfolgreich eine Verbindung zu einem anderen Client aufgebaut werden konnte. Um diesen Test erfolgreich auszuführen muss sich ein andere Client im aktuellen Peer mesh befinden.

8 https://modernizr.com/

```
_handleChunk(message) {
  const req = this._getRequest(message.from, message.hash);
  var response = {}
  req.chunks.push({id: message.chunkId, data:
  ↪  message.data});

  if(req.chunks.length === message.chunkCount) {
    response.data = this._concatMessage(req.chunks)
    response.from = message.from;
    response.peerId = this.peerId;
    this._removeRequest(message.from, message.hash);
    req.respond(response);
  }
}
```

Listing 3: Buffersize Berücksichtigung

5.11 ERFASSEN VON STATISTIKEN

Um die Nutzungsstatistiken des CDNs zu erfassen sendet jeder client periodisch POST requests an den Server. Dazu sammelt der Service Worker alle Anfragen die in einem Zeitraum von 10 Sekunden angefallen sind und sendet sie gebündelt als JSON an den Server. Erfasst werden:

PEERID
: Die PeerId bezeichnet die Id des peers der die Statistik sendet.

METHOD
: Method gibt an wie die Anfrage behandelt wurde und kann die Werte 'cacheResponse', 'peerResponse' oder 'serverResponse' beinhalten. Ein cacheResponse konnte aus dem eigenen Cache beantwortet werden. ServerResponse bedeutet das die Anfrage über den externen Server geladen werden musste. Der Wert peerResponse gibt an das die Anfrage über das Peer To Peer CDN bearbeitet werden konnte.

FROM
: 'From' gibt an woher die Anfrage geladen wurde. Im Falle eines serverResponses beinhaltet sie den Wert 'server' und bei einem cacheResponse den Wert cache. Wurde die Anfrage über das Peer To Peer Netzwerk beantwortet beinhaltet sie die peerId des Peers der die Anfrage beantwortet hat. Dazu sendet das Script neben der eigentlichen Anfrage auch die eigene PeerId und die PeerId des peers der die Anfrage beantwortet hat an die Service Worker. (siehe 7)

```
function sendStatisticToServer() {
  if(!serverSendTimeout && config.statisticPath){
    serverSendTimeout = setTimeout(function(){
      try {
        fetch(config.statisticPath, {
          method: 'POST',
          body: JSON.stringify(requests),
          headers:{
            'Content-Type': 'application/json'
          }
        });
      } catch(e) {

      } finally {
        serverSendTimeout = 0;
        requests = [];
      }
    }, sendStatisticDelay)
  }
}
```

Listing 4: Erfassen der Statistiken

URL
: Die URL enthält die URL der angefragten Ressource.

TIMING
: Timing beinhaltet die Zeitspanne die benötigt wurde um die Anfrage zu beantworten, beginnend vor der Entscheidung wie der Request abgearbeitet werden soll(siehe 6) und endend nach dem die Anfrage empfangen wurde. Nicht enthalten in der Zeitspanne ist die Entscheidung ob der Service worker den Request bearbeitet und die Renderzeiten des Browsers. Diese Zeiten sind nicht abhängig von der Art der Request Beantwortung.(siehe Evaluation)

Mit Hilfe der Configuration kann festgelegt werden an welchen Endpunkt die Statistik gesendet werden soll. Die Anwendung ist für das speichern und verarbeiten der Daten zuständig, dies ist nicht Teil des Plugins. Slidesync speichert die Daten als JSON in Redis und stellt einen JSON Endpunkt zur Verfügung mit dem die Statistiken abgerufen werden können. Für die Labortests werden die Daten in JSON Dateien zur späteren Verarbeitung abgelegt.

- evtl. Mongo db
- anzeigetool

```
function logStatistic(url, method, request, timing, from,
↪  peerId) {
  if(!config.statisticPath) return;
  var p_Id = peerId ? peerId : config.clientId;
  var data = {
    'peerId': p_Id,
    'method': method,
    'from': from,
    'url': url,
    'loadTime': timing
  };
  requests.push(data);
  sendStatisticToServer();
}
```

Listing 5: Erfassen der Statistiken

- schoolcloud

5.12 QUOTA LIMITS - LÖSCHEN VON REQUESTS AUS DEM CACHE

[9]

Table 2: Browser Storage Quotas

Browser	Limit
Chrome	< 6% des freien Fesplattenspeichers
Firefox	< 10% des freien Fesplattenspeichers
Safari	< 50MB
IE10	< 250MB
Edge	Abhännig von der Festplattengröße

Browser stellen den Clients unterschiedlich viel Speicherplatz für offline Caches zur Verfügung. Ist das Quota limit erreicht versuchen Firefox und Chrome Speicher frei zu machen indem Elemente aus dem Cache gelöscht werden. Dabei werden jedoch keine einzelnen Elemente aus dem Cache gelöscht sondern mittels Last-recently-used(LRU) werden ganze Caches gelöscht. Safari und Edge haben keinen Mechanismus zum automatischen Löschen von Elementen sondern werfen

9 https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa

lediglich einen Fehler.[10] Deshalb ist es notwendig das der Service Worker in dem Fall das das Limit erreicht wird Elemente löscht.

Mit Hilfe der Quota Management API[11] ist es möglich die momentane Speichernutzung auszulesen ebenso wie den maximal Verfügbaren Speicherplatz. Ist diese Limit oder das Quota Limit welches über die Konfiguration angegeben wurde erreicht, löscht der Service Worker so lange die ältesten Einträge im Cache, bis genügend Speicher für den nächsten Request vorhanden ist. Dazu berechnet der Service Worker die Größe des zu speichernden Request.

5.13 RESOURCE LOADING

5.14 CONFIGURATION VON UNTERNEHMENSNETZWERKEN

- Port range
- firewalls?w
- client firewalls

10 https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa

11 https://developers.google.com/web/updates/2011/11/Quota-Management-API-Fast-Facts

```
function handleRequest(url, clientId) {
  return new Promise((resolve) => {
    var startTime = performance.now();

    sha256(url).then(hash => {

      // check cache
      getFromCache(hash).then(cacheResponse => {
        if (cacheResponse && config.cachingEnabled) {
          log('cacheResponse ' + cacheResponse.url);

          // This notify should not be needed
          notifyPeersAboutAdd(hash, clientId);
          var endTime = performance.now();
          logStatistic(url, 'cacheResponse', cacheResponse,
          ↪  endTime-startTime, 'cache');
          resolve(cacheResponse);
          return;
        }
        // check peers
        getFromClient(clientId, hash).then(data => {
          if (data && data.response) {
            var peerResponse = data.response;
            log('peerResponse ' + peerResponse.url);
            putIntoCache(hash, peerResponse, clientId);
            notifyPeersAboutAdd(hash, clientId);
            var endTime = performance.now();
            logStatistic(
              url,
              'peerResponse',
              peerResponse,
              endTime-startTime,
              data.from,
              data.peerId
            );
            resolve(peerResponse);
            return;
          }
          // get from the internet
          getFromInternet(url).then(response => {
            log('serverResponse ' + response.url);
            putIntoCache(hash, response, clientId);
            notifyPeersAboutAdd(hash, clientId);
            var endTime = performance.now();
            logStatistic(url, 'serverResponse', response,
            ↪  endTime-startTime, 'server');
            resolve(response);
          });
        });
      });
    });
  });
}
```

```
_handleChunk(message) {
  const req = this._getRequest(message.from, message.hash);
  var response = {};
  req.chunks.push({id: message.chunkId, data:
  ↪  message.data});

  if(req.chunks.length === message.chunkCount) {
    response.data = this._concatMessage(req.chunks);
    response.from = message.from;
    response.peerId = this.peerId;
    this._removeRequest(message.from, message.hash);
    req.respond(response);
  }
}
```

Listing 7

6

EVALUATION

6.1 PREQUISITES

6.2 BROWSER COMPATBILITY

Chrome	Firefox	Edge	Safari	Internet Explorer
75	68	76	12.1	Not supported

IOS Safari	Samsung Internet	Android Browser	Chrome (Android)
12.3	9.2	67	75

Table 3: Unterstützte Browser Versionen

Um die unterstützten Browser zu verifizieren wurden die vom Peer To Peer CDN verwendeten Browser Funktionalitäten bei caniuse.com[1] eingegeben. Darüber hinaus wurden Chrome, Firefox, Edge, Safari und der Internet Explorer manuell getestet, um sicherzustellen das das Peer To Peer CDN das Nutzererlebnis nicht negativ beeinflusst. Der Internet Explorer unterstütz auch in der aktuellen Version keine Service Worker[2]. Auch wenn der Edge Browser Webrtc seit Version 17 unterstützt, ist es leider erst ab version 76 möglich Datachannels zu verwenden. Da der Datenverkehr des Peer To Peer CDNs über Datachannel gehandhabt wird, ist eine Verwendung erst ab der nächsten Version die auf Chrome aufbauen wird, möglich.

6.2.1 *Browser Usage in coropate networks*

- Statistic über typische nutzung
- Statistic über Nutzung bei dem test
- Diskussion Edge on Chromium
- evtl aussage von deutscher bank mit aufnehmen
- — major update hinauszögern, in 1-2 Jahren
-

6.2.2 *Browser usage in eduational networks*

6.3 WEBRTC ROUTING

- mit und ohne STUN
- lokales routing
- externes Routing
- implikationen für Konfiguration

6.4 BANDWIDTH

6.4.1 *Latency*

6.4.1.1 *Webrtc Verbindungsaufbau*

- Page load bis cdn is ready
- verschiedene Anzahl an Peers im Netzwerk

6.5 SIMULIERTER WORKLOAD

6.5.1 *Chunking*

- messen des aufwandes für chunking und wieder zusammensetzen
- woher kommt der zusätzliche Aufwand bei größen dateien??
- limit ist der durchsatz nicht das chunking

6.5.2 *Durchsatz - Ladezeiten von verschiedenen Dateigrößen*

Um die Ladezeiten verschiedener Dateigrößen zu evaluieren wurde auf einem Mac book pro Ende 2013 mit 2.3 GHz und 16GB Arbeitsspeicher getestet. Getestet wurde mit zwei Teilnehmern. Einer lud die Ressource vor und der zweite lud sie über das Peer To Peer CDN. Die Timeouts des Peer To Peer CDNs wurden für diesen Test deaktiviert. Um Netzlaufzeiten auszuklammern befand sich der Anwendungsserver ebenso wie die zu ladene Ressource auf dem selben Rechner wie die Clients.

- Mac book pro Ende 2013
- 2.3 ghz i7
- 16gb ram

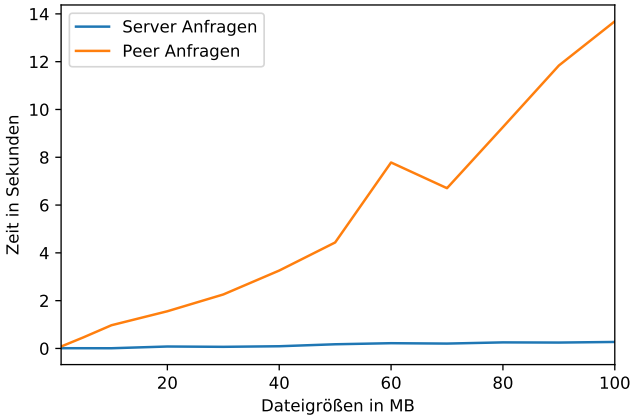

Figure 14

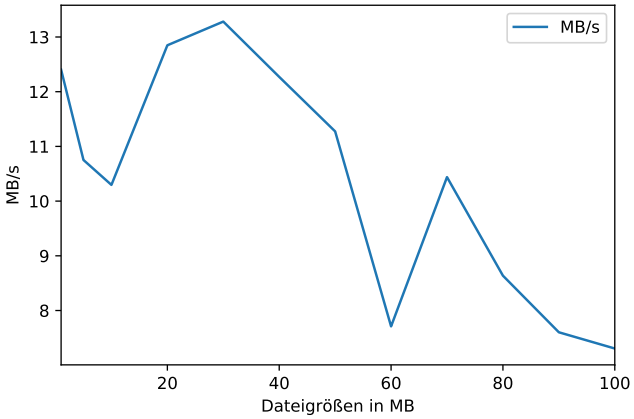

Figure 15: Durchsatz

- 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 , 100 mb
- durchsatz berechnen
- Limitierung durch websockets
- lokal auf einem Rechner um Netzlaufzeiten auszuklammern
- 1 Peer lädt vor
- der andere lädt von peer
- benchmarkt das nur das lokale netzwerk?? –> routing
- sicherstellen das datenverkehr nicht über internet geroutet werden...

6.5.3 *Live Streaming*

Um einen Live Stream mit höherer Teilnehmerzahl zu testen wurde ein Script geschrieben, das mit Hilfe von puppeteer[3] mehrere Chrome browser um headless Mode startet und die Event Seite besucht. Dabei musste eine Chrome installation gewählt werden, da Chromium nicht über die notwendigen Codecs verfügt um HLS Video wieder zu geben. Das Script wurde auf Amazon AWS t2.xlarge Instanzen installiert. Jede dieser Instanzen verfügt über 8 vcpus und 32 GB Arbeitsspeicher. Auf jeder Instanz wurden 25 Browser Sessions geöffnet. Das CDN wurde so konfiguriert das es ausschließlich .ts Dateien, also die Video Segmente bearbeitet. Bei 25 Teilnehmern lag die CPU Last bei den Testservern bei deaktiviertem Peer To Peer CDN bei 50-60%. Die Teilnehmer luden die Event Seite während das Event noch nicht Live geschaltet war. Wenn alle Teilnehmer die Seite geladen hatten wurde das Event live geschaltet und die Teilnehmer auf die Live stream Seite weiter geleitet. Die Weiterleitung geschieht verzögert in einem Zeitraum von 18-60 Sekunden mit einer Dreicks-verteilung um die Last auf Seiten der Slidesync Servern zu verringern Der Stream wurde für zehn Minuten live geschaltet und anschließend beendet.

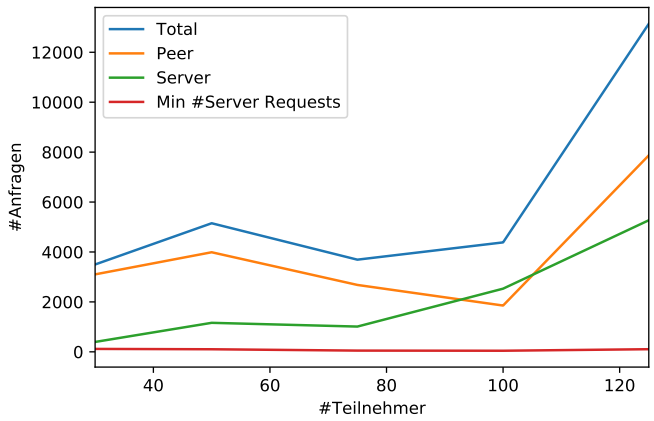

Figure 16: Vergleich der Bearbeitung nach Art in einem Mesh

Datenpunkte mehr ersichtlich machen

Abbildung 16 zeigt die wie sich das Peer To Peer CDN bei steigender Teilnehmerzahl mit einem Mesh verhält. Bis 50 Teilnehmern hat das CDN ein lineares Verhalten hinsichtlich bearbeiteter Anfragen. Ab 75 Teilnehmern ist zu sehen das die Gesamtanzahl der verarbeiten Anfragen einbricht. Die CPU Last der Test Server stieg auf 100% und die Clients waren dadurch nicht mehr in der Lage an dem CDN Teilzunehmen und Anfragen zu bearbeiten. Der Kommunikationsaufwand die Peers in dem Mesh über Veränderungen des Caches zu Benachrichtigen wurde zu groß. Zwar wurden bei 125 Teilnehmern

3 TODO: puppeteer

wieder mehr Anfragen vom CDN verarbeitet, jedoch stieg der Anteil der Anfragen die über den Server liefen weiter an. Die höhere Anzahl an Anfragen die vom CDN bearbeitet wurden ist vor allem darauf zurück zu führen, dass die Verbindungen zwischen den Peer durch die hohe Last teilweise unterbrochen wurde, wodurch sich die Last bei den Teilnehmern verringerte.

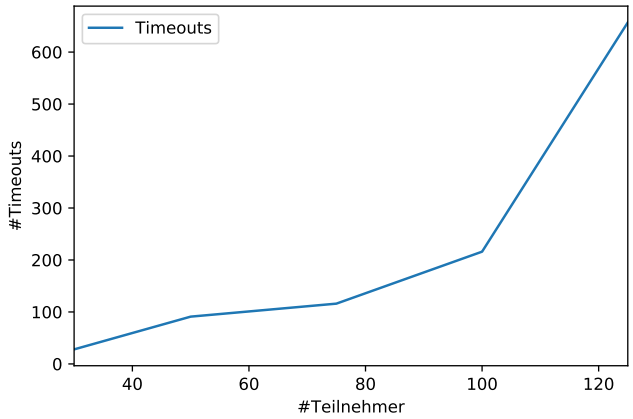

Figure 17: Timeouts bei einem Mesh

Betrachtet man die Anzahl an Timeouts, so bestätigen sich die vorherigen Beobachtungen. Ab 75 Teilnehmern steigt die Anzahl an Timeouts stark an. Die CPU Last auf Seiten der Teilnehmern wurde zu groß um alle Anfragen in der geforderten Zeit zu beantworten. Neben der zu hohen Last ist kann es zu Timouts kommen falls zu viele Teilnehmer gleichzeitig eine Anfrage an den Selben Teilnehmer stellen. Der betroffene Teilnehmer ist in diesem Fall nicht in der Lage alle Anfragen schnell genug zu bearbeiten. Der Timeout wurde mit drei Sekunden so gewählt das es trotz Timeouts nicht zu einer Unterbrechung des Streams kommt. Der Video Player von Slidesync lädt drei HLS vor die je sechs Sekunden Video beinhalten insgesamt werden also 18 Sekunden vor geladen. Zu einer Unterbrechung der Wiedergabe kann es also erst kommen wenn eine Anfrage länger als 6 Sekunden benötigt. Da die längste verarbeitete Anfrage XXX Sekunden benötigte kam es zu keiner Unterbrechung der Wiedergabe aufgrund von Timeouts.

Abbildung 18 zeigt die Verarbeitungsart der Anfragen bei steigender Mesh Größe. Alle Tests wurden mit einer Teilnehmerzahl von 125 durchgeführt. Bei einer Mesh Größe von zehn ist die mindestanzahl an Anfragen die von dem Server beantwortet werden müssen mit 10% relativ groß. Auch wenn nur selten Timeouts durch zu viele Anfragen bei dem selben Teilnehmer vorkommen konnten nur 84% der Anfragen über das Peer To Peer CDN beantwortet werden. Die Beste Abdeckung hatte das CDN bei einer Mesh Größe von 30 Teilnehmern mit 92%. Bei einer Mesh Größe sind müssen mindestens 4% der Anfrage über den

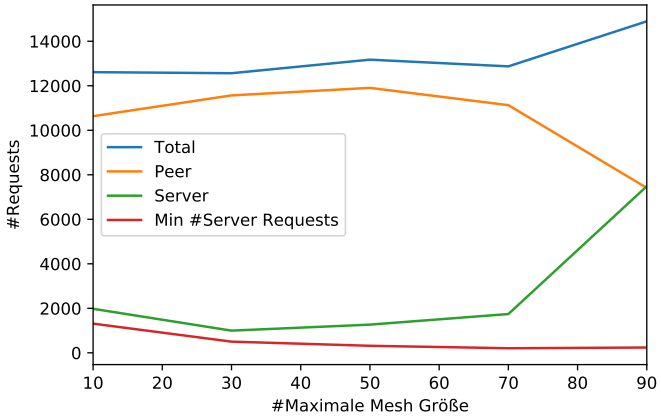

Figure 18: Vergleich verschiedener Mesh Größen

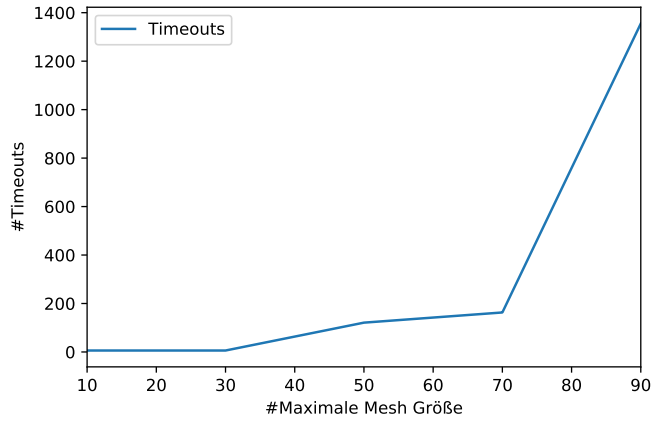

Figure 19: Timeouts bei verschiedenen Mesh Größen (125 Teilnehmer)

Server beantwortet werden, damit jedes HLS Segment in jedem Peer Mesh vorhanden ist. Wird eine größere Mesh Größe als 30 gewählt so steigt auch die Anzahl der Timeouts ebenso wie die CPU Auslastung bei den Teilnehmern. So konnte bei einer Mesh Größe von 90 lediglich die Hälfte aller Anfragen durch das Peer To Peer CDN beantwortet werden.

Um zu testen wie sich das CDN bei größeren Teilnehmerzahlen verhält wurden Tests mit einer Mesh Größe von 30 durchgeführt. Abbildung 20 zeigt einen Vergleich der Verarbeitungsarten. Die Anzahl an Peer Anfragen Verhält sich linear mit steigender Teilnehmerzahl. Die Abdeckung durch das CDN schwankt zwischen 70-76%.

Abbildung 21 zeigt eine zeitliche Darstellung von beginn des Streams bis zum Ende des Streams für 30 Teilnehmer in einem Mesh. Es ist gut zu sehen, dass zu Beginn des Tests mehr Anfragen über die Server beantwortet werden müssen. Auch die Gesamtanzahl der Anfragen ist zu beginn des Tests höher als im späteren Verlauf. Nachdem die Seite

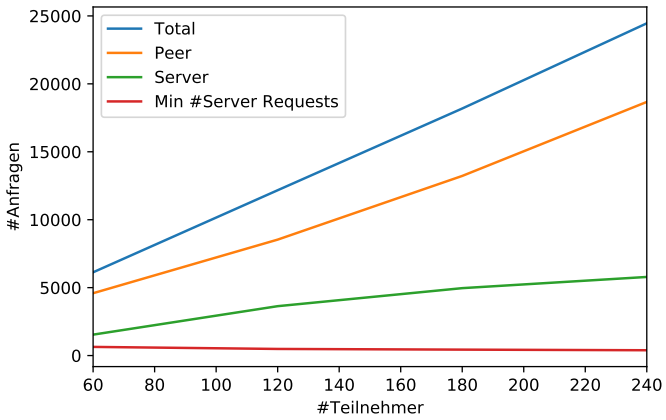

Figure 20: Mesh Größe von 30 bei steigender Teilnehmer Zahl

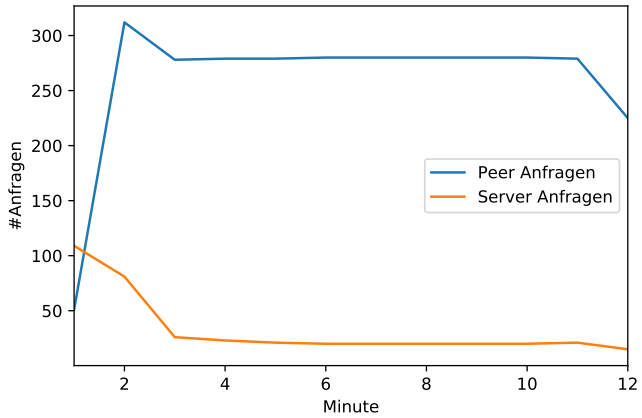

Figure 21: Anfrageart über die Zeit bei 30 Teilnehmern

geladen ist laden alle Teilnehmer drei HLS Segemente vor. Diese HLS Segmente werden gleichzeitig geladen, wodurch sich die Wahrscheinlichkeit erhöht das mehere Teilnehmer annähernd gleichzeitig die selbe Ressource anfragen und kein anderer Teilnehmer sie bereits geladen hat.

Rahmenbedingungen müssen klar sein

- Verhältnisse berechnen
- evtl ramp up anschauen
-
- 92872 pro segment (ca 92 kb)
- ca 9,2 mb pro client pro event
- bandbreiten diagram?
- Wenn client von server lädt teilt er das direkt mit

6.5.4 *Schul-Cloud*

- Testsetup
- 30 Schüler
- 3 Clicks
- Dashboard
- kurs liste
- kurs
- Thema
- hauptsächlich css etc
- 1 Server macht anfragen
- verschieden zufällig gewählte startzeiten in verschiedenen Zeiträumen
- wo turbolinks erwähnen?

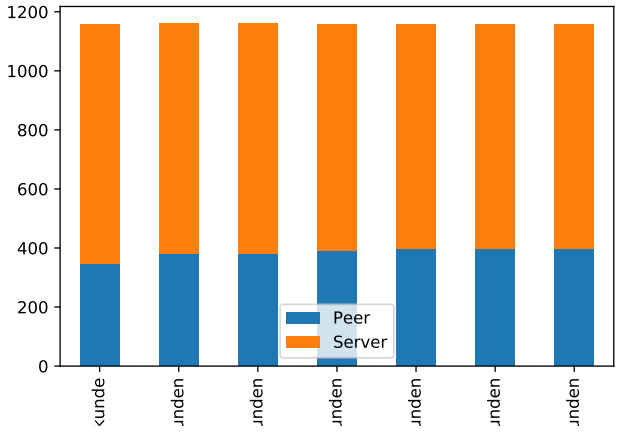

Figure 22: Vergleich der Anfragen nach Anfrageart

- Kein Großer unterschied zwischen den Zeiträumen

Beschriftungen

- First load vs navigationen
- First load schlechte abdeckung -> erst möglich wenn cdn initialisiert ist
- navigationen sehr hohe abdeckung
- swa/turbolinks notwendig

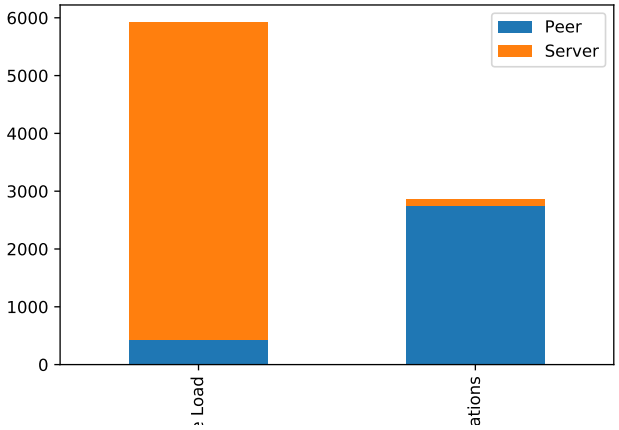

Figure 23: Vergleich der Anfragen nach Anfrageart

- weiteres szenario
- 1 User hat geladen(Lehrer)
- rest kommt
- inhalt mit pdf

6.6 LIVESTREAMING IN UNTERNEHMENSNETZWERKEN

Um die Funktionalität des CDNs im Falle eines Livestreams zu testen wurde ein Livestream mit der Infrastruktur von Slidesync aufgesetzt. Slidesync verwendet bei der Verteilung des Videos das Datenformat HLS. Das CDN wurde so konfiguriert das ausschließlich die HLS Segmente über das CDN behandelt werden. Sämtliche im folgenden betrachteten Anfragen sind demnach HLS Segmente. Der Test wurde in einem Unternehmensnetzwerk durchgeführt um die Funktionalität in einem solchen Netzwerk zu gewährleisten. In diesem Netzwerk wurden insgesamt 30 Rechner aufgestellt. Da es vor allem um den Durchsatz des CDNs ging wurde auf allen Clients ein aktueller Chrome Browser verwendet. Auf den Clients wurde zuerst die Event Seite geladen und erst im Anschluss der Livestream gestartet. Getestet wurden sechs, zehn, 15, 20 und 30 Clients, die sich alle im selben lokalen Netzwerk befanden. Der Streaming- und der Anwendungsserver befand sich außerhalb des Netzwerkes und waren nur über die Internetverbindung zu erreichen. Bei jedem Test wurde der Stream zehn Minuten laufen gelassen und alle Clients befanden sich in dem selben Peer Mesh.

Abbildung 24 zeigt die Abhandlungsart der Anfragen. Dabei ist zu beachten das ca 100 Requests notwendig waren damit sämtliche HLS Segmente im Peer To Peer Netzwerk verfügbar sind. Bei steigender Anzahl von Teilnehmern ist zu beobachten das eine größere Anzahl

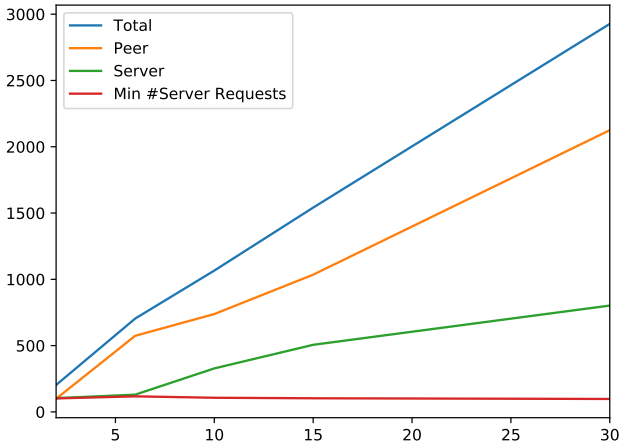

Figure 24: Vergleich der Anfragen nach Anfrageart

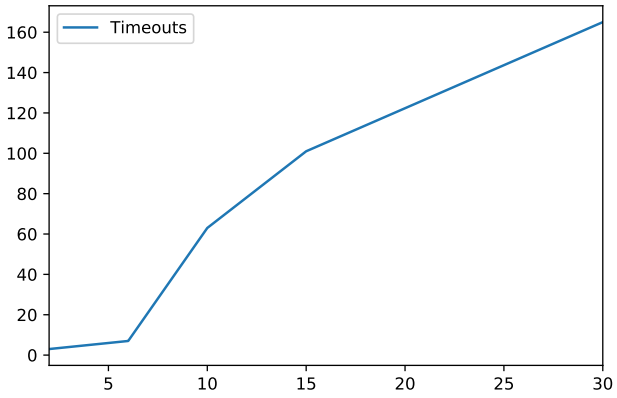

Figure 25: Anzahl der Timeouts des CDNs

an Anfragen über den Server geladen werden müssen. Ein möglicher Grund hierfür ist Die Zeitliche Verteilung der Anfragen. Befinden sich mehr Peers im Netzwerk so wird es wahrscheinlicher das zwei Teilnehmer sich beide an der selben Stelle im Video befinden, die HLS Segmente laden müssen, jedoch noch kein Teilnehmer das Segment heruntergeladen hat. Auch wird es wahrscheinlicher das eine größere Anzahl an Teilnehmern ein Segment über den selben Teilnehmer laden wollen. Fragen zu viele Teilnehmer eine Ressource bei selben Teilnehmer an, so kann er nicht mehr alle Anfragen bearbeiten. Die Anfrage über das Peer To Peer CDN wird abgebrochen und über den Server geladen. Abbildung 25 zeigt die Anzahl der Timeouts bei steigender Teilnehmerzahl. Dabei ist zu beachten das der Timeout so gewählt wurde das das Video weiterhin flüssig lief.

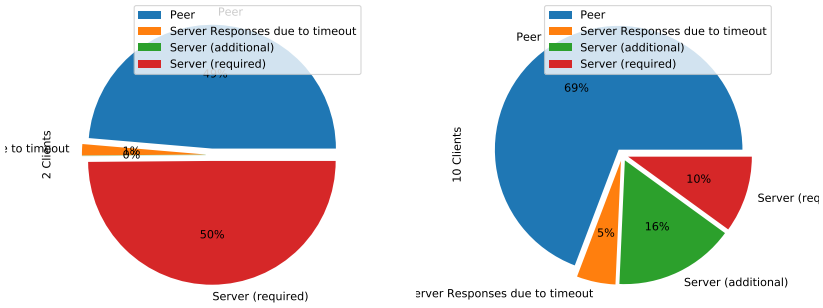

Figure 26: Verteilung der Anfragen bei zwei und 10 Clients

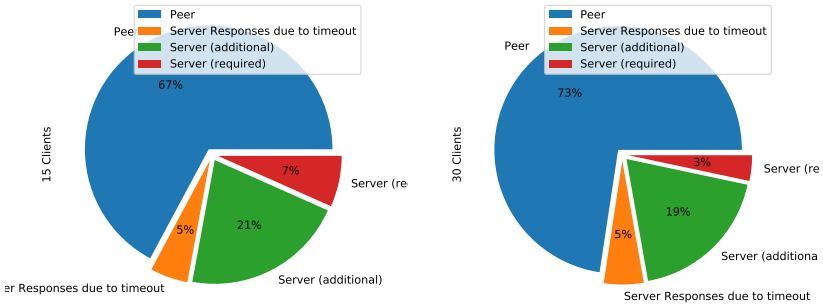

Figure 27: Verteilung der Anfragen bei zwei und 10 Clients

Betrachtet man die zusätzlich Notwendigen Server Anfragen so lässt sich beobachten das der Prozentsatz in diesem Test annähernd konstant ist.

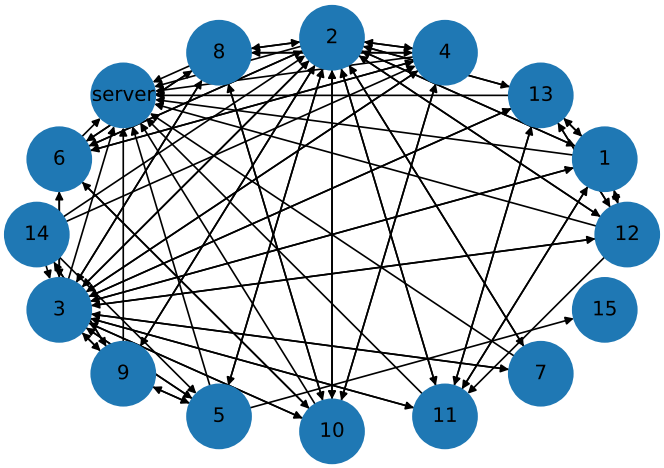

Figure 28: Vergleich der Anfragen nach Anfrageart

Abbildung 28 zeigt welche Teilnehmer untereinander Daten ausgetauscht haben. Auffällig ist das einige Teilnehmer besonders vielen Teilnehmern Daten gesendet haben. Dies war besonders dann der Fall wenn sie die einzigen im Netzwerk waren die das Segment bereits geladen hatten. Die meisten Kanten sind jedoch bidirektional, sprich zwar haben einige Teilnehmer besonders viele Anfragen beantwortet, jedoch haben sie sich nicht als zentrale Punkte des Netzwerkes etabliert sondern ebenfalls ihre Daten von anderen Teilnehmern geladen. *gewichtung??*

Request Art	Anfragen	Durchschnitt	Standardbweichung	Min	Max
Peer	4571	201.60 ms	190.14 ms	27.02 ms	2832.27 ms
Server	1870	63.61 ms	62.99 ms	9.41 ms	649 ms

Table 4: Browser Storage Quotas

Bei der Betrachtung der Ladezeiten wurden alle erfassten Anfragen berücksichtigt. Allerdings wurden die Daten um die Server Anfragen bereinigt bei denen zuvor das Peer To Peer CDN einen timeout verursacht hat. Insgesamt wurden 4571 Peer Anfragen und 1870 Server Anfragen betrachtet. Anfragen die vom Peer To Peer CDN beantwortet wurden brauchten im Durchschnitt 201,60ms. Server Anfragen waren im Vergleich mit 63,31ms im Durchschnitt deutlich schneller. Die ist auf den geringeren Durchsatz der Webrtc Datachannels zurückzuführen. (Siehe Dateigrößen) Damit war das Peer To Peer CDN zwar deutlich langsamer, jedoch schnell genug um eine flüssige Wiedergabe zu gewährleisten. Da eine Anfrage die länger als 3000ms brauchte bei dem Peer To Peer CDN zu einem timeout geführt hat betrug die höchste erfasste Ladezeit 2832,27 ms. *ref*

- andere auflösungen
- s vs p
- server downloadtime berücksichtigen
- besonders vergleich für hohe auflösungen
- vergleich verschiedener Auflösungen

6.6.1 *Nutzerzufriedenheit*

wie soll das aussehen?

6.7 SECURITY CONSIDERATIONS

6.8 DRM LICENCING

7

CONCLUSION

- Browser nutzung

A

AN APPENDIX

Some stuff here

BIBLIOGRAPHY

[1] Ali Alabbas and Joshua Bell. "Indexed Database API 2.0." In: *W3C Recommendation*. W3C, Jan. 2018. URL: https://www.w3.org/TR/2018/REC-IndexedDB-2-20180130/.

[2] Adam Bergkvist, Daniel C. Burnett, Cullen Jennings, Anant Narayanan, Bernard Aboba, Taylor Brandstetter, and Jan-Ivar Bruaroey. "WebRTC 1.0: Real-time Communication Between Browsers." In: *W3C Candidate Recommendation 27 September 2018*. W3C, Sept. 2018. URL: https://www.w3.org/TR/webrtc/.

[3] Daniel C. Burnett, Adam Bergkvist, Cullen Jennings, Anant Narayanan, and Bernard Aboba. "Media Capture and Streams." In: *W3C Candidate Recommendation*. W3C, Oct. 2017. URL: https://www.w3.org/TR/2017/CR-mediacapture-streams-20171003/.

[4] I. Fette and A. Melnikov. "The WebSocket Protocol." In: *Request for Comments: 6455*. Internet Engineering Task Force (IETF), Dec. 2011. URL: https://tools.ietf.org/html/rfc6455.

[5] Fiona Fui-Hoon Nah. "A study on tolerable waiting time: how long are Web users willing to wait?" In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163. DOI: 10.1080/01449290410001669914. eprint: https://doi.org/10.1080/01449290410001669914. URL: https://doi.org/10.1080/01449290410001669914.

[6] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)." In: *Network Working Group*. The Internet Society, Mar. 2003. URL: http://www.ietf.org/rfc/rfc3489.txt.

[7] Alex Russell, Jungkee Song, Jake Archibald, and Marijn Kruisselbrink. "Service Workers 1." In: *W3C Working Draft, 2 November 2017*. W3C, Nov. 2017. URL: https://www.w3.org/TR/service-workers-1/.

[8] Information Sciences Institute University of Southern California. *INTERNET PROTOCOL*. Aug. 1981. URL: https://tools.ietf.org/html/rfc791.

[9] Kurt Tutschku and Phuoc Tran-Gia. "Peer-to-Peer-Systems and Applications." In: Jan. 2005.

[10] Mingchen Zhao, Paarijaat Aditya, Ang Chen, Yin Lin, Andreas Haeberlen, Peter Druschel, Bruce Maggs, Bill Wishon, and Miroslav Ponec. "Peer-Assisted Content Distribution in Akamai NetSession." In: *Proceedings of the 2013 conference on Internet measurement conference Pages 31-42*. ACM New York, NY, USA, Oct. 2013.

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

DECLARATION OF ORIGINALITY

I hereby declare that I have prepared this master's thesis myself and without inadmissible assistance. I certify that all citations and contributions by other authors used in the thesis or which led to the ideas behind the thesis have been properly identified and referenced in written form. I also warrant that the above statement applies to the implementation of the project.

Hiermit versichere ich, dass ich die Masterarbeit selbständig und ohne unzulässige Hilfe Anderer angefertigt habe und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die entsprechenden Quellen angegeben habe. Ich erkläre hiermit weiterhin die Gültigkeit dieser Aussage für die Implementierung des Projektes.

Potsdam, October 5th, 2016

Tim Friedrich