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

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 volländigen 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-Streamings 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 Microservice 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 detailiert 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 wird auch ein Managed Service Angeboten bei dem Media Event Services das komplette Streaming übernimmt. Die Plattform ist für eine große Anzahl von Nutzern ausgelegt und ist hochverfügbar um den Ansprüchen von Unternehmen gerecht zu werden. Sie

1 https://schul-cloud.org/

stellt unter anderem Funktionen bereit um Events mit Registrierung und Foliensätzen zu realisieren.

1.4 ZIELE

1.4.1 *Forschungsfrage*

- Wie können in einem Netzwerk mit geringerer Internetanbindung Datenintensiven Resourcen 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 Resourcen 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*

- Nutzer generierter content
- Nutzer spezifischer inhalt
-

3.2 CDN

Detailierter 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. Infrastuktur 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 aktuallisieren 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 meherern tausend replica Servern. So hat Akamai, einer der größten CDN Anbietern, über 137000 Server in 87 Ländern. **[akamaiPeer]**

3.2.2 *Peer To Peer basierte CDNs*

3.2.3 *Hyrid CDNs*

Hybrid CDNs kombinieren Peer To Peer CDNs und Infrastuktur 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[**p2pBook2005**]:

- Heteorogenität 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.

3.5 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 aufzubauen und Daten direkt zwischen den Clients auszutauschen ohne das externe Plugins erforderlich sind. Insbesondere der Austausch von Multimedia Inhalten soll ermöglicht werden. Neben der Unterstützung für Video und Audio Inhalten gibt es jedoch auch die Möglichkeit Daten auszutauschen. Webrtc wird vom W3C[**w3Webrtc**] standarisiert und definiert eine Sammlung von APIs und Protokollen.

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

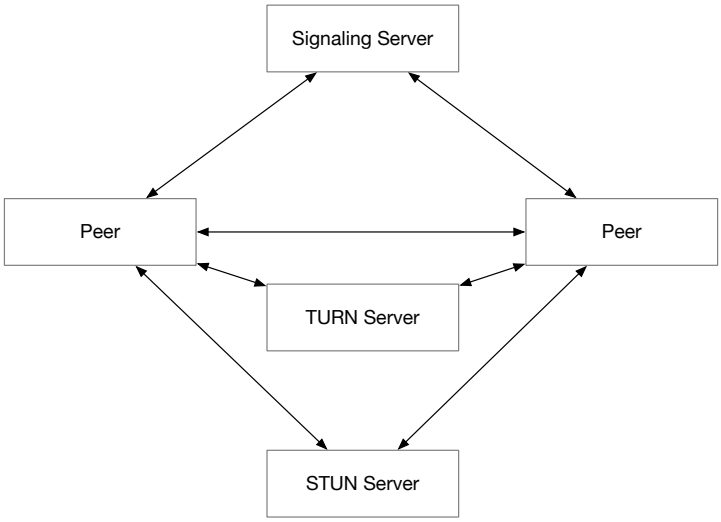

Figure 1: Überblick Server/Client Struktur Webrtc

3.5.1 *RTCPeerConnection*

- Representiert verbindung zum peer
- Code beispiel

3.5.2 *RTCDataChannel*

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

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

- 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

3.5.3 *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.[**w3MediaStream**]

3.5.4 *Singaling*

Das Signaling koordiniert die Kommunikation der Verbindungen zwischen Peers. Mit Hilfe des Signalings werden unter anderem die Metadaten ausgetauscht, die benötigt werden, um eine erfolgreiche WebRtc Verbindung aufzubauen. Unter anderem sind das:

- 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. Um Signaling zu ermöglichen ist ein bidirektionaler Kommunikationskanal zwischen client und server notwendig, was Websockets zu einem beliebten Kandidaten macht um das Signaling zu implementieren

3.5.5 *SDP*

- 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

3.5.6 *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.

3.5.7 *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 [**rfcStun**] definiert und ist nicht auf Webrtc beschränkt.

3.5.8 *ICE*

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

3.6 DATACACHE API

Die Datacache API ermöglicht es Netzwerk Requests zu cachen. Ursprünglich wurde die Api entwickelt um Serivce 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[**w3IndexedDB**] und soll den veralteten websql standard ablösen. Im gegensatz zu websql hat die IndexedDB keine strukturierte Query Language und ihr liegt kein relationelles 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 asyncron 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 Netwerk Proxies bereit. Durch Service Worker ist es möglich die Anfragen einer Seite zu kontrollieren auf sie zu reagieren und in den Prozess einzugreifen.[**w3ServiceWorker**] 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*

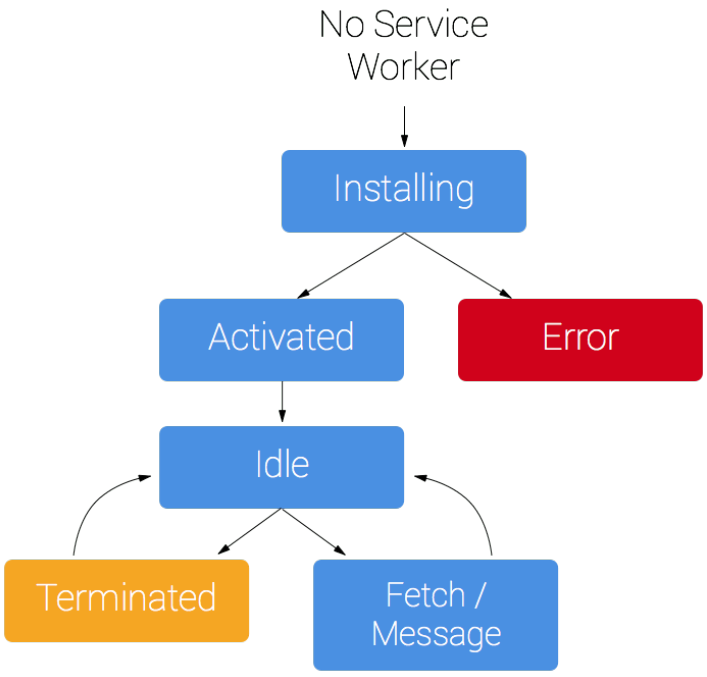

Figure 2: Lebenszyklus eines Service Workers[2]

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 benögten Zustand statt dessen 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 Anfrage des Clients Daten an ihn zu senden. Zum initieren 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[**rfcWebsockets**] 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[**rfcWebsockets**]

3.10 DISTRIBUTED CACHES

3.11 IP ADRESSEN

3.11.1 *Aufbau von IP Adressen*

3.11.2 *Network Adress Translation(NAT)*

-

3.12 RUBY ON RAILS

- Framework
- Wie detailiert???
- dogams etc
- Backround Jobs

3.13 REDIS

- In memory Datenbank

- Key Value Store mit Datentypen
- Verwendete Datentypen vorstellen
- set
- counter
- key value
- list

4

KONZEPT

Beim klassischen Client Server Modell muss sich jeder Client Ressourcen über das WAN laden. Abbildung 3 zeigt den typischen Aufbau eines solchen Netzwerks. Sind die geladenen Ressourcen groß kann die WAN Anbindung zu einem Flaschenhals werden. Durch die dadurch resultierenden langen Ladezeiten kann es zu einer starken Einschränkung des Nutzererlebnisses und der Nutzerzufriedenheit kommen.(*studie einfügen*)

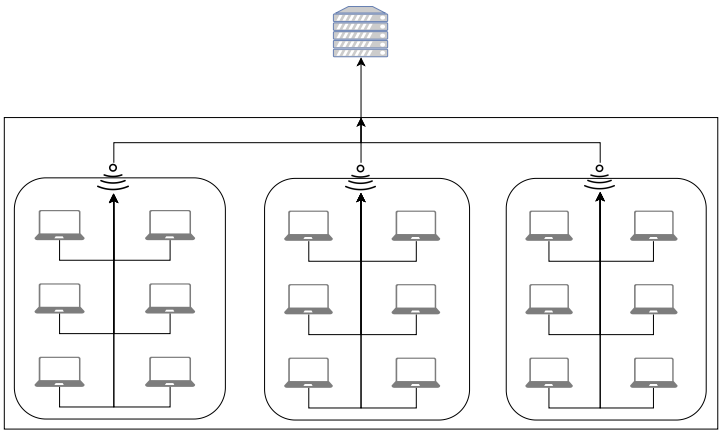

Figure 3: Netzwerkverkehr in einem herkömmlichen Netzwerk

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 Ansatzes untersucht werden. Abbildung 5 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 zuerst 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.

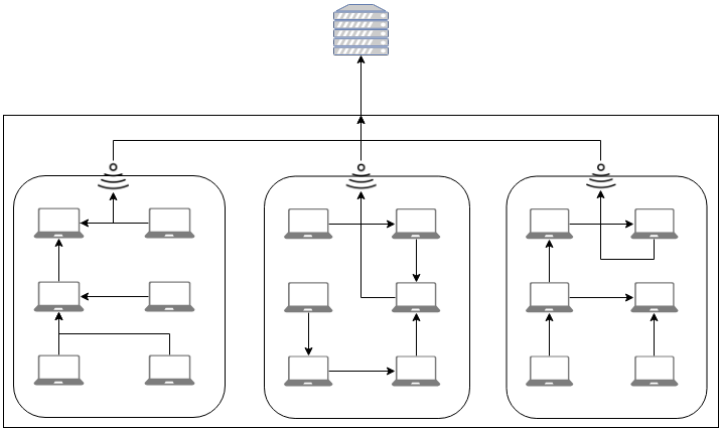

Figure 4: Netzwerkverkehr in einem Peer To Peer CDN

Da sowohl im Kontext der Schule als auch bei Unternehmen kein Wissen im Bereich der Computer Administration seitens der Nutzer vorausgesetzt werden kann, wurde ein Ansatz gewählt werden der keine Installation auf Seiten der Nutzer benötigt. Das Nutzererlebnis soll dabei nicht negativ beeinflusst werden.

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 Hauptanwendungsszenarien 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 hauptsaächlich 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. 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

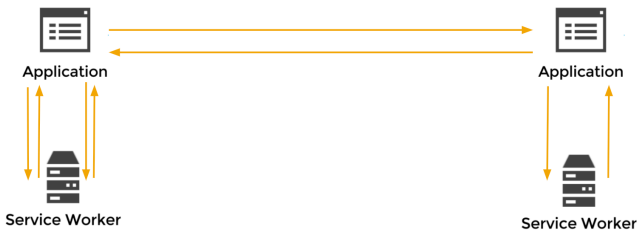

Figure 5: Service Worker - Webrtc

- Browser based without plugin
- einfache einbindung
- Server architektur diagram –> Server, Stun, CDN, P2P CDN
- Discussion verschiedener Request interception techniken
- Special Tags
- http interceptor
- only ajax calls

  - erst wenn seite komplett geladen
  - Browser Plugins
  - https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Intercept$_H$
- extra software auf client
- Service Worker – > WEBRTC
- Script für execution
- Webrtc - Datachannel
- Service worker cached request
- warum service worker
- vergleich anderer Ansätze
- Literatur anschauen

4.3 MESH ZUORDNUNG - VERBINDEN VON PEERS

Im Folgenden werden verschiedene Zuordnungsstragien zur Bildung von Peer Meshes diskutiert. Dazu wird der typische Workload beider Anwendungen analysiert um eine jeweils geeignete Strategie zu wählen.
- Vergleich von Meshing verfahren/peer routing aus literatur

4.3.1 *Routing*

Im Folgenden werden verschiedene Routing Peer To Peer Routing Algorithmen aus der Literatur vorgestellt und diskutiert welcher Ansatz für die betrachteten Anwendungsfälle am geeignetsten ist.

- Datenstruktur
- Dezentralisierte Datenspeicherung
- Daten werden über SPeicherknoten verteilt
- Jeder Knoten eintrag in Hashtabelle
- direct storage: Daten in Hashtabelle
- nur für kleine Daten
- indirect storage: Verweis auf daten in Hashtabelle
- Eigenschaften: Fehlertoleranz Lastenverteilung Robustheit Selbstorganisation Skalierbarkeit
- consistent hashing
- Server zum routen https://www.coralcdn.org/pubs/

4.3.1.1 *Chord*

How to save files in advance??? save file references

4.3.1.2 *Kademlia*

4.3.1.3 *IPFS*

4.3.1.4 *Kelips*

4.3.1.5 *Pastry*

4.3.2 *Schul-Cloud*

Analyse workload mit grafik

- Schulcloud
-

Schulcloud - Global einfache umsetzung Kurs unabhänige assets können geteilt werden Maximale Mesh größe Mehrere Meshes nötig Möglicherweise nicht im Mesh mit relevanten Peers Maximiert die Anzahl der Peers pro Mesh Wahrscheinlichkeit für selbes Subnetz eher gering Funktioniert auch mit wenigen Clients Ineffektiv wenn viele Clients vorhanden sind

- Schule Einfache Umsetzung Funktioniert auch mit wenigen Clients Relativ hohe trefferrate da schulen selten mehr als 1000 Schüler hat Max mesh größe um die 256(Chrome) Relativ wahrscheinlich gleiches Subnetz

- Kurs Hohe trefferrate Kleine Meshes benötigt mehr Clients Relativ wahrscheinlich gleiches Subnetz –> Daten die das bestätigen

hybride Ansätze: Zwei priorisierte Meshes Schule und Klasse

Berechnung eines Scores für jeden Peer: Möglichst viele gemeinsame kurse Liste von Kursen für jeden Peer Schnittmenge für jeden peer der online ist bilden Kardinalität der Menge = Score von Peer Rechenaufwendig Aber machbar Beste Trefferrate Funktioniert wenn wenige und wenn viele Peers anwesend sind

4.3.3 *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 besteht aus HLS Videoseqmenten. Jeder Teilnehmer eines Events benötigt die selben Inhalte. *Grafik visits* *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 Kommunkationsoverhead 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 6 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.

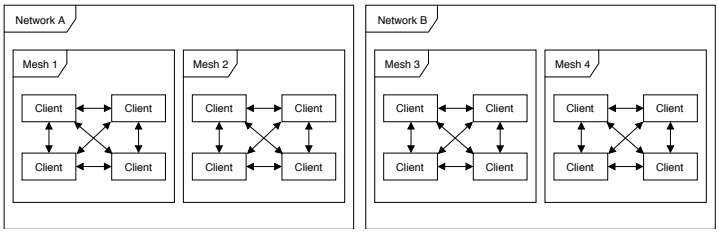

Figure 6: Peer to Peer Meshes - Slidesync

- Ip Subnetzerkennung

4.4 ROUTING - FINDEN VON RESSOURCEN

Um das Peer To Peer Netzwerk als CDN Nutzbar zu machen ist es wichtig das ein Peer in der Lage ist herauszufinden wer welche resource bereitstellt. Dazu lassen sich verschiedene Ansätze verfolgen.

DHT

distributed hash tables erklären

Structured vs untructured

Da das Routing von Ressourcen bei den Betrachteten Anwendungsfällen in einem Zeitkritischen Moment erfolgen muss wurde sich für einen anderen Ansatz entschieden. Jeder Peer hält eine Hashtabelle mit den Ressourcen seiner Peers vor. Fügt ein Peer eine neue Ressource zu seinem Cache hinzu oder entfernt sie muss er alle verbundenen Peers über diese Änderung informieren. Dadurch muss im Falle einer Anfrage nicht erst die Ressource im Netzwerk gesucht werden. Dies ist möglich durch die Struktur des Netzwerkes. Da nicht alle Peers miteinander verbunden sind sondern voll vermaschte submeshes gebildet werden ist es möglich alle relevanten Peers über Änderungen zu informieren. Dadurch ist es möglich die Rechenleistung

für das auffinden von Ressourcen in einen weniger Zeitkritischen Moment zu verlagern. Jedoch hat dies zur Folge das die Meshes so gebildet werden müssen das die Peers möglichst viele Ressourcen gemeinsam benötigen. Ist eine Ressource nicht im Mesh vorhanden kann sie vom Server geladen werden. *Duplicate*

4.4.1 *Updates*

- Diagramm
- Client lädt ressource
- Client cached Resource
- Sendet nachricht an peers das Resource gecached wurde und verfügbar ist
- Clients speicher Client Resource hash in Hashmap

4.4.2 *Subnetzerkennung*

4.4.3 *Struktur von IP Adressen*

eher Konzept

4.5 WIEDERVERWENDBARKEIT

4.6 OPEN SOURCE

- evtl. Open source erklären
- bereitstellung für die öffentlichkeit
- Welche Lizenz?

4.7 OFFLINE SUPPORT

- Motivation: Internet in Schulen ist nicht immer verfügbar
- 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
- 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

4.8 SECURITY

- 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

5

IMPLEMENTIERUNG

5.1 ARCHITECTUR

- Technologiewahl
- ES6 compilation
- Beschreibung der Komponenten:
- Service Worker
- Cached Ressourcen
- Proxy
- P2PCDN
- Schnittstelle für externe Anwendungen
- Nimmt Konfiguration entgegen
- Initialisiert CDN
- Middleware
- Vermittler zwischen Service Worker und Script
- Arbeitet mit Events
- Welche Events gibt es wer registriert sich drauf?
- Peer
- Representiert eigenen Peer
- Hält verbindungen zu anderen Peers
- Signaling
- FayeConnection

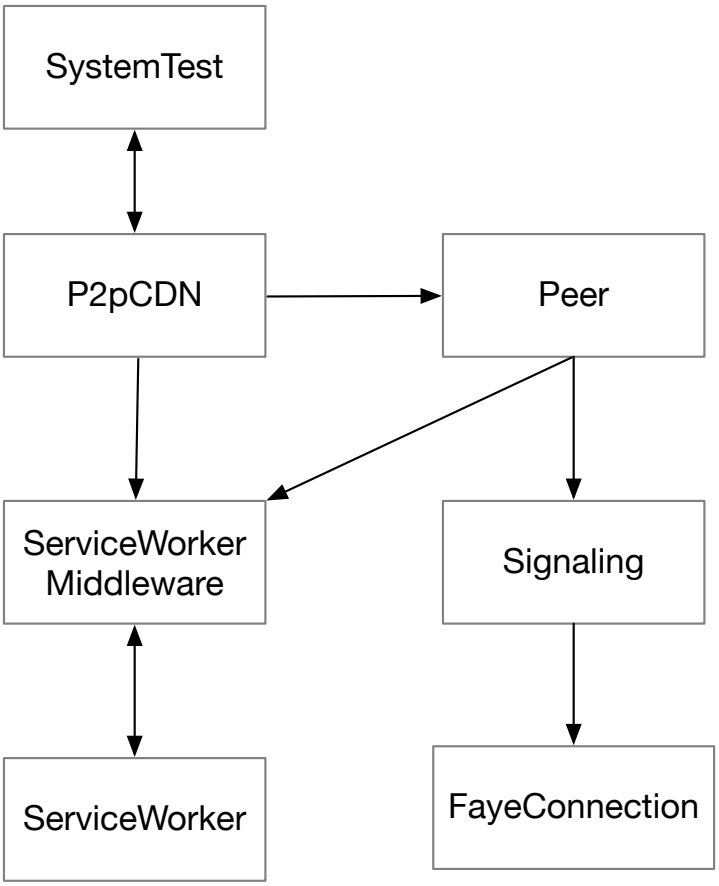

Figure 7: Klassendiagramm

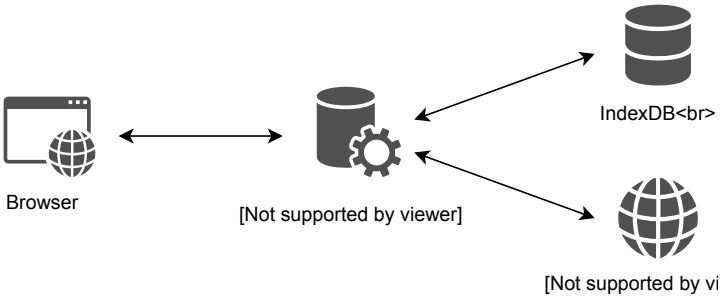

Figure 8: A Figure Title

5.1.1 *Service worker*

- warten bis sw active ist/alle verbindungen ready sind vs request normal durchgehen lassen bis alles fertig ist.
- Austausch erst möglich wenn script geladen ist
- blockent:
- Ressourcen werden so lange aufgehalten bis verbindung initialisiert

- –> längere Ladezeiten
- mehr Bandbreite kann gespart werden
- nicht blockent:
- Ressourcen werden über server geladen bis verbinungen aufgebaut sind
- Weniger Bandbreite wird gespart
- Ladezeit wird nicht verzögert
- Heartbeat um zu überprüfen ob script geladen ist
- Mehrere Tabs?
-
- first page load
- Single page apps
- Turbolinks
- Requests werden erst nach erfolgreicher registrierung behandelt
- clients.claim + skip waiting für den ersten aufruf (12)
- Codebeispiel für wartende messages
- Callbacks

5.1.2 *Tests*

- mocha
- chai
- karma
- event handler testen
- registrieren/abmelden –> test helper

5.2 RESSOURCEN MANAGEMENT

5.3 CONFIGURATION

- id
- id length
- beispiel

- verbose
- ...
- Config Übergabe zu Service worker über IndexedDB
- Script lädt Config in indexedDB
- Service worker lädt config falls noch keine vorhanden ist und im activate event

5.3.1 *Quota limits - Löschen von Requests aus dem Cache*

- benötigte speichermenge wird geschätzt
- headersize plus 'Content-Length' header wert
- solange löschen bis genug speicher frei ist
- Quotas flexibel
- Quota kann abgefragt werden
- Quota für browser nicht für page
- berechnung der größe schwer
- löschen des ersten Elements im Cache
- evtl verschiedene verfahren diskutieren
- Fehlerfall muss abgefangen werden

5.4 CLIENT UI EVENTS

- Events um die einbindende Anwendung über neue/gelöschte Peer Verbindungen zu informieren
- ui:onUpdate

5.5 SIGNALING SERVER

The signaling server itself uses socket.io and can be found here. The client ID is created there and is essential for the lifecycle of a peer in the whole network. It is not clear if the client IDs given by socket.io always have the same length. Therefore, client IDs will be padded to a maximal length of 24. This is necessary because the client IDs need to be sent via the binary datachannel and consequently, this requires a fixed length.

- websockets faye

- gleicher data channel zur Bildung von meshs
- ID pro client
- max id length muss festgelegt werden für chunking
- Protokoll von P2PCDN umgesetzt
- Protokoll erklären
- Diagram Protokoll
- Peer Meshing wird von anwendung übernommen

5.5.1 *Slidesync*

- Rails Model
- Code Beispiel
- Tracking in Redis welcher Peer in welchem Submesh ist
- Peer wird aus liste nicht voller mehses gewählt
- Falls alle voll sind wird neues Mesh angelegt und garbage collection gestartet.
- Datentyen und mapping erklären
- Background job räumt leere meshes auf (Garbage collection)
- löscht leere meshes aus liste

5.5.2 *Schul-Cloud*

5.6 MESSAGE PROTOCOL

Diagramm aktualisieren

5.6.1 *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

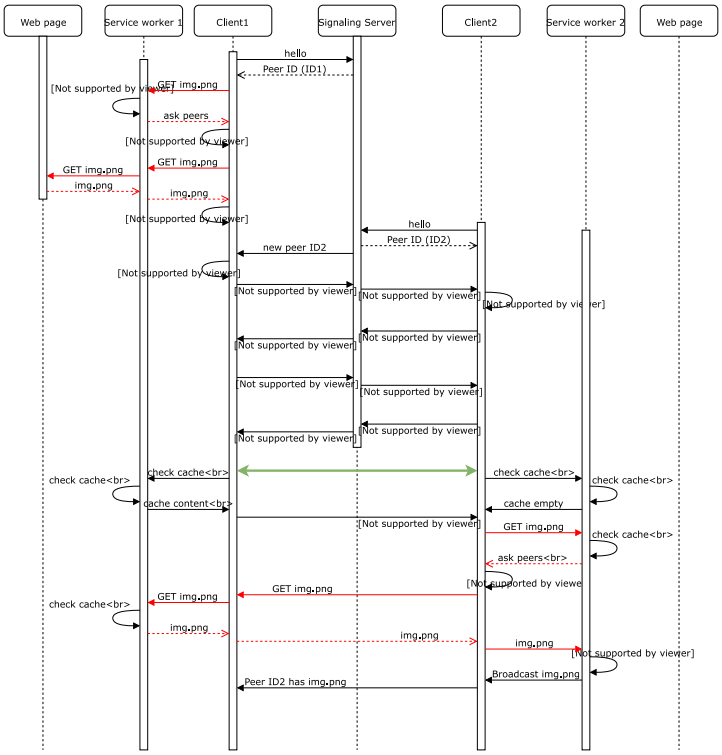

Figure 9: A Figure Title

5.6.2 *Client fragt Ressource an*

- Message Format
- Diagram beschreiben

5.7 MESH ZUORDNUNG

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. In unserem Kontext ist es aber notwendig auch größere Dateien zu übertragen, weshalb aktuell viele kleine Datenpakete verwendet werden müssen. Hierdurch entsteht ein nicht zu vernachlässigender Overhead.

- SCTP(DataChannel) message size limit 16kiB
- chunking reassembling nötig

- Performance einbussen
- abToMessage und sendToPeer
- Details zum Algorithmus

5.10 SYSTEM TEST

- Browser Support
- modernizer
- testet auf Browserversion
- Test von Verbindungsaufbau
- mit anderen Clients verbunden?
- Code Beispiel verwendung des Interfaces

5.11 ERFASSEN VON STATISTIKEN

- Client Script sendet an Service Worker von wem etwas geladen wurde
- Service Worker trackt wie Request abgearbeitet wurde
- Service Worker sendet periodisch alle Metadaten an Server
- Server trackt statistiken
- Fall Slidesync:
- Live statistik tracking
- evtl. Mongo db
- anzeigetool

5.12 STORAGE QUOTAS

- Wichtig um fehlervorzubeugen
- Browser besitzt dynamische Quotas, abhänig von
- Quoata für CDN kann über einstellungen festgelegt werden.
- navigator.storage.estimate()
- Quota Management API
- LRU falls quota erreicht

  - Löscht kompletten Cache für eine Seite
  - https://developers.google.com/web/updates/2017/08/estimating-available-storage-space
  - https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Browser_storage_li

- Round Robin
- Größe des Requests wird geschätzt
- Exakte größe schwer berechnen bar da cache komprimiert
- Größe des Kontents plus größe des Headers plus offset
- Cache wird geleert bis genug platz frei ist.

```
class ContextData < TransformStep

  def transform(original_event, processing_units, load_commands, pipeline_ctx)
    processing_units.each do |processing_unit|
      next if processing_unit[:in_context].nil? || processing_unit[:in_context][:user_agent].nil?

      user_agent = processing_unit[:in_context][:user_agent]

      browser = Browser.new(user_agent)

      processing_unit[:in_context][:platform] = browser.platform.name
      processing_unit[:in_context][:platform_version] = browser.platform.version
      processing_unit[:in_context][:runtime] = browser.name
      processing_unit[:in_context][:runtime_version] = browser.version
      processing_unit[:in_context][:device] = browser.device.name
    end
  end

end
```

Listing 1: Some Code Snipped

5.13 RESOURCE LOADING

6

EVALUATION

6.1 PREQUISITES

6.2 TECHINCAL EVALUATION

6.2.1 *Bandwidth*

6.2.1.1 *Simulierter Workload*

- batches
- gleicher Zeitabstand gleiche Client zahl
- Random client ankunft
- Future Soc lab

6.2.1.2 *Educational context*

6.2.1.3 *Live streaming in the coporate context*

6.2.2 *Nutzerzufriedenheit*

wie soll das aussehen?

6.3 BROWSER COMPATBILITY

- Kein IE
- Edge nur in neuer version mit chrome dahinter
- Chrome
- Firefox
- Analyse von verwendeten Clients aktueller Events

7

CONCLUSION

All in all, this is a nice thesis

A

AN APPENDIX

Some stuff here

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