MASTER'S THESIS

DISTRIBUTION OF LARGE DATA IN NETWORKS WITH LIMITED BANDWIDTH

MEINE THESIS ÜBER E-LEARNING UMGEBUNGEN

TIM FRIEDRICH

CHAIR

Internet Technologies and Systems

SUPERVISORS

Prof. Dr. Christoph Meinel

Dipl.-Inf. (FH). Jan Renz

October 5th, 2016

[January 21, 2019 at 16:12 – classicthesis]

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 beschäftigt sich mit der Frage ob ein Peer to Peer Ansatz eingesetzt werden kann um die Ladezeiten in Netzwerken mit begrenzter WAN Bandbreite die Ladezeiten zu verringern.

1.2 LOW BANDWITH NETWORKS

1.2.1 *Educational networks*

1.2.2 *Corporate networks*

1.3 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 ein Microservice Architektur. 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

1.4 SLIDESYNC

1.5 ZIELE

1.5.1 *Forschungsfrage*

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

1 https://schul-cloud.org/

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

1.6

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 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.

3.2 CDN

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. [8]

3.2.2 *Peer To Peer basierte CDNs*

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. Man unterscheidet zwischen unstrukturierten und strukturierten Peer To Peer Netzwerken.

Unstrukturierte Netzwerke haben keine Zuordnung von Objekten und Teilnehmern. Es ist nicht möglich gezielt nach einem Objekt zu suchen. 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.

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.

Typische wenn auch nicht notwendige Charakteristika sind laut Steinmetz[7]:

- 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

Reine Peer To Peer CDN Systeme sind im Kontext von Internet Anwendungen eher selten zu finden. Ein klassischer Anwendungsfall für diese Systeme ist das Filesharing.

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 WEBRTC- WEB REAL-TIME COMMUNICATION

Webrtc ist ein Framework 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 ist ein vom W3C[2] standarisierter offener Standart an dem seit Frühjahr 2011 gearbeitet wird. 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.

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

3.3.1 *RTCPeerConnection*

3.3.2 *RTCDataChannel*

3.3.3 *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.[3]

3.3.4 *Singaling*

3.3.5 *SDP*

3.3.6 *TURN Server*

3.3.7 *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 [5] definiert und ist nicht auf Webrtc beschränkt.

3.3.8 *ICE*

3.4 DATACACHE

3.5 INDEXEDDB

IndexedDB ist ein HTML 5 Feature um Daten im Browser zu speichern. Es wurde vom W3C standarisiert[1] 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.

3.6 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.[6] 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.

3.6.1 *Lebenszyklus*

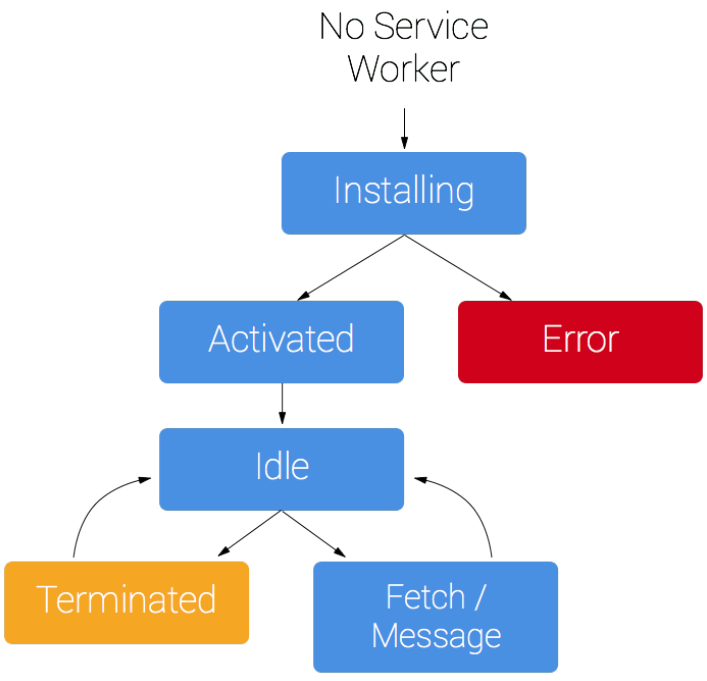

Figure 1: Lebenszyklus eines Service Workers[2]

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.

3.7 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[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]

3.8 DISTRIBUTED CACHES

4

KONZEPT

Beim klassischen Client Server Modell muss sich jeder Client Ressourcen über das WAN laden. Abbildung 2 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*)

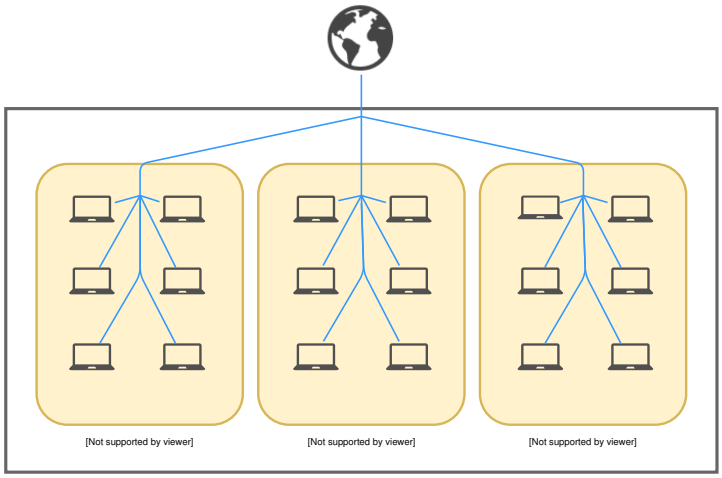

Figure 2: Lebenszyklus eines Service Workers[1]

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 3 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.

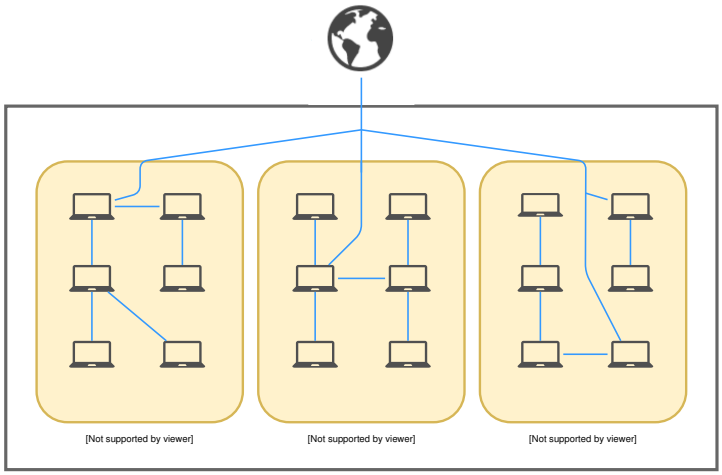

Figure 3: Lebenszyklus eines Service Workers[2]

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

4.1 MESH ZUORDNUNG - VERBINDEN VON PEERS

4.2 OBJEKT ZUORDNUNG - 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.

server hält liste vor –> kommunikation mit server nötig. *Studie*

peer fragt im Netzwerk an –> mehr kommunikation wenn in zeitkritischem moment *Studie*

peer halten liste von requests aktuell. Kommunikationsoverhead in zeitunkritischem moment *Studie*

4.2.1 *Subnetzerkennung*

4.2.2 *Struktur von IP Adressen*

eher Konzept

4.3 WIEDERVERWENDBARKEIT

4.4 OPEN SOURCE

5

IMPLEMENTIERUNG

Für unsere Implementation wird für das Zwischenspeichern von Daten ein Serviceworker eingesetzt. Serviceworker können wie ein Proxy zwischen dem Webbrowser und dem Webserver agieren, welcher die Webseite bereitstellt. Stellt ein Browser eine Anfrage, so wird diese vom Serviceworker abgefangen. Der Serviceworker schaut zunächst in seinem Cache, der sog. IndexDB, ob er die gestellte Anfrage beantworten kann. Ist dies nicht der Fall, so wird die Anfrage an den Webserver weitergeleitet. Wird die gleiche Anfrage nochmals gestellt, kann diese aus dem Cache beantwortet werden, da gestellte Anfragen eine gewisse Zeit lang zwischengespeichert werden.

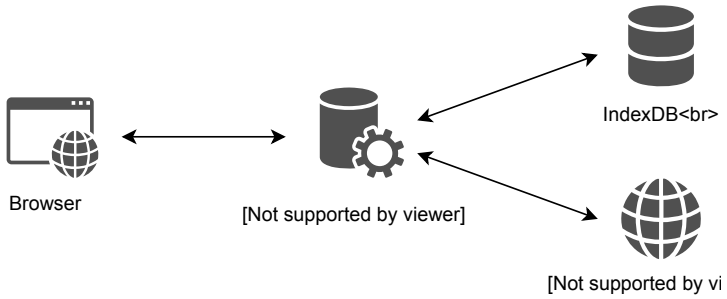

Figure 4: A Figure Title

Die von uns 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 wird von uns ein sog. DataChannel genutzt.

Für den Datenaustausch müssen wechselseitig DataChannel zueinander aufgebaut werden. Die Ausgangslage ist, dass die Schüler 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. Technisch wird ein SDP-offer gesendet, wobei SDP für Session Description Protocol

steht. Dieses SDP-offer leitet der Signaling server an die Schüler in der Klasse/Schule weiter. Geantwortet wird mit einer SDP-answere, 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 Conectivity 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.

In dem Signaling server selbst wird die Logik abgebildet, wie die Klassen und Schüler miteinander in Verbindung stehen. Implementiert wurde dieser mit socket.io, da die native Klassenorganisation und Websocket-Technologie sich nahezu perfekt für unser Szenario anbot.

5.1 ARCHITECTUR

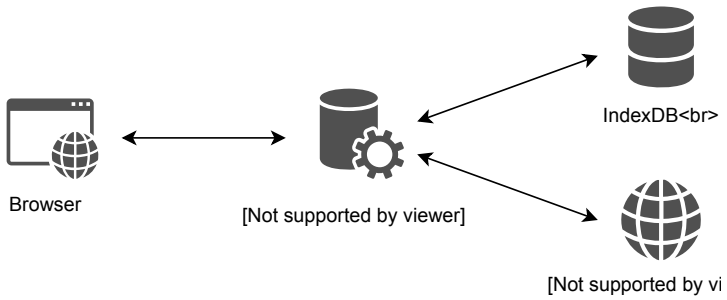

Figure 5: A Figure Title

5.1.1 *Service worker*

- warten bis sw active ist/alle verbindungen ready sind vs request normal durchgehen lassen bis alles fertig ist.
- Requests werden erst nach erfolgreicher registrierung behandelt
- clients.claim + skip waiting für den ersten aufruf (12)

5.1.2 *Tests*

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

5.2 RESSOURCEN AUS DEM CACHE LÖSCHEN

- benötigte speichermenge wird geschätzt
- headersize plus 'Content-Length' header wert
- solange löschen bis genug speicher frei ist

5.2.1 *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 socket io
- gleicher data channel zur Bildung von meshs

5.3 MESSAGE PROTOCOL

In unserer Implementation wird, sobald eine neuer Besucher der Webseite hinzukommt, sofort ein DataChannel, mittels WebRTC, STUN, ICE und Signaling server zu allen anderen aktiven Besuchern aufgebaut. Über diesen werden zu zwei Zeitpunkten Informationen darüber ausgetauscht, welche Ressourcen bei dem jeweiligen Nutzer vorliegen: Direkt nach Aufbau des DataChannels und immer dann, wenn ein Nutzer eine neue Ressource (aus dem Internet oder lokal) geladen und in seinem Cache gespeichert hat:

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

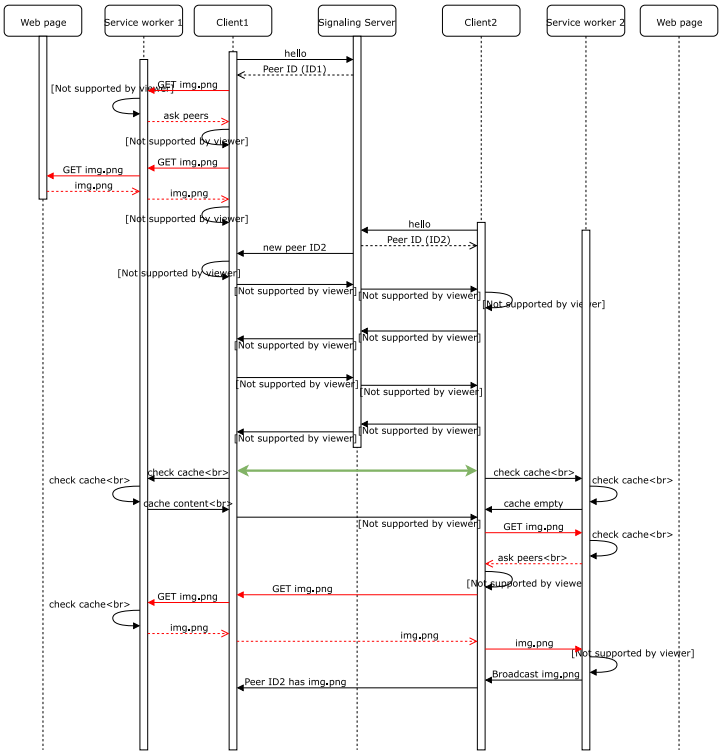

Figure 6: A Figure Title

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

5.3.1 *hash updates*

message types

5.4 DATA SERIALIZATION

- SCTP(DataChannel) message size limit 16kiB
- chunking reassembling nötig
- Performance einbussen
- abToMessage und sendToPeer

5.5 MESH ZUORDNUNG

5.6 REUSABILITY

5.7 SYSTEM TEST

5.8 STORAGE QUOTAS

navigator.storage.estimate()

The minted package is really nice for code formatting.

```ruby
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

5.9 RESOURCE LOADING

6

EVALUATION

6.1 PREQUISITES

6.2 TECHINCAL EVALUATION

6.2.1 *Bandwidth*

6.2.1.1 *Simulierter Workload*

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

6.3.1 *Browser Usage in coropate networks*

6.3.2 *Browser usage in eduational networks*

6.4 SECURITY CONSIDERATIONS

6.5 DRM LICENCING

7

CONCLUSION

All in all, this is a nice thesis

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] 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.

[6] 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/.

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

[8] 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