MASTER'S THESIS

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 22, 2019 at 22:47 –]

Tim Friedrich: *Webbasierte Verteilung großer Datenmengen in lokalen Netzwerken* , 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. Laut Statista belief sich das weltweite Datenvolumen auf 33 Zettabyte und wird sich bis zum Jahr 2025 auf 175 Zettabyte steigern.[26] Zwar ist die vorhandene Bandbreite bei vielen Nutzern ebenfalls gestiegen, jedoch ist die Bandbreite vor allem in ländlicheren Regionen oft noch nicht ausreichend.[15] Insbesondere dann wenn viele Nutzer sich gemeinsam eine Internetanbindung teilen müssen, ist dies ein Problem.

Immer mehr Unternehmen halten Ihre Hauptversammlungen, Kundgebungen, und Pressemitteilungen über Live-Streams im Internet ab. Dies stellt sie vor das Problem, dass 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, beläuft sich aber schon bei mittelständischen Unternehmen auf bis zu 25000 Euro pro Stunde.[22]

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 zu geringe Bandbreiten. Viele Schulen haben eine schlechtere Internetanbindung als viele Privathaushalte.[10] Um 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. Eine gängige Methode, dies zu erreichen, ist die Installation von lokalen Schulservern oder Caching Appliances. Diese Lösungen bringen jedoch einige Nachteile mit sich. So muss eine Schule die fachlichen und räumlichen Kapazitäten zur Installation und Wartung der Server bereitstellen.

Beide Anwendungsfälle haben gemeinsam, dass viele Nutzer die selben Inhalte zur annähernd gleichen Zeit benötigen und zum aktuellen Zeitpunkt häufig über das Internet laden müssen. Diese zeitliche und inhaltliche 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 HPI SCHUL-CLOUD

Das Projekt Schul-Cloud[1] ist ein Gemeinschaftsprojekt des Hasso-Plattner-Instituts und des nationalen Excellence-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 webbasierte 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.

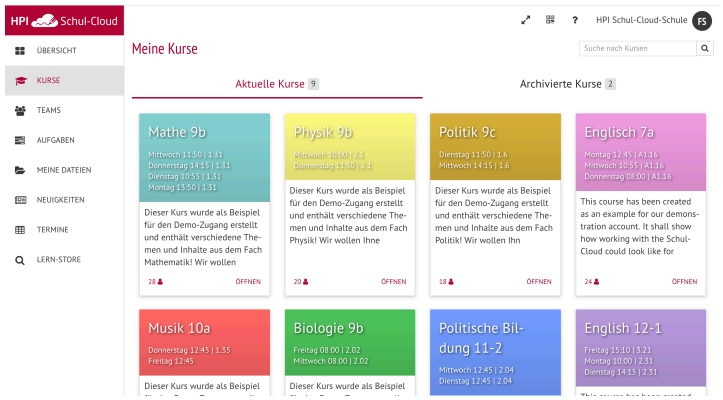

Figure 1: Schul-Cloud Webseite

Das Projekt wird als Open Source zur Verfügung gestellt und basiert auf einer Microservices-Architektur. Bei diesem Architekturmuster setzt sich die Software aus unabhängigen Softwarekomponenten (Services) zusammen. 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

1 https://schul-cloud.org/

Anbindung an bestehende Infrastrukturen ermöglicht. Desweiteren können einzelne Services ersetzt werden, um die Plattform an die Anforderungen der Schulen anzupassen. Bereitgestellt wird die Plattform mit Hilfe von Cloud Hosting, bei dem die Infrastruktur zentral und nicht von jeder Schule bereitgestellt wird. Dies ermöglicht eine einfache Skalierung. Neben der Webanwendung existieren native Apps für Android und iOS.

1.3 SLIDESYNC

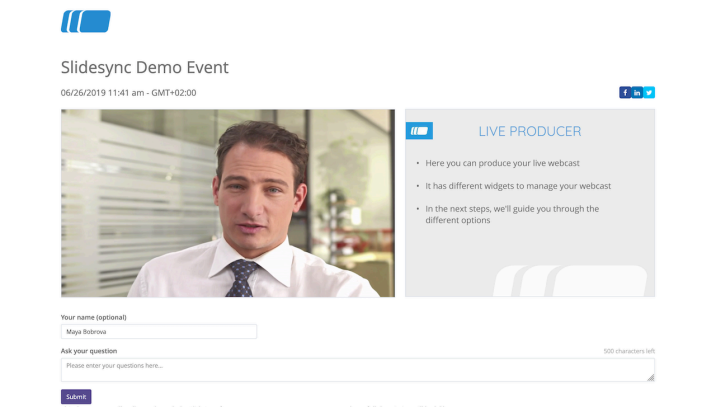

Figure 2: Slidesync Event Seite

Slidesync[2] ist eine Live-Streaming Plattform des Unternehmens MediaEvent Services GmbH Co. KG. 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 MediaEvent 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 werden. Sie stellt unter anderem Funktionen bereit, um Veranstaltung mit Registrierung und individualisierbaren Präsentationen zu realisieren. Die Plattform wird in Ruby on Rails entwickelt und stellt neben den Anwendungsservern auch eigene Streaming-Server bereit.

1.4 ZIELE

1.4.1 *Forschungsfrage*

- Wie können in einem Netzwerk mit begrenzter Internetanbindung datenintensive Ressourcen ausgeliefert werden?

2 www.slidesync.com

- Eignet sich ein Peer-to-Peer-Ansatz, um die benötigte Internetbandbreite von Schulen im Unterricht und Unternehmen im Rahmen von Livestreams zu verbessern?

Diese Arbeit wird versuchen die Frage zu beantworten, wie in einem Netzwerk mit geringerer Internetanbindung datenintensive Ressourcen ausgeliefert werden können.

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

2

ABGRENZUNG

Im Rahmen dieser Arbeit werden ausschließlich Peer-to-Peer-basierte Ansätze für die Inhaltsverteilung untersucht. Es wird untersucht, ob ein rein browserbasiertes CDN geeignet ist, die benötigte Bandbreite von Live-Streams beim Einsatz in Unternehmen und in Schulen zu reduzieren. Nicht näher betrachtet werden die Verteilung von On-Demand-Videos sowie Lösungsansätze, die eine Installation weiterer Software benötigen. Da es in dieser Arbeit um die Reduzierung von Bandbreite in geteilten Netzwerken geht, wird nicht näher untersucht, ob das CDN ebenfalls dazu geeignet ist, Inhalte über ein lokales Netzwerk hinaus zu verteilen. Auch wenn das vorgestellte CDN sowohl in der Schul-Cloud als auch bei Slidesync produktiv eingesetzt werden soll, ist es nicht Ziel dieser Arbeit, alle dafür notwendigen Hintergründe zu behandeln. Wichtige dafür notwendige Themenbereiche, wie z.B. DRM-Lizensierung, werden nicht näher betrachtet und bedürfen weiterer Untersuchung. Vielmehr handelt es sich um eine Machbarkeitsstudie, die ermitteln soll, ob der Ansatz in der Praxis eingesetzt werden kann.

3

GRUNDLAGEN

In dem folgenden Kapitel werden Technologien und Konzepte vorgestellt die für das Verständnis der Arbeit erforderlich sind. Es wird angenommen das der Leser über grundlegendes Verständnis der Funktionsweise des Internets und der Webentwicklung verfügt.

3.1 RESSOURCEN

Unter statischen Ressourcen werden im Rahmen dieser Arbeit Inhalte einer Website verstanden, die für alle Nutzer gleich sind. Sie sind im Gegensatz zu dynamischen Inhalten nicht nutzerspezifisch und können daher gut über ein CDN (siehe 3.2) verteilt werden. Insbesondere die so genannten Assets einer Internetseite sind meist statisch. Dies sind meist Javascript- und CSS-, aber auch Bild-Dateien. Auch Videos fallen häufig in diese Kategorie.

Im Gegensatz zu statischen Ressourcen werden im Kontext dieser Arbeit Inhalte als dynamisch generiert bezeichnet, wenn sie zur Laufzeit der Website erzeugt werden. Dabei lässt sich zwischen nutzergenerierten Inhalten und automatisch generierten Inhalten, z.B. Statistiken, unterscheiden.

Dynamische Inhalte können nutzerspezifisch sein, in diesem Fall werden jedem Nutzer bei selber Abfrage andere Inhalte angezeigt.

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 geringere Netzlaufzeiten erreicht.

Es lassen sich drei Klassen von CDNs unterscheiden. Infrastrukturbasierte CDN, die auf einer geografisch verteilten Serverinfrastruktur basieren, Peer-to-Peer-basierte CDNs bei denen die Inhalte direkt zwischen den Teilnehmern verteilt werden, und hybride CDNs

die auf einer Kombination aus Serverinfrastruktur und Peer-to-Peer-Verteilung beruhen.

3.2.1 *Infrastrukturbasierte-CDNs*

Infrastrukturbasierte CDNs bestehen aus einem Ursprungsserver, der von dem Bereitsteller der Inhalte kontrolliert wird, und einem Netzwerk aus Replikatservern. Die Replikatserver 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 Replikaten zu aktualisieren und übernimmt das Routing bei einer Anfrage eines Clients. Unter Zuhilfenahme verschiedener Metriken versucht das Distributionssystem, einen möglichst optimalen Replikatserver für den Client zu finden. Diese Metriken unterscheiden sich zwischen den Anbietern. Häufig werden jedoch geografische Entfernung, Latenzzeiten und die Übertragungsrate berücksichtigt. Um eine möglichst geringe Latenz zu erreichen, sind infrastrukturbasierte CDNs häufig geografisch sehr verteilt und bestehen aus mehreren tausend Replikaservern. So besitzt Akamai, einer der größten CDN-Anbieter, über 137000 Server in 87 Ländern. [47]

3.2.2 *Peer-to-Peer-basierte CDNs*

Bei einem Peer-to-Peer-basierten CDN laden Clients Inhalte nicht von einem Ursprungsserver, sondern von einem anderen Client. Dazu bilden sie Overlay-Netzwerke und setzen Techniken von Peer-to-Peer-Netzwerken zum Auffinden von Ressourcen ein.(siehe 3.3) Dabei gilt es zwischen Filesharing-Netzwerken und CDN-Netzwerken zu unterscheiden. Zwar lassen sich prinzipiell für beide dieselben Techniken verwenden, jedoch zielen Filesharing-Netzwerke auf die Verteilung von Dateien zur späteren Verwendung ab. CDNs wiederum werden unter anderem von Websites zur Auslieferung von statischen Inhalten verwendet. Während es bei Filesharing-Netzwerken akzeptabel sein kann, dass das Auffinden von Ressourcen eine gewisse Zeit in Anspruch nimmt, ist dies bei CDNs nicht der Fall.

3.2.3 *Hybride CDNs*

Hybrid CDNs kombinieren Peer-to-Peer-CDNs und infrastrukturbasierte CDNs. Bei hybriden CDNs wird zuerst versucht, die Resource über das Peer-Netzwerk zu laden. Ist dies nicht möglich, wird auf ein infrastrukturbasiertes CDN zurückgegriffen. 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 Voraussetzungen mitbringen, um an dem Peer-to-Peer-Netzwerk teilzunehmen, ist eine entsprechende alternative Lösung nötig. Da viele Websites bereits mit einem infrastrukturbasierten CDN arbeiten, liegt nahe, dieses weiter zu verwenden.

3.3 PEER-TO-PEER-NETZWERKE

Bei einem Peer-to-Peer-Netzwerk handelt es sich um eine Netzwerkstruktur, bei der alle Teilnehmer gleichberechtigt sind. Sie bildet damit das Gegenkonzept 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[40]:

- Heterogenität der Internetbandbreite der Teilnehmer
- Verfügbarkeit und Qualität der Verbindung zwischen Teilnehmern kann nicht vorausgesetzt werden
- Dienste werden von den Teilnehmern angeboten und genutzt
- 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

Peer-to-Peer-Netzwerke 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 dass 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.

Je nachdem, ob die Overlay-Struktur eines Peer-to-Peer-Netzwerkes hierarchisch ist oder nicht, spricht man von hierarchischen oder nicht hierarchischen Peer-to-Peer-Netzwerken. Die meisten zentralisierten und hybriden Netzwerke verfügen über eine hierarchische Overlay-Struktur. Hierarchische Overlays bieten gute Skalierbarkeit und Effizienz beim Routing. Reine Peer-to-Peer-Netzwerke haben häufig eine

nicht hierarchische Overlay-Struktur. Nicht hierarchische Netzwerke haben den Vorteil, die Last gut auf viele Peers verteilen zu können und verfügen über eine hohe Ausfallsicherheit.[17]

Peer-to-Peer-Netzwerke lassen sich des Weiteren nach Datenspeicherort und Netzwerkstruktur in unstrukturierte, lose strukturierte und stark strukturierte Netzwerke unterteilen. [17]

In unstrukturierten Peer-to-Peer-Netzwerken bestehen keine Regeln dafür, wo Daten gespeichert werden. Die Zuordnung ist willkürlich. Ein Beispiel für diese Art von Peer-to-Peer-Netzwerken ist das Gnutella Netzwerk.[28] Um ein Objekt zu finden, müssen alle Teilnehmer des Netzwerks gefragt werden. Dadurch steigt die Belastung des Netzwerks mit zunehmender Anzahl an Peers. Um die Last auf Seiten der Peers zu reduzieren, wurden eine Reihe von Algorithmen vorgeschlagen, unter anderem iterative deepening[46], direct BFS[46], intelligent search[16] und locale indices based search[46] vorgeschlagen.

In lose strukturierten Peer-to-Peer-Netzwerken, wie z.B. Freenet[8] oder Symphonie[18] bestehen keine strikten Regeln für die Zuordnung. Um das Routing zu beschleunigen, wird mit Hinweisen gearbeitet. So hat Symphonie zwar eine eindeutige Zuordnung von Objekten zu Peers, das Overlay-Netzwerk wird jedoch propabilistisch bestimmt.

Strukturierte Peer-to-Peer-Netzwerke, wie Chord[38] oder Kademlia[19], haben eine strike Zuordnung von Objekt und Peer, ebenso wie eine eindeutige Festlegung des Overlay-Netzwerkes. Es ist also möglich, gezielt nach einem Objekt zu suchen. Mit Hilfe von verteilten Hash-Tabellen ist das Auffinden von Ressourcen in diesen Netzwerken in logarithmischer Zeit möglich. Bei einer verteilten Hash-Tabelle wird die Routing-Tabelle nicht auf einem Peer oder Server gespeichert, sondern auf die Peers verteilt. Jeder Eintrag der Tabelle wird gehasht, ebenso wie jede Peer-Id. Jeder Peer ist für die Verwaltung einer Teilmenge der Einträge verantwortlich.

3.4 WEBRTC WEB REAL-TIME COMMUNICATION

Webrtc ist ein offener Standard, mit dem Echtzeitkommunikation zwischen Browser und mobilen Anwendungen ermöglicht wird. Mit Hilfe von Webrtc ist es möglich, eine direkte Verbindung zwischen Browsern aufzubauen und Daten direkt zwischen den Clients auszutauschen, ohne dass externe Plugins erforderlich sind. Webrtc ermöglicht neben dem Austausch von Video- und Audio-Daten auch die Übertragung von Binärdaten.[14] Der W3C[4] standardisiert Webrtc und definiert dafür eine Sammlung von APIs und Protokollen.

Aktuell wird Webrtc von Chrome, Firefox, Safari, Android und iOS unterstützt.[7]

3.4.1 *Verbindungsaufbau - Signalling*

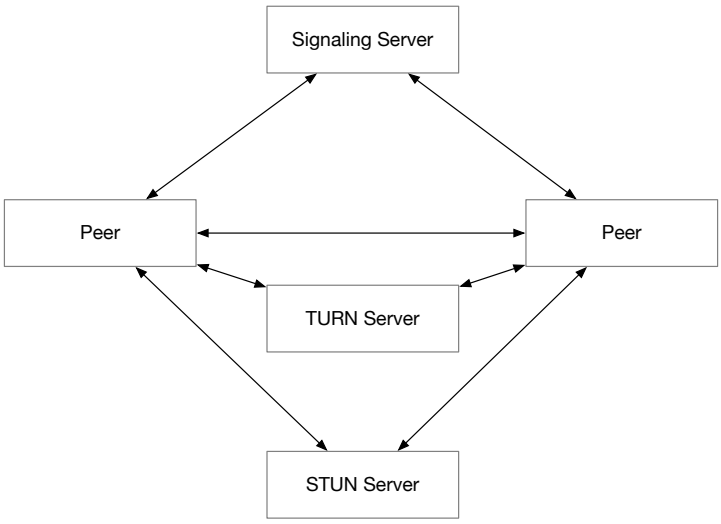

Figure 3: Überblick Server/Client Struktur Webrtc

Damit zwei Clients sich 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 Metadaten ausgetauscht, die benötigt werden, um eine erfolgreiche Webrtc-Verbindung aufzubauen. Dazu wird das SDP-Protokoll verwendet. Unter anderem werden folgende Metadaten ausgetauscht[44]:

- Session-Metadaten zum Öffnen/Schließen von Verbindungen
- Fehlernachrichten
- Metadaten über die zu übertragenden Medien (z.B. Codecs)
- Schlüsseldaten für verschlüsselte Verbindungen
- Netzwerkdaten, wie öffentliche IP-Adressen und Ports

Der Webrtc-Standard legt keine für das Signaling zu verwendende Technologie oder Protokolle fest, um so 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, dass eine bidirektionale Kommunikation zwischen den Clients notwendig ist.

3.4.2 *STUN-Server*

Da die Anzahl von IPv4 Adressen begrenzt ist, verwenden die meisten Subnetze NATs. Das hat zur Folge, dass diese Clients nicht wissen,

über welche IP-Adresse und welchen Port sie wie 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 weiterzureichen und damit auch außerhalb seines lokalen Netzwerkes erreichbar ist. Das STUN-Protokoll ist im RFC 3489[30] definiert und ist nicht auf Webrtc beschränkt.

3.4.3 *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, dass 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, sodass die Kommunikation nicht unterbrochen wird.

3.4.4 *SDP - Session Description Protocol*

```
1  v=0
2  o=Alice 1234 1234 IN IP4 host.provider1.com
3  s=Video von 987654
4  c=IN IP4 host.provider2.com
5  t=0 0
6  m=audio 20000 RTP/AVP 97
7  a=rtpmap:97 iLBC/8000
8  a=fmtp:97 mode=30
9  m=video 20001 RTP/AVP 31
10 a=rtpmap:31 H261/90000
```

Listing 1: Beispiel eines SDP Paketes

SDP[6] wurde zur Verwaltung von Kommunikationssitzungen, wie z.B. SIP entwickelt. Mit Hilfe von SDP werden beim Verbindungsaufbau wichtige Metadaten ausgetauscht. Über SDP-Pakete teilt ein Nutzer einem anderen Nutzer unter anderem verfügbare Codecs, seine IP-Adresse und den zu verwendenden Port mit. Listing 1 zeigt beispielhaft ein SDP-Paket.

3.4.5 *ICE - Interactive Connectivity Establishment*

Um den Verbindungsaufbau zwischen zwei Clients auszuhandeln, greift Webrtc auf das ICE[29] Framework zurück. Aufgrund des begrenzten IPv4-Adressraums verwenden viele Netzwerke NATs. Das hat zur Folge, dass ein Client nicht mehr direkt über seine öffentliche IP-Adresse erreichbar ist. ICE ist dafür zuständig, einen Weg zu finden zwei Peers dennoch zu verbinden. Dazu wird zuerst versucht, eine direkte Peer-to-Peer-Verbindung mit Hilfe der lokalen IP des Host-Gerätes aufzubauen. Ist dies nicht möglich, wird versucht, mit Hilfe eines STUN Servers die öffentliche IP und den öffentlichen Port zu ermitteln und über diese eine Verbindung zu initialisieren. Schlägt dies ebenfalls fehl, so wird auf einen TURN-Server zurückgegriffen, der als Mittelsmann für die Kommunikation fungiert. Alle drei Verfahren werden parallel angestoßen, um so den effektivsten Weg zu ermitteln. Die bereitgestellten Kandidaten werden nach Priorität sortiert bereitgestellt. Aus allen Kandidaten wird der mit dem geringsten Mehraufwand verwendet.

3.4.6 *Webrtc APIs*

Um eine Peer-to-Peer Verbindung zu ermöglichen stellt der Webrtc-Standard drei APIs zur Verfügung. Im folgenden wird ein grober Überblick über die APIs gegeben.

RTCPEERCONNECTION
: Das RTCPeerConnection Interface repräsentiert eine Verbindung vom lokalen Client zu 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
: Die RTCDataChannel API ermöglicht die Übertragung von Text und Binärdaten in Form von Bitstreams. Mit ihr ist es möglich die Datenformate String, Blob, Arraybuffer und Arraybuffer-View zu übertragen.[42] Um eine verschlüsselte Übertragung zu ermöglichen, werden Daten mittels SCTP (Stream Control Transmission Protocol) übertragen. SCTP ist ein verbindungsorientiertes Protokoll mit Unterstützung für Multistreaming.[43]

  RTCDataChannel stellen mehrere Übertragungsmodi zur Verfügung. Wird ein Datachannel als Reliable initialisiert, so wird garantiert, dass der Empfänger die Nachricht erhält. Ist dies nicht erforderlich, so kann ein Datachannel als Unreliable initialisiert werden. Dadurch entsteht weniger Mehraufwand und die Übertragung wird beschleunigt. Wird ein Datachannel als Ordered initialisiert, so wird nicht nur sichergestellt, dass die Pakete

empfangen wurden, sondern auch, dass sie in der richtigen Reihenfolge beim Empfänger ankommen. Es ist ebenso möglich, DataChannels als Partiallly Reliable zu definieren. Partially Reliable garantiert die Übertragung der Daten unter bestimmten Bedingungen, wie z.B. Timeouts.

MEDIASTREAM

Die MediaStream API, auch getUserMedia, ermöglicht es, in Echtzeit Daten wie Audio oder Video aufzunehmen, anzuzeigen und an andere Clients weiterzuleiten und repräsentiert Medien-Streams wie z.B. Audio- oder Video-Streams. Sie ermöglicht unter anderem den Zugriff auf Videokameras und Mikrofone. Durch sie ist es möglich auf die Hardwareunterstützung für Videos mittels OpenGL zuzugreifen. MediaStreams lassen sich mithilfe des src-Attributes von HTML 5- Videoelementen in das DOM einbinden. MediaStreams wurden von W3C in einem eigenen Standard definiert.[5]

3.5 DATACACHE-API

Die DataCache-API ermöglicht es, Netzwerk-Requests zwischenzuspeichern.[41] Ursprünglich wurde die API entwickelt, um Service Workers die Möglichkeit zu geben, einen Cache anzulegen und selbst zu verwalten. Dadurch ist es möglich, mithilfe von Service Workers und der DataCache-API Webseiten auch dann verfügbar zu machen, wenn das Internet nicht verfügbar ist.

3.6 INDEXEDDB

IndexedDB ist ein HTML 5-Feature, um Daten im Browser zu speichern. Es wurde vom W3C standardisiert[2] 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 bereitzustellen. Dabei ist der Datenzugriff auf die selbe Domain beschränkt. Die API ist überwiegend asynchron und basiert auf Promises.

3.7 SERVICE WORKER

Service Workers sind Skripte, die im Browser als separate Prozesse im Hintergrund laufen, so genannte Web Workers. Sie stellen die Funktionalitäten eines programmierbaren Netzwerk-Proxys bereit. Durch Service Workers ist es möglich, die Anfragen einer Seite zu kontrollieren, auf sie zu reagieren und in den Prozess einzugreifen.[32] Service Workers haben keinen Zugriff auf das DOM. Sie könne mehrere

Browser-Tabs verwalten. 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 festgelegt, für den der Service Worker zuständig ist. Nur Anfragen, die sich innerhalb des URL-Scopes des Service Workers befinden, können von diesem bearbeitet werden.

3.7.1 *Lebenszyklus*

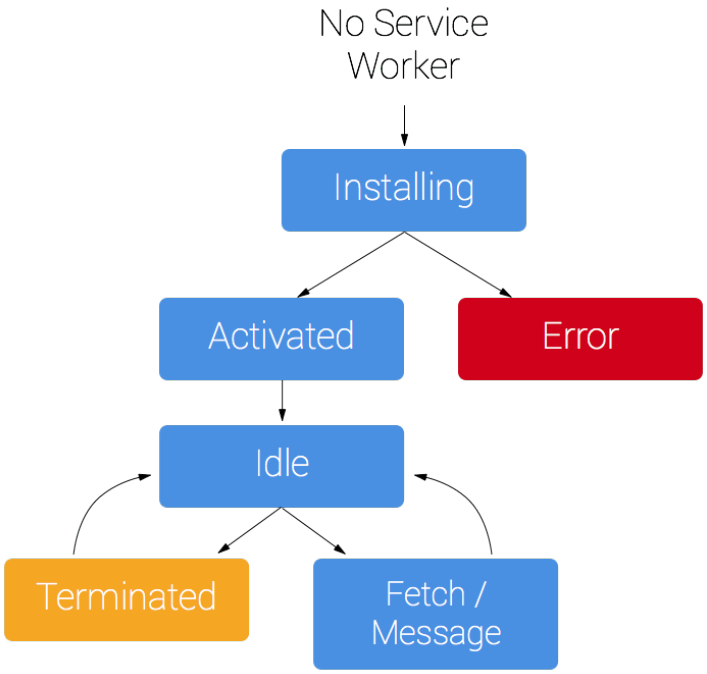

Figure 4: Lebenszyklus eines Service Workers[1]

Nachdem ein Service Worker registriert wurde, befindet er sich im Zustand der Installation. 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 zur Folge, dass ein Service Worker sich nicht auf den globalen Zustand verlassen kann, sondern stattdessen seinen Zustand auf die IndexedDb ausgelagert werden muss.

3.8 WEBSOCKETS

Websockets ist ein auf TCP basierendes Protokoll, das bidirektionale Verbindungen zwischen Server und Webanwendung ermöglicht. Zum

Table 1: Beispiel einer IP-Adresse mit Subnetzmaske

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

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. Nachdem der Client eine Websocket-Verbindung zum Server aufgebaut hat, ist es dem Server anders als bei HTTP möglich, ohne vorherige Anfrage des Clients Daten an ihn zu senden. Um eine Abwärtskompatibilität zu gewährleisten, werden ähnliche Header wie bei HTTP verwendet.

Neben dem unverschlüsseltem URI-Schema definiert das RFC6455[11] auch das verschlüsselte Schema wss. Da sehr wenig Daten-Overhead bei der Kommunikation besteht, eignen sich Websockets insbesondere für Anwendungen, die eine geringe Latenz benötigen. Websockets werden von allen modernen Browsern unterstützt.

3.9 IP-ADRESSEN

Eine IP-Adresse ist ein eindeutiger Bezeichner für Computer in einem Netzwerk. Jedem Computer in einem Netzwerk wird eine eindeutige 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. [20]

3.9.1 *Aufbau von IP-Adressen*

IPv4 wurde im Jahr 1981 durch das RFC 791[37] 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 dezimal in vier Blöcken dargestellt. IP-Adressen bestehen aus einem Netz- und einem Hostanteil. Mit dem Netzanteil wird das Teilnetz beschrieben, in dem sich das Gerät befindet, 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. 1998 entwickelte der ITFE deshalb 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 hexadezimal dargestellt. Zur Unterscheidung von Host- und Netzanteil werden Präfixlängen angegeben.

3.9.2 *Network Address Translation (NAT)*

Mithilfe von NAT können mehrere Geräte über dieselbe 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 die IP Adress Header -Felder der Datenpakete verändert.

3.10 RUBY ON RAILS

Ruby on Rails ist ein in Ruby geschriebenes Opensource-Webframework. Zentraler Ansatz der Entwicklung des Frameworks ist es, Software-Entwicklern zu erleichtern, Webanwendungen zu schreiben. Die Philosophie des Frameworks beinhaltet zwei Prinzipien[31]:

Don't Repeat Yourself(DRY): Jede Information und Funktionalität soll genau eine Repräsentation im System haben. Dadurch soll erreicht werden, dass die Software einfacher zu warten, übersichtlicher und fehlerfreier wird.

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, unterscheiden sich zu Beginn eines Projektes aber noch nicht. So werden z.B. auch Vereinbarungen über die Benennung von Methoden und Klassen getroffen (Naming Conventions).

3.11 TURBOLINKS

Turbolinks[2] ist eine Javascript-Bibliothek zur Beschleunigung der Navigation auf Internetseiten. Klickt 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 vorherigen Zustand bestehen, da kein kompletter Pageload vorgenommen werden muss.

2 https://github.com/turbolinks/turbolinks

3.12 SINGLE PAGE APPLICATIONS

Unter Single Page Applications (SPA) versteht man Webanwendungen, bei denen der Client aus einem einzigen HTML-Dokument besteht.[36] Inhalte werden meist dynamisch mit AJAX oder Websocket nachgeladen. Sie bieten sich für Anwendungen mit hoher Nutzerzahl an, da der Aufwand der HTML-Renderings vom 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.

3.13 REDIS

Bei Redis handelt es sich um einen In-Memory-Datenstrukturspeicher der als LRU-Cache, Datenbank oder Message Broker verwendet werden kann.[3] 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 könne, wobei jeder String nur einmal pro Set vorkommt. Hinzufügen von Elementen, 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, d.h., sie können jegliche Daten, z.B. auch Bilddaten, enthalten. Strings können auch als atomare Zähler verwendet werden. Dazu stellt Redis Kommandos zum Inkrementieren oder Subtrahieren von Strings bereit.

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.

3.14 HTTP LIVE-STREAMING - HLS

Beim HTTP Live-Streaming[12] werden konventionelle Webserver zur Auslieferung der Videosegmente verwendet. Das Video wird in einzelne Segmente zerlegt, die jeweils einen Teil des Videos enthalten. Diese Segmente werden über HTTP an die Nutzer verteilt. Dadurch ist es möglich, auf bestehende Verteilungsstrukturen zurückzugreifen. Indem für jedes Segment unterschiedliche Qualitäten vorgehalten werden, ist es möglich, die Qualität des Video an die Bandbreite des Nutzers anzupassen, während es abgespielt wird.

3 https://redis.io/

4

KONZEPT

In dem folgenden Kapitel wird die Netzwerkstruktur von Clients, in Schulen und bei Unternehmen-Livestreams, betrachtet und anschließend ein Peer-to-Peer-CDN vorgestellt, das auf die Anforderungen dieser Strukturen angepasst ist. (4.2) Dabei wird betrachtet und besprochen, wie Anfragen, die vom Browser gestellt werden, abgefangen werden können (4.3), wie eine direkte Verbindung zwischen Clients hergestellt werden kann (4.4), wie Ressourcen im Netzwerk gefunden werden können4.5 und wie eine Zuordnung der Clients zu Peer-Meshes erfolgen kann. (4.6) Anschließend wird erläutert wie sichergestellt wird, dass Nutzer mit älteren Browsern, die nicht alle benötigten Funktionen bereitstellen, nicht negativ beeinflusst werden. (4.7)

4.1 NETZWERKSTRUKTUREN

Im Folgenden wird betrachtet, wie sich die Netzwerkstruktur der Clients im Kontext der betrachteten Anwendungsfälle - Lernanwendungen in Schulen und unternehmensinterne Live-Streams - zusammensetzt.

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 mehrere Schüler, während sie sich im gleichen lokalen Netzwerk befinden, dieselben 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 sie die Daten nicht notwendigerweise in einem kurzen 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 an 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. 41% der Teilnehmer der in **??** untersuchten Events befanden sich während des Live Events im Firmennetzwerk. In dieser Arbeit wird betrachtet, wie die Last auf das interne Netz reduziert werden kann, ohne dass zusätzliche Caching-Server eingesetzt werden müssen.

4.1.3 *Gemeinsamkeiten*

Schul- und Unternehmensnetzwerke sind meistens so aufgebaut, dass 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 dann der Fall, wenn mehrere Clients dieselben Ressourcen benötigen. Abbildung 5 zeigt den typischen Aufbau eines solchen Netzwerks. Übersteigt die benötigte Bandbreite der Clients der durch das WAN zur Verfügung gestellten, so kommt es zu mitunter sehr teuren Netzwerkausfällen, die ganze Unternehmensstandorte betreffen können. Durch die dadurch resultierenden langen Ladezeiten kann es zu einer starken Einschränkung des Nutzererlebnisses und der Nutzerzufriedenheit kommen.[23] Um dem entgegenzuwirken, 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. Dies verursacht jedoch Kosten und Konfigurationsaufwand. Damit eignet es sich nur für größere Unternehmen, die den Service häufig nutzen.

In den betrachteten Anwendungsfällen besteht eine hohe zeitliche und inhaltliche Lokalität der Daten. Diese 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 9 zeigt exemplarisch den Aufbau eines solchen Netzwerkes. Anstatt, dass 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.

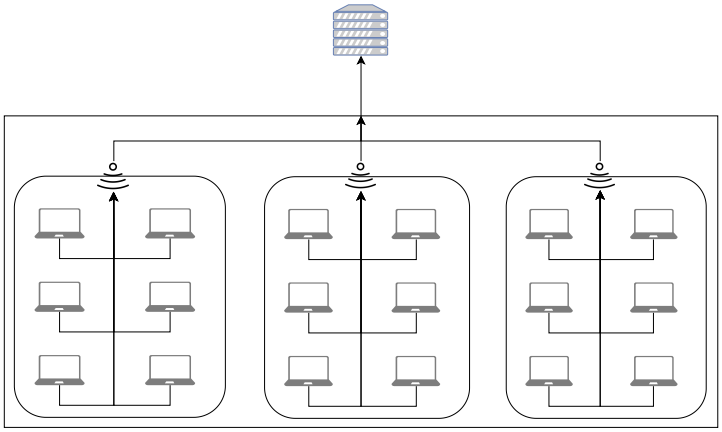

Figure 5: Netzwerkverkehr in einem herkömmlichen Netzwerk

4.2 ARCHITEKTUR

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

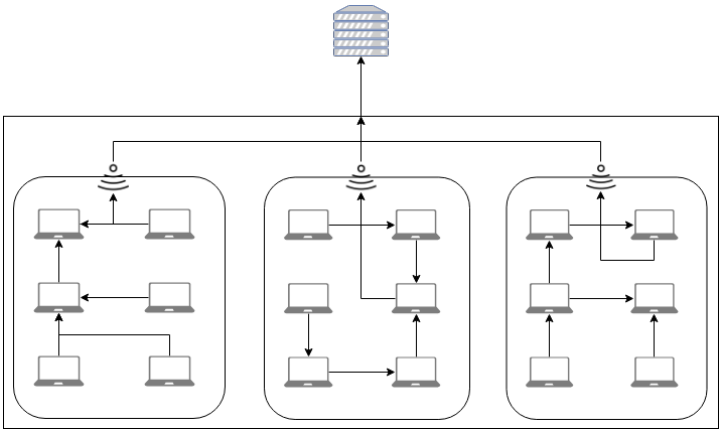

Figure 6: Netzwerkverkehr in einem Peer To Peer CDN

Da sowohl im Kontext der Schule als auch bei Unternehmen kein Wissen im Bereich der Computeradministration 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 Workers 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 domainspezifisches Wissen ausgenutzt werden, um eine

bessere Überlappung der von den Clients benötigten Ressourcen zu erreichen.

Das vorgestellte Peer-to-Peer-CDN lässt sich in drei Komponenten gliedern. Den Service Worker, der für das Abfangen und Zwischenspeichern von Anfragen zuständig ist, das Javascript-Plugin, das die Webrtc-Kommunikation übernimmt, und den Signaling Server, der für das Peer Meshing verantwortlich ist.

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 - der sogenannte Signaling Server.

Zunächst werden Informationen über die Verbindung, die aufgebaut werden soll, an den Signaling Server gesendet. Es wird ein SDP-(Session Description Protocol)-Offer gesendet. Dieses SDP-Offer leitet der Signaling Server an die Peers im selben Mesh weiter. Geantwortet wird mit einer SDP-Answer, welche Informationen über die abgestimmte Verbindung enthält und über den Signaling Server zurückgeleitet 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 den 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.

4.3 JAVASCRIPT-PROXYS - ABFANGEN VON ANFRAGEN

Damit ein Clientseitiges CDN möglich ist, ist es notwendig, dass die Abfragen des Browsers abgefangen und auf anderem Weg beantwortet werden können. Nachdem der Browser nach einer Anfrage die URL

aufgelöst hat (DNS Lookup) lädt er die abgefragte Seite. Ist die Seite geladen, beginnt der Browser die im HTML-Dokument verlinkten Dateien 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.

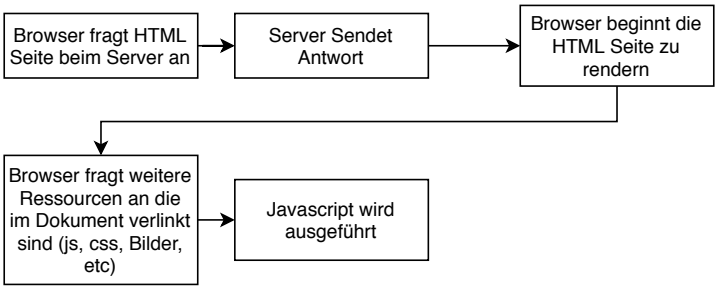

Figure 7: Ablauf einer HTML abfrage im Browser

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-Pageload 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-Skript geladen wurde, mit Ajax nachzuladen. Dadurch lässt sich mit Javascript kontrollieren, von wo die Ressource geladen werden soll. Allerdings können Ressourcen, die über das Peer-to-Peer-CDN geladen werden sollen, erst dann geladen werden, wenn das komplette HTML-Dokument und das CDN-Skript 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, nachdem das Event ausgelöst wurde, so wird dieser Code nicht mehr ausgeführt.

Service Workers 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

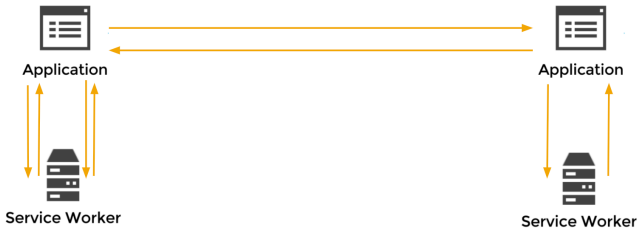

Figure 8: Service Worker - Webrtc

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 weiterzuleiten. 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 Node.js und SlideSync in Ruby on Rails programmiert sind, bietet 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 Node.js und Ruby an.

Tritt ein Client einem Peer Mesh bei, so sendet er eine Nachricht mit seiner eigenen Peer-Id 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 10 zeigt den Verbindungsaufbau zwischen zwei Peers. Nachdem Peer1 über den Websocket-Channel mitgeteilt hat, dass er dem Peer Mesh beitreten will, sendet Peer2 ein SDP-Offer über einen

1 https://faye.jcoglan.com/

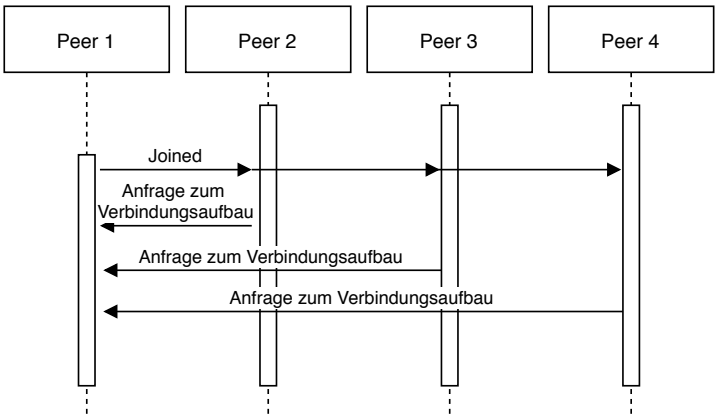

Figure 9: Signaling Ablauf

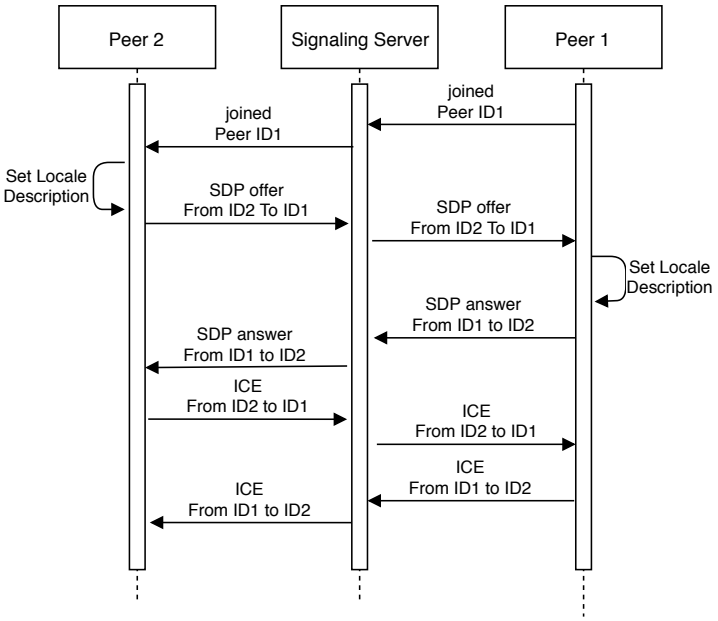

Figure 10: Ablauf Verbindungsaufbau

Websocket-Channel an 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 eine SDP-Answer zurück. Mit den dadurch erhaltenen Informationen tauschen die beiden Clients nun ICE-Pakete über den Websocket-Channel aus. Im Anschluss können beide Clients über Webrtc direkt miteinander kommunizieren.

Verlässt ein Client das Peer-to-Peer-Netzwerk, so müssen die anderen Clients darauf reagieren und ihn aus ihrer Liste von Peers löschen. Zwar wäre es möglich, im Fall, dass ein Client die Seite verlässt, eine Nachricht zu senden, z.B. mittels dem Javascript Event onbeforeunload[2], dies ist jedoch sehr unzuverlässig. Im Fall, dass der Client z.B. die Internetverbindung verliert oder der Computer aus-

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

geschaltet 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, wird der Peer aus dem Netzwerk entfernt. Auf diesem Weg können fehlerhafte Peers entfernt werden, ohne darauf angewiesen zu sein, im Fehlerfall noch in der Lage zu sein, 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 und während der Ladezeit der Seite sich bei den betrachteten Anwendungsfällen gut vorhersagen lässt, welche Ressourcen benötigt werden, erscheint es sinnvoll, das Routing im Vorfeld geschehen zu lassen. Zwar skalieren Algorithmen wie Kademlia [19] sehr gut für eine große Anzahl an Teilnehmern, jedoch ist zum Zeitpunkt 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, müssen sämtliche Verbindungen im Moment des Routings aufgebaut werden. Für das Beispiel mit 1000 Teilnehmern hieße es, dass drei 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 hält in dem vorgeschlagenen CDN jeder Teilnehmer eine Hashtabelle 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 ZUORDNUNG ZUM MESH-NETZ

Als Netz-Topologie wurde ein vollvermaschtes 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 teilvermaschten 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 kann, geringer. Dadurch ist es besonders wichtig, die Teilnehmer sinnvoll auf die Meshes zu verteilen, wobei sinnvoll bedeutet, dass sie eine möglichst große Schnittmenge an gemeinsamen Ressourcen haben. Da dies sehr domänenspezifisch 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*

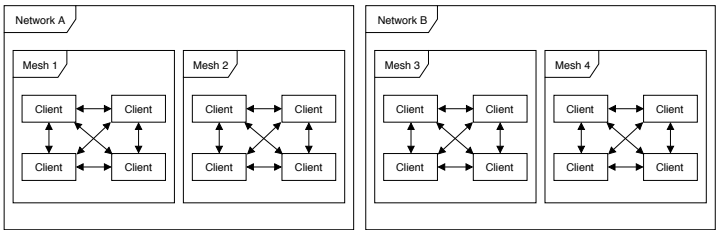

Figure 11: Peer to Peer Meshes - SlideSync

SlideSync ist eine Plattform, deren Nutzung stark durch die durchgeführten Live-Events dominiert wird. Ein Moderator erstellt das Event und 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.

Die Peer Meshes in SlideSync werden als vollvermaschte Netzte abgebildet. Da alle Teilnehmer eines Events zu großen Teilen dieselben Daten benötigen, können sie im selben Mesh untergebracht werden. Um zu gewährleisten, dass sich die Peers im selben Subnetz befinden, teilen sich nur solche ein Peer-Mesh die sich in derselben IP-Range befinden. Ein weiterer wichtiger Faktor ist der Kommunikationsmehraufwand der durch das Aufrechterhalten von Verbindungen zu vielen Peers entsteht. Deshalb ist es nicht möglich, bei größeren Events alle Peers im selben Peer-Mesh unterzubringen. Deshalb werden Sub-Meshes gebildet, in denen sich eine maximale Anzahl an Peers befinden können.

Abbildung 11 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, dass ein Peer sich auch aktiv am Mesh beteiligen kann, sendet er regelmäßig Ping-Nachrichten an den Server. Da dieser

Mechanismus in SlideSync schon zuvor 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, dass 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 Hintergrundjob gestartet, der alle Peers, 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 Hintergrundjob 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.

4.6.2 *Schul-Cloud*

Bei der Schul-Cloud erfolgt die Aufteilung der Nutzer anhand von Klassen. Alle Schüler, die in derselben Klasse sind, werden demselben 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 aus 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 manuelle 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, nicht mit Schülern innerhalb des Schulnetzwerks über Webrtc verbinden und wird als möglicher Peer aussortiert.

4.6.3 *Subnetzerkennung*

Um sicherzustellen, dass 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 desselben lokalen Netzwerkes. Dies hat den Nachteil, dass 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, dass bereits bei der Einteilung in 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 NICHT UNTERSTÜTZTEN 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 Systemtest(siehe **??**) verwendet. Wird der Browser nicht unterstützt so wird die Initialisierung des Skripts abgebrochen. Kann eine Verbindung zu einem Teilnehmer nicht aufgebaut werden oder bricht sie ab, so wird er aus der eigenen Liste der Teilnehmer gelöscht, sodass 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 Netzwerkeinstellungen dies nicht zulassen, so werden Anfragen so beantwortet, als wäre das CDN nicht vorhanden.

5

IMPLEMENTIERUNG

Das folgende Kapitel stellt Details zur Implementierung und zum Anwendungsdesign vor. Dabei wird in 5.1 auf die verwendeten Technologien eingegangen. 5.2 stellt die Architektur der Implementierung vor. In Kapitel 5.3 wird genauer auf die Realisierung des in 4.6 vorgestellten Peer Meshing und auf das in 4.4 beschriebene Signaling eingegangen. Da für die Funktion des CDNs eine größere Anzahl von Nachrichten nötig ist, wird in 5.5 näher auf das verwendete Nachrichtenprotokoll eingegangen. Kapitel 5.6 erläutert die notwendige Implementierung von Request Chunking. Da es unerlässlich ist, Statistiken zu erheben, um das CDN zu evaluieren, beschreibt Kapitel 5.8, wie das CDN Echtzeitsstatistiken erfasst und diese an einen Server sendet. Im Anschluss (5.10) wird darauf eingegangen, wie das CDN und das Netzwerk konfiguriert werden können, um die Funktion des CDNs zu gewährleisten.

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 -Skript, und das Peer-to-Peer CDN Client Script. Beide Skripte müssen von der Anwendung eingebunden werden. Der Server muss einen Faye Websocket -Server bereitstellen. Da das Peer Meshing über die Konfiguration des Faye Channels geschieht, muss der Server gegebenenfalls eine Peer Meshing-Strategie implementieren. Zur einfacheren Einbindung in bestehende Projekte werden ein Node.js-Modul sowie ein Ruby Gem bereitgestellt. Das Ruby Gem bindet das Service Worker -Skript im Public Ordner der Ruby on Rails -Anwendung ein und das Peer-to-Peer CDN-Skript im vendor/assets Ordner.

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

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

5.2 ARCHITEKTUR

Im Folgenden wird die Software-Architektur des Peer-to-Peer-CDNs beschrieben. Dazu wird genauer auf das clientseitige Skript sowie den Service Worker eingegangen.

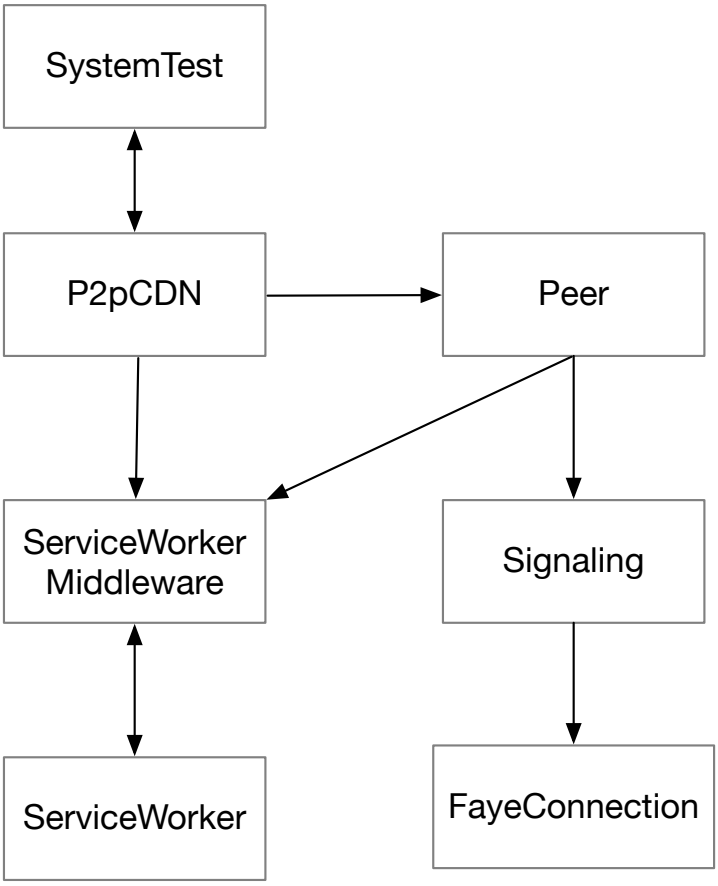

Figure 12: Klassendiagramm

5.2.1 *Client-Skript*

Das Javascript-Plugin besteht aus sieben Komponenten. Die Klasse P2pCDN bildet die Schnittstelle nach außen und stellt Funktionalitäten zum Initialisieren und Konfigurieren des CDN 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 Client in dem Peer-to-Peer-Netzwerk. Es hält den eigenen Zustand und verwaltet die Verbindungen zu anderen Clients und verwaltet deren Ressourcen-Hashes. Das Peer Objekt nutzt Funktionalitäten des Signaling-Moduls um Verbindungen zu anderen Clients aufzunehmen. Das Signaling-Modul implementiert das Webrtc Signaling Protokoll. FayeConnection abstrahiert die beim Signaling verwendete Websocket-Bibliothek, in diesem Fall Faye, um

es zu ermöglichen, auch andere Websocket-Bibliotheken zu verwenden. Die Service Worker-Middleware fungiert als Vermittler zwischen Service Worker und Client-Script und ist für die Initialisierung des Service Workers zuständig. Die Komponente arbeitet eventbasiert und registriert sich auf folgende Events:

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 Event peer:onAddedResource aus, um den Peer über die Änderung zu informieren, der das Update an die verbundenen Clients weiterleitet.

SW:ONREQUESTCACHE
: Stellt der Peer eine neue Verbindung zu einem anderen Client her, 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 Event sw:onRequestCache mit einem Callback, der bei Erfolg ausgeführt wird. Die ServiceWorker-Middleware leitet die Anfrage über den Message Channel an den Service Worker weiter.

5.2.2 *Service Worker*

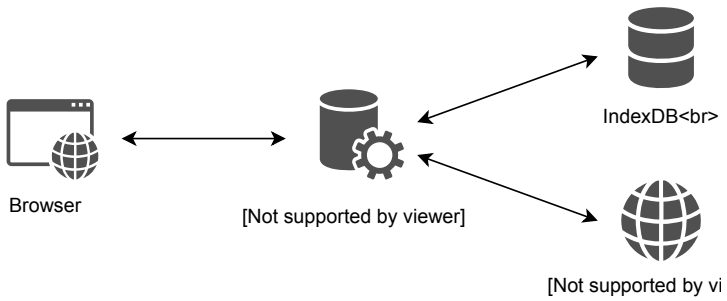

Figure 13: Service Worker

diagram fixen

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 zunächst 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, lädt er sie vom Server. Um eine Anfrage über das Peer-to-Peer-Netzwerk zu beantworten, muss zuvor das Client-Skript geladen sein, da Service Workers die Webrtc-API nicht unterstützen. Die Kommunikation zwischen Service Worker und Anwendungs-Skript geschieht mit Hilfe der postMessage-API[3]. Um sicherzustellen, dass das Client-Skript geladen und zur Kommunikation bereit ist, sendet der Service Worker eine nicht blockende Heartbeat-Anfrage an das Skript. Antwortet das Skript nicht innerhalb eines bestimmten Zeitfensters, so muss der Service Worker davon ausgehen, dass es noch nicht geladen ist. Die Anfrage wird an den Server weitergeleitet. Da nicht davon ausgegangen werden kann, dass das Skript verfügbar bleibt nachdem es geladen wurde, die Seite könnte z.B. geschlossen worden sein, muss der Service Worker sich vor jeder blockenden Anfrage vergewissern dass das Script zu antworten in der Lage ist. Nachrichten, die zwischen Anwendungs-Skript und Service Worker ausgetauscht werden, haben das Format: { type: "Art der Nachricht", msg: "Inhalt der Nachricht" }. Das Attribut "type" kann folgende Werte enthalten: "resource", "cache", "heartbeat". Nachrichten vom Typ "resource" ordnen den Service Worker an, eine Nachricht aus seinem Cache an das Anwendungs-Skript zu übergeben. Empfängt 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 gecacheten Ressourcen mitzuteilen. Mit dem Typ "heartbeat" erfragt der Service Worker beim Anwendungs-Skript, ob es momentan verfügbar ist. Ist das Skript noch nicht verfügbar, hat der Service Worker zwei Möglichkeiten damit umzugehen. Er kann darauf warten, dass das Skript 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, wird die Ladezeit der Ressource nicht verzögert, jedoch ist es erst möglich Ressourcen über das Peer-to-Peer-CDN zu laden, wenn das Skript 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 CDN dafür entschieden, dass in diesem Fall Anfragen direkt an den Server weitergeleitet werden sollen. Das hat zur Folge, dass sich die gewählte Implementierung besonders für Single Page Applications sowie für

format formatieren

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

Anwendungen, die Navigationen z.B. mithilfe von Turbolinks vermeiden, eignet. Bleibt beim Laden von neuen Inhalten das Anwendung-Skript 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-Skript 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, unterbricht er die Verarbeitung und die Anfrage wird vom Server beantwortet. Sobald der Service Worker aktiv ist, ruft er self.clients.claim() auf, um sicherzustellen, dass er schon beim ersten Seitenaufruf aktiv ist und Anfragen der Clients verarbeiten kann.

5.3 SIGNALING SERVER

Der Signaling Server nutzt Faye[4] als Websocket-Bibliothek. Die Anwendung muss lediglich einen Faye-Server anbieten, um das Signaling zu ermöglichen. Der Verbindungsaufbau wird im Anschluss clientseitig gehandhabt. Um Clients den Peer Meshes zuzuordnen, werden verschiedene Websocket-Channels verwendet. Wird zwei Clients in der Konfiguration derselbe Datachannel 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 domänenspezifisches Peer Meshing vorzunehmen.

5.3.1 *SlideSync*

Um das Signaling zu realisieren, implementiert SlideSync eine eigene Klasse, die für die Zuordnung von Clients zu Peer Meshes verantwortlich ist. Der Klasse wird der Scope des Aufrufs sowie die maximale Mesh Größe bei der Initialisierung übergeben, wobei der Scope die ID des aufgerufenen Events ist. Durch Aufruf der Methode join() 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 skalierten Multi Server Setup

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

funktionieren. Um dies zu erreichen, wird Redis als Datenspeicher verwendet. Redis bietet geeignete 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 Hintergrundjob 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, ergibt 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.

Schreibweise: ID

5.3.2 *Schul-Cloud*

Die Schul-Cloud stellt ebenfalls einen Faye-Server bereit, über den das Signaling gehandhabt wird. Als Scope für die Peer Meshes wird die Schulklasse der Schüler verwendet. Da Schulklassen nur eine begrenzte Größe haben, ist eine weitere Unterteilung nicht notwendig.

5.4 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 sowie deren Ressourcen hat.

5.5 NACHRICHTENPROTOKOLL

Für die Funktion des CDNs sind einige Nachrichten zwischen den Komponenten und den Clients notwendig. Im Folgenden wird auf einige dieser Nachrichten genauer eingegangen, sowie der Kommunikationsablauf zweier Clients anhand eines Beispiels erläutert.

5.5.1 *Beispielhafter Kommunikationsfluss zwischen zwei Clients*

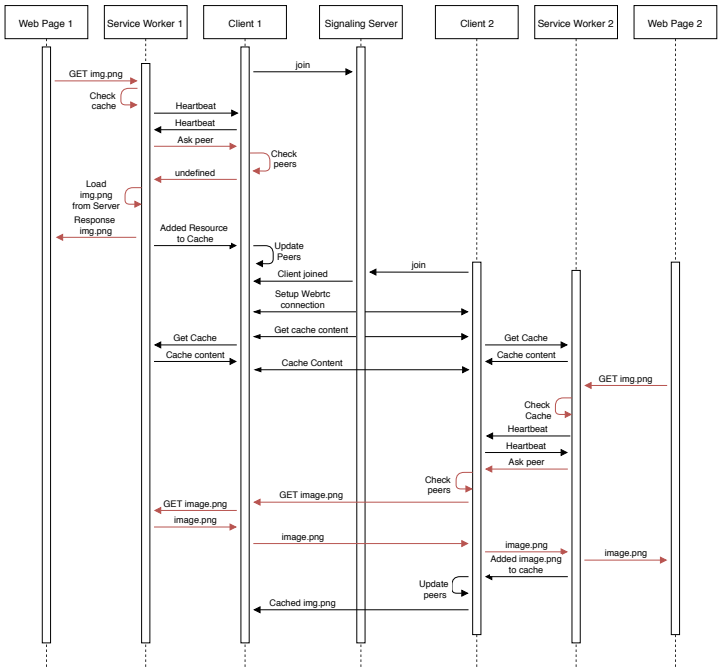

Figure 14: Kommunikationsfluss zwischen zwei Clients um eine Ressource auszutauschen

Abbildung 14 zeigt eine vereinfachte Darstellung des Kommunikationsflusses zweier Clients, welche die Ressource img.png laden. Client 1 lädt als erster die Webseite, initialisiert das CDN und meldet dem Signaling Server zurück, dass er dem Peer

Mesh beigetreten ist. Die Website versucht anschießend, die Ressource img.png zu laden. Diese Anfrage wird vom Service Worker abgefangen, welcher als erstes versucht, die Anfrage über seinen eigenen Cache zu beantworten. Da dies nicht möglich ist, versucht er nun, die Anfrage über das Peer-to-Peer-Netzwerk zu bearbeiten. Dazu muss er das Client Script auffordern, bei geeigneten Peers nach der Ressource zu fragen. Da diese Anfrage an die Client blockend ist muss er sich zuerst über einen Heartbeat versichern das das Client Script auch verfügbar ist. Zur Übersichtlichkeit des Diagrams wurde der Heartbeat nicht bei jeder Kommunikation hinzu gefügt. Hat der Service Worker sich vergewissert, dass das Client Script in der Lage ist zu antworten, so sendet er die Anfrage der Ressource image.png an das Client Script. Dieses sucht in der lokalen List von Peers und deren Ressourcen nach einem geeigneten Peer. Dieser ist in diesem Fall nicht vorhanden, weshalb es an den Service Worker undefined sendet. Daraufhin lädt der Service Worker die Ressource über den Server herunter. Anschließend speichert er die Ressource in seinem Cache.

Schreibweise:Skript

Als nächstes lädt Client 2 die Webseite und teilt dem Signaling Server mit, dass er dem Peer Mesh beitreten will. Dieser sendet eine Nachricht an Client 1, dass ein neuer Peer dem Mesh beigetreten ist. Client 1 initialisiert daraufhin die Webrtc-Verbindung zu Client 2. Ist die Verbindung hergestellt, tauschen die beiden Clients den Inhalt ihres Caches aus und speichern, welche Ressourcen der andere Client gespeichert hat. Nun versucht die Webseite von Client 2, die Ressource img.png zu laden. Der Service Worker von Client 2 fängt die Anfrage ab. Da er die Anfrage nicht mit Hilfe seines Caches beantworten kann, beauftragt er Client Script 2 mit der Beantwortung der Anfrage über das Peer-to-Peer-Netzwerk. Client 2 ermittelt Client 1 als geeigneten Peer und sendet die Anfrage der Ressource img.png an Client 1 weiter. Client 1 sendet nun die Anfrage an seinen Service Worker weiter. Dieser lädt die Ressource aus seinem Cachen und leitet sie an Client 1 weiter. Client 1 serialisiert und unterteilt, falls nötig, die Ressource in Chunks und sendet sie an Client 2. Client 2 setzt die Antwort nun wieder zusammen, deserialisiert sie und sendet die Antwort an seinen Service Worker. Der Service Worker sendet das Response Object an die Webseite und das Bild kann geladen werden.

img.png falsch

5.5.2 *Heartbeats*

Sendet der Service Worker eine Anfrage an das Client Script, so muss dieser sicher stellen, dass das Script sowohl geladen ist

als auch auf den Service Worker Message Channel registriert ist. Zwar ist auch die eigentliche Anfrage an das Client Script mit einem Timeout versehen, jedoch ist dieser deutlich höher als der des Heartbeats. Dieser Heartbeat muss zwischen jeder Kommunikation zwischen Service Worker und Client Script, die blockt, durchgeführt werden, da es möglich ist, dass zwischen der aktuellen und der letzten Anfrage das Client Script beendet wurde oder nicht mehr in der Lage ist, auf Anfragen des Service Workers zu reagieren. Gründe dafür können unter anderem sein, dass ein Fehler geworfen oder dass die Seite geschlossen wurde. Wird der Heartbeat weggelassen, so muss im Falle eines nicht antwortenden Skripts immer die maximale Timeout-Dauer für Anfragen über das Peer-to-Peer-CDN abgewartet werden. Als Folge würden sich diese Anfragen deutlich verlangsamen.

5.5.3 *Verbindungsaufbau*

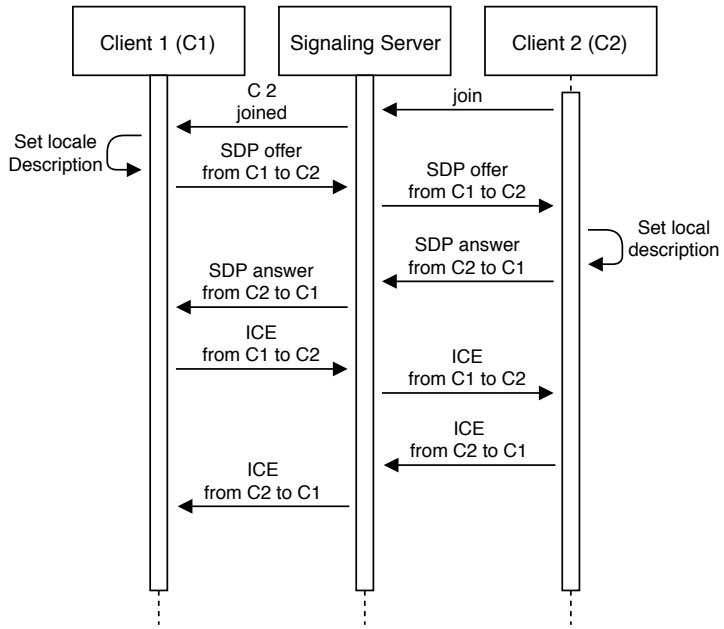

Figure 15: Flussdiagramm Verbindungsaufbau

Tritt ein Client einem Peer Mesh bei, so muss er zuerst eine Verbindung zu allen Clients aufbauen, die sich bereits im Peer Mesh befinden. Dazu sendet er eine Nachricht an den Signaling Server, der diese an alle Peers weiterleitet. Empfängt ein Peer die Nachricht, dass ein neuer Peer dem Netzwerk beigetreten ist, so beginnt er eine Webrtc-Verbindung zu dem Peer aufzubauen. Dazu sendet er über den Signaling Server ein SDP-Offer an den Peer, der mit einer SDP-Answer-Nachricht antwortet. Empfängt ein Peer in 15 C1 eine SDP Answer, so sendet er ein ICE-Paket an

den Peer, von dem er es erhalten hat (C2). Dieser antwortet ebenfalls mit einem ICE-Paket und ein DataChannel wird eröffnet. Sämtliche weitere Kommunikation zwischen den beiden Peers wird über den DataChannel gehandhabt.

5.5.4 *Cache-Initialisierung*

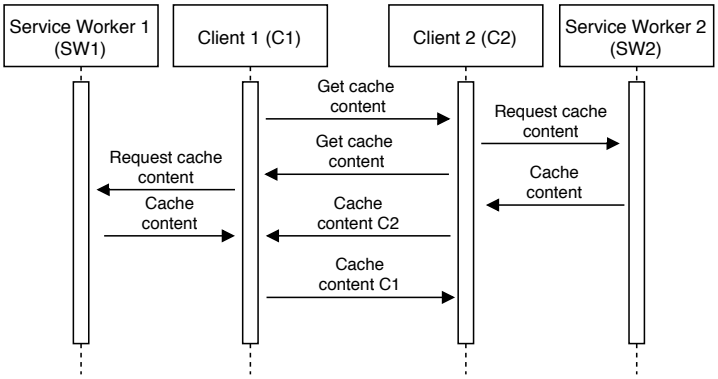

Figure 16: Flussdiagramm Cache-Initialisierung

Nachdem eine Webrtc-Verbindung zwischen den beiden Clients hergestellt ist, müssen Informationen über den Inhalt der Caches ausgetauscht werden. Dazu senden sich die Clients gegenseitig die Anfrage nach dem Inhalt des Caches. Beide Clients fragen nun ihren Service Worker, welche Ressourcen bereits in ihrem Cache vorhanden sind und senden die Liste der zwischengespeicherten Ressourcen an den Client zurück x

5.5.5 *Updates*

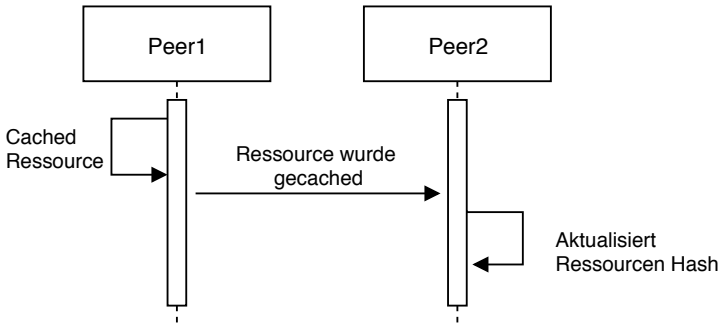

Figure 17: Flussdiagramm Ressourcen-Update

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.

Verlässt ein Client das Peer Mesh, so wird er aus der Liste der verfügbaren Peers gelöscht. Anschließend werden alle offenen Kommunikationen zu dem Peer entfernt. Ein Peer wird als nicht mehr verbunden betrachtet, wenn der DataChannel nicht mehr offen ist.

5.6 CHUNKING

Für den Austausch von Daten werden die von WebRTC angebotenen DataChannel genutzt, welche das Stream Control Transmission Protocol, kurz SCTP, verwenden. Problem hierbei ist, dass dieses Protokoll ursprünglich für die Übertragung von Kontrollinformationen designed 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.

```
1  if(peer.dataChannel.bufferedAmount <= 16000000) {
2    peer.dataChannel.send(msg);
3    return;
4  }
5  // if maximum buffersize is reached delay sending of chunks
6  peer.requestQueue.push(msg);
7  peer.dataChannel.bufferedAmountLowThreshold = 65536;
8  peer.dataChannel.onbufferedamountlow = function () {
9    var reqs = peer.requestQueue.slice();
10   peer.requestQueue = [];
11   reqs.forEach(_msg => send(_msg));
12 }
```

Listing 2: Buffersize Berücksichtigung

Datachannel haben in der aktuellen Browser-Implementation eine maximale Buffergröße. Der Client, der die Datachannel verwendet, ist dafür verantwortlich sicherzustellen, 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 zusammensetzen.

```
_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: Berücksichtigung der Buffersize

Dazu wird für die Anfrage ein Array verwaltet, in dem alle empfangenen Chunks gespeichert werden. Wurden alle Chunks einer Anfrage empfangen, werden sie wieder zu einem Objekt zusammengesetzt und die Anfrage wird an den Service Worker weitergeleitet.

5.7 SYSTEMTEST

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[5] 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()

5 https://modernizr.com/

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 anderer Client im aktuellen Peer Mesh befinden.

5.8 ERFASSEN VON STATISTIKEN

```
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

Um die Nutzungsstatistiken des CDN 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 be-

```
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

deutet, dass 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 serverResponse 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 6)

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 7) und endend nachdem 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 Beantwortung des Requests (siehe Evaluation).

Mit Hilfe der Konfiguration 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

```
_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 6

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.

5.9 QUOTA LIMITS - LÖSCHEN VON REQUESTS AUS DEM CACHE

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

Table 2: Browser Storage Quotas[24]

Browser stellen den Clients unterschiedlich viel Speicherplatz für Offline-Caches zur Verfügung. Ist das Quota limit erreicht, versuchen Firefox und Chrome Speicher freizumachen, 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 lediglich einen

```
getFromCache(hash).then(cacheResponse => {
  if (cacheResponse && config.cachingEnabled) {
    resolve(cacheResponse);
    return;
  }
  getFromClient(clientId, hash).then(data => {
    if (data && data.response) {
      resolve(data.response);
      return;
    }
    getFromInternet(url).then(response => {
      resolve(response);
    });
  });
});
```

Listing 7: Abarbeitung eines Request im Service Worker

Fehler.[24] Deshalb ist es notwendig, dass der Service Worker, wenn das Limit erreicht wird, Elemente löscht.

Mit Hilfe der Quota Management API[27] ist es möglich, die momentane Speichernutzung auszulesen, ebenso wie den maximal verfügbaren Speicherplatz. Ist dieses 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.

5.10 KONFIGURATION

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 in IndexedDB gespeichert und von dort durch den Service Worker geladen.8 zeigt eine beispielhafte 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 demselben Channel befinden sich im selben Peer Mesh.

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

Listing 8: Beispielhafte Konfiguration

CLIENTID
: Eindeutiger Bezeichner, mit dem der Peer identifiziert werden kann. Ähnlich einer Session ID wird er verwendet, um Clients wiederzuerkennen und anzusprechen.

IDLENGTH
: Bezeichnet die maximale Länge der ClientID. 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 5.9)

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 URLs, die mit Hilfe des CDN bearbeitet werden sollen. URL können als Regex definiert werden.

EXCLUDEDURLS
: Liste von URLs, die explizit von dem CDN ausgeschlossen werden sollen. Ebenso wie urlsToShare werden excludedUrls als Regex interpretiert.

PATH
: 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 5.9).

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 5.8).

5.10.1 *Netzwerkkonfiguration*

Um eine fehlerfreie Funktion des CDNs zu gewährleisten, muss sichergestellt sein, dass sich die Clients miteinander verbinden können. Insbesondere in Unternehmensnetzwerken kann dies ein Problem darstellen. Häufig sind in diesen Netzwerken strenge Firewallregeln aktiv, die nur die Kommunikation über bestimmte Ports erlauben. Da WebRTC eine relativ große Range an Ports verwendet, aus der zufällig ein freier ausgewählt wird, ist es nicht immer eine Option, alle Ports explizit zu öffnen. Wird

eine Firewall verwendet, die nur den ein- und ausgehenden Datenverkehr überwacht, so können weiterhin Verbindungen innerhalb des lokalen Netzwerkes hergestellt werden. Das im Rahmen dieser Arbeit vorgestellte CDN ist damit weiterhin in der Lage, Inhalte unter den Nutzern zu verteilen. Ein größeres Problem stellen hier clientseitige Firewalls dar. Zwar könnte theoretisch der Datenverkehr über TURN-Server geleitet werden, jedoch wird in diesem Fall der gesamte Verkehr über einen Server geleitet, der sich unter Umständen nicht im selben lokalen Netzwerk befindet. Das CDN ist in diesem Fall nicht in der Lage, die Belastung des Netzwerkes zu reduzieren. In diesem Fall bleibt nur die Möglichkeit, die verwendeten Ports clientseitig zu öffnen. Chrome bietet die Möglichkeit, die für Webrtc verwendeten Ports einzustellen.[25]

Soll eine Kommunikation zwischen Nutzern möglich sein, die sich nicht im selben lokalen Netzwerk befinden, so muss in den meisten Netzwerken aufgrund von NAT ein STUN-Server spezifiziert werden.

6

EVALUATION

Um die Effektivität und Funktion des CDNs zu evaluieren, wurden simulierte Tests sowie Tests unter echten Bedingungen durchgeführt. Neben der Betrachtung der Effektivität, ist es ebenfalls wichtig sicherzustellen, dass das CDN von den Nutzern verwendet werden kann. Dazu wurden die verwendeten Browser der Nutzer evaluiert und betrachtet, wie viele Nutzer einen geeigneten Browser verwenden. Da die Internetanbindung an Schulen in vielen Fällen nicht gegeben ist, wird erläutert, wie eine reine Offline-Nutzung des CDNs gewährleistet werden könnte. Eine Betrachtung der Implikationen für die Sicherheit bei der Verwendung des CDNs befindet sich am Ende des Kapitels.

6.1 BROWSERKOMPATIBILITÄT

Chrome	Firefox	Edge	Safari	Internet Explorer
22	23	76	11	Not supported

IOS Safari	Samsung Internet	Android Browser	Chrome (Android)
11	4	67	76

Table 3: Unterstützte Browserversionen

Um die unterstützten Browser zu verifizieren, wurden die vom Peer-to-Peer-CDN verwendeten Browserfunktionalitäten bei caniuse.com[1] eingegeben. Darüber hinaus wurden Chrome, Firefox, Edge, Safari und der Internet Explorer manuell getestet, um sicherzustellen, dass das Peer-to-Peer-CDN das Nutzererlebnis nicht negativ beeinflusst. Der Internet Explorer unterstützt 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

1 https://caniuse.com/
2 https://caniuse.com/search=service%20worker

Datenverkehr des Peer-to-Peer-CDNs über Datachannel gehandhabt wird, ist eine Verwendung erst ab der nächsten Version, die auf Chrome aufbauen wird, möglich.

6.1.1 *Browsernutzung von SlideSync-Kunden*

Event	Intern	Extern	Unterstützt	Nicht unterstützt
Event 1	2816	4073	4357	2532
Event 2	50	48	37	60
Event 3	43	248	224	67
Event 4	740	569	735	573
Event 5	300	520	600	220
Event 6	365	656	773	248
Gesamt	4314	6114	6726	3700

Table 4: Vergleich von 6 Events

Um die Browsernutzung von SlideSync-Nutzern zu evaluieren, wurden je drei Events von zwei Kunden betrachtet. Beide Kunden verwenden momentan einen von SlideSync vertriebenen internen Caching-Server und mit beiden Kunden wird derzeit eine Verwendung des Peer-to-Peer-CDNs anstelle der momentanen CDN-Server diskutiert. Event 1 bis 3 wurden von Kunde 1, und Event 4 bis 6 wurden von Kunde 2 durchgeführt.

Von den insgesamt 10426 betrachteten Browser Sitzungen verfügten 64,5% über einen Browser, von dem Peer-to-Peer-CDN unterstützt wird. Die vergleichsweise geringe Prozentzahl ist vor allem auf die nach wie vor starke Verbreitung des Internet Explorers in Unternehmen zurückzuführen. So handelte es sich bei 96,5% der nicht unterstützten Browser von Event 1 um Nutzer mit Internet Explorer 11. In Gesprächen mit Kunden wurde deutlich, dass diese jedoch in absehbarer Zeit auf Edge 76 wechseln werden, welcher auf Chrome basiert und damit volle Unterstützung des CDNs bietet.[21] Ein Mitarbeiter einer großen Bank gab einen Zeithorizont von ein bis zwei Jahren an. Außerdem würden sie als Bank Updates eher hinauszögern, weshalb bei anderen Unternehmen ein kürzerer Zeithorizont zu erwarten sei.

6.1.2 *Browser usage in educational networks*

Browser usage schul-cloud schreiben

- evtl einfach normale verteilung

- schauen ob es quellen gibt

6.2 SIMULIERTER WORKLOAD

Im Folgenden werden verschiedene Tests zur Evaluation des Peer-to-Peer-CDNs vorgestellt und erläutert. Um zu untersuchen, wie sich ‚das CDN unter verschiedenen Bedingungen verhält wurde die Ladezeit verschieden großer Dateien gemessen, Live-Streams mit simulierten Nutzern getestet und eine Schulklasse simuliert. Um möglichst viele Variablen kontrollieren zu können, wurden die Tests unter Laborbedingungen durchgeführt und alle Nutzer simuliert.

6.2.1 *Ladezeiten von verschiedenen Dateigrößen*

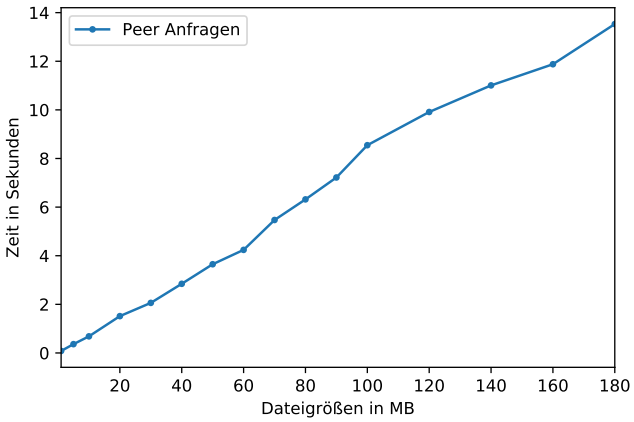

Figure 18: Vergleich verschiedener Dateigrößen

Um die Ladezeiten verschiedener Dateigrößen zu evaluieren wurde auf einem Macbook Pro Late 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. Alle tests wurden zehn Mal ausgeführt. Das Peer-to-Peer Datenpackete wurden über das lokale Netzwerk geleitet.

Leerstelle vor Einheiten

Abbildung 18 zeigt, wie sich das CDN bei steigender Dateigröße verhält. Es ist deutlich zu sehen, dass die Ladezeit bei steigender Größe linear ansteigt. Zwar schwankt der Durchsatz des CDNs in einem Bereich von 20 MBits/s, jedoch scheint es keinen Zusammenhang zur Größe der Datei zu geben. Das Chunking der Dateien durch das Peer-to-Peer-CDN scheint keinen nega-

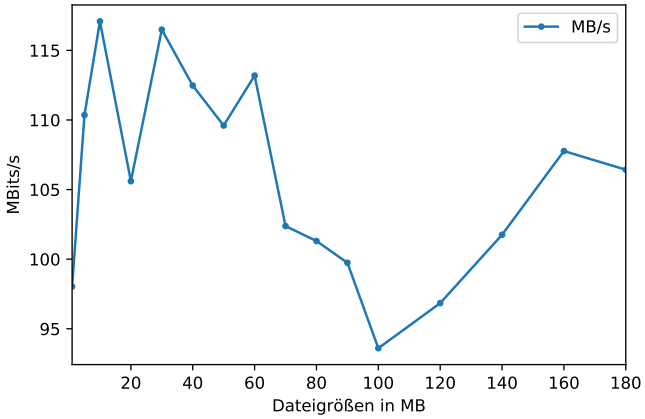

Figure 19: Durchsatz des CDNs in einem lokalen Netzwerk

tiven Einfluss auf die Ladegeschwindigkeit zu haben. Im Bereich von 100 MBits/s ist nach wie vor die zur Verfügung stehende Bandbreite und nicht das Chunking als Bottleneck zu betrachten.

6.2.2 *Live-Streaming*

Um einen Live-Stream mit höherer Teilnehmerzahl zu testen, wurde ein Script geschrieben, das mit Hilfe von puppeteer[3] mehrere Chrome-Browser im 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 wiederzugeben. Das Skript 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, dass es ausschließlich .ts-Dateien, also die Videosegmente 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 weitergeleitet. Die Weiterleitung erfolgte verzögert in einem Zeitraum von 18-60 Sekunden mit einer Dreiecksverteilung, um die Last auf Seiten der Slidesync-Server zu verringern. Der Stream wurde für zehn Minuten live geschaltet und anschließend beendet.

Abbildung 20 zeigt, wie sich das Peer-to-Peer-CDN bei steigender Teilnehmerzahl mit einem Mesh verhält. Bis 50 Teilnehmern

3 https://github.com/GoogleChrome/puppeteer

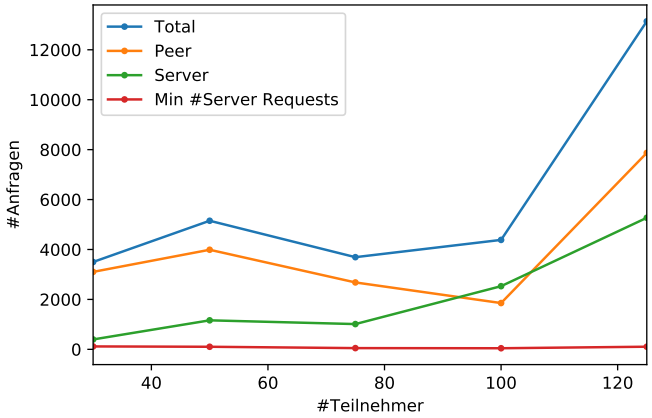

Figure 20: Vergleich der Bearbeitung nach Art in einem Mesh

zeigt das CDN ein lineares Verhalten hinsichtlich bearbeiteter Anfragen. Ab 75 Teilnehmern ist zu sehen, dass die Gesamtanzahl der verarbeiten Anfragen einbricht. Die CPU-Last der Testserver stieg auf 100% und die Clients waren dadurch nicht mehr in der Lage, am 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 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 zurückzuführen, dass die Verbindungen zwischen den Peer durch die hohe Last teilweise unterbrochen wurde, wodurch sich die Last bei den Teilnehmern verringerte.

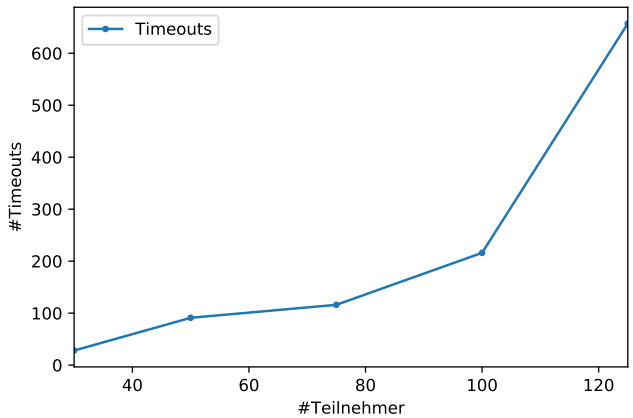

Figure 21: Timeouts bei einem Mesh

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 kann es zu Timeouts kommen, falls zu viele Teilnehmer gleichzeitig eine Anfrage an denselben 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, dass 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. *XXX*

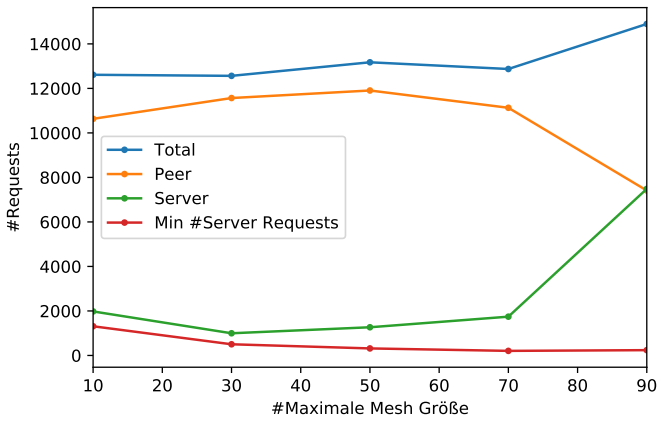

Figure 22: Vergleich verschiedener Mesh Größen

Abbildung 22 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 vom Server beantwortet werden müssen, mit 10% relativ groß. Auch wenn nur selten Timeouts durch zu viele Anfragen bei demselben 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 müssen mindestens 4% der Anfrage über den Server beantwortet werden, damit jedes HLS-Segment in jedem Peer-Mesh vorhanden ist. Wird eine größere Mesh-Größe als 30 gewählt, steigt auch die Anzahl der Timeouts, ebenso wie die CPU-Auslastung bei den Teilnehmern. So konnte bei einer Mesh-

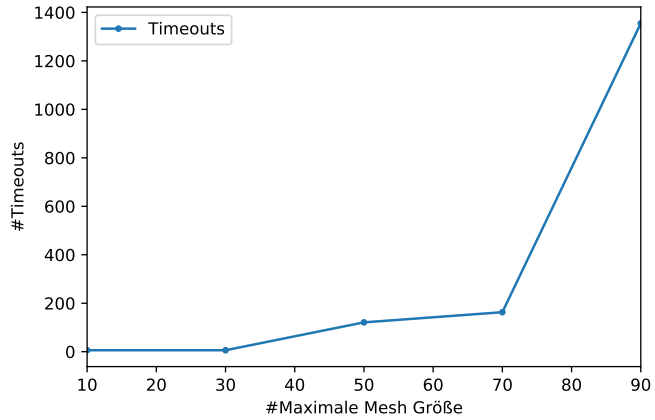

Figure 23: Timeouts bei verschiedenen Mesh-Größen (125 Teilnehmer)

Größe von 90 lediglich die Hälfte aller Anfragen durch das Peer-to-Peer-CDN beantwortet werden.

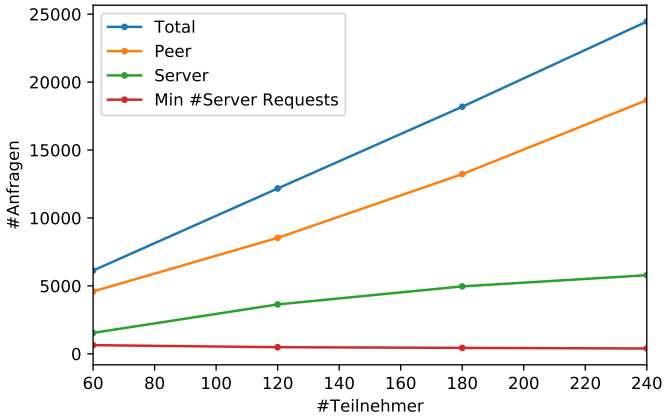

Figure 24: Mesh-Größe von 30 bei steigender Teilnehmer Zahl

Um zu testen, wie sich das CDN bei größeren Teilnehmerzahlen verhält, wurden Tests mit einer Mesh-Größe von 30 durchgeführt. Abbildung 24 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 25 zeigt eine zeitliche Darstellung von Beginn 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 geladen ist, laden alle Teilnehmer drei HLS-Segmente vor. Diese HLS-Segmente werden gleichzeitig geladen,

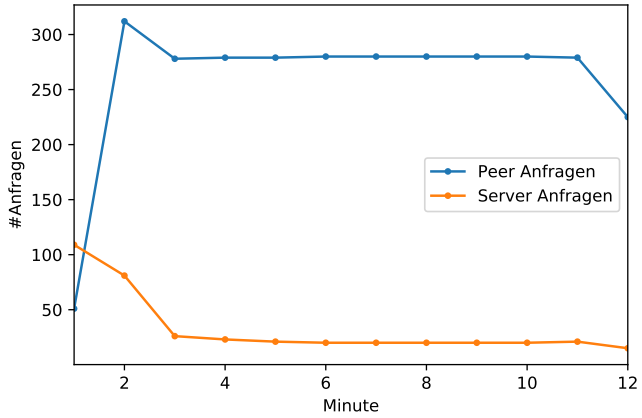

Figure 25: Anfrageart über die Zeit bei 30 Teilnehmern

wodurch sich die Wahrscheinlichkeit erhöht, dass mehrere Teilnehmer annähernd gleichzeitig dieselbe Ressource anfragen und kein anderer Teilnehmer sie bereits geladen hat.

6.2.3 *Schul-Cloud*

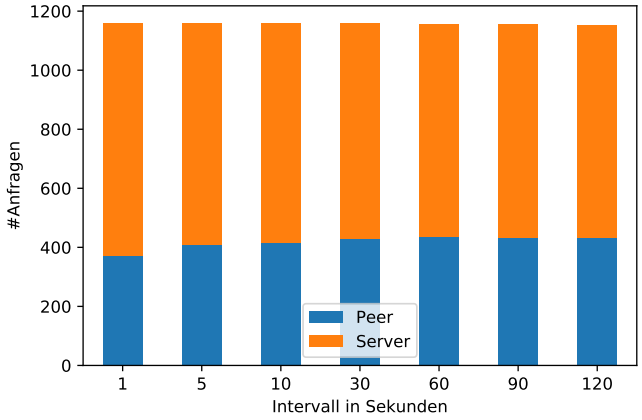

Figure 26: Anfragearten gegliedert nach Ladeintervall

Um die Verwendung des CDNs in Schulen zu evaluieren, wurde simuliert, dass 30 Schüler im Rahmen des Unterrichts auf die Seite Dashboard zugreifen, die Kursliste aufrufen, einen Kurs öffnen und ein Thema auswählen. Alle Clients wurden von einem Server gestartet und riefen die Seite zu einem zufälligen Zeitpunkt innerhalb verschiedener Intervalle auf.

Abbildung 26 zeigt die Verteilung der Anfragen nach Anfrageart. Es ist gut zu sehen, dass die Größe des Intervalls, in dem die Schüler auf die Seiten zugriffen, keinen großen Einfluss auf die

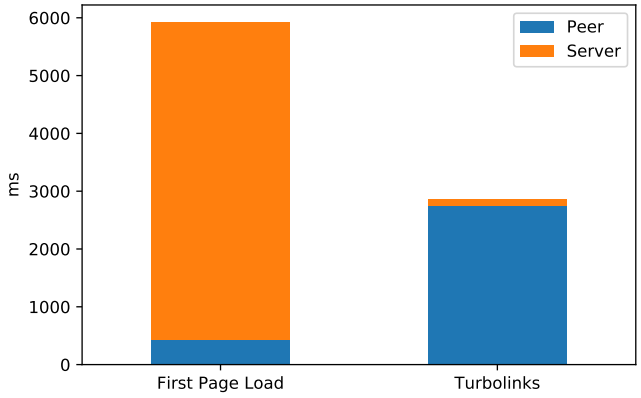

Figure 27: Erstmaliges Laden vs. Navigation

Effektivität des Peer-to-Peer-CDN hatte. Zwar war die Effektivität bei einem Zugriff aller Schüler innerhalb einer Sekunde geringfügig schlechter, jedoch fällt dies kaum ins Gewicht. Bei einem Intervall von mehr als 5 Sekunden ist kaum noch eine Verbesserung der Effektivität zu beobachten. Jedoch liegt die Ef-

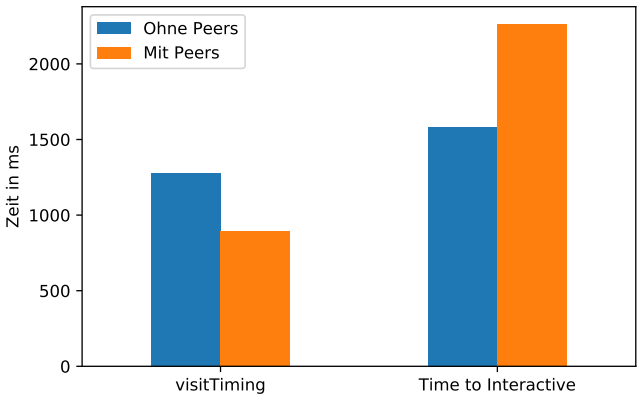

Figure 28: Seitenladezeiten

Bildbeschriftung

fektivität des CDN bei den durchgeführten Tests bei nur ca. 38%. Betrachtet man den Vergleich des ersten Ladens der Seite und den folgenden Navigationen, wird schnell ersichtlich, worauf dies zurückzuführen ist. Beim ersten Laden der Seite, einem kompletten Page Load, werden Anteilig deutlich mehr Anfragen gestellt, als bei den folgenden Navigationen. Da das CDN jedoch noch nicht komplett geladen ist, kann nur ein kleiner Anteil der Anfragen durch das CDN beantwortet werden. Da die Navigationen über Turbolinks gehandhabt wurden, war kein kompletter Page Load notwendig. Das CDN war zwar von Be-

ginn der Navigation an verfügbar, jedoch war Turbolinks in der Lage einen Großteil der Anfragen zu vermeiden, da es sich um Assets handelte, die für beide Seiten gleich waren. Ein erneutes Laden dieser Ressourcen war demnach nicht notwendig. Bei den Navigationen konnte das CDN eine Effektivität von annähernd 100% erreichen.

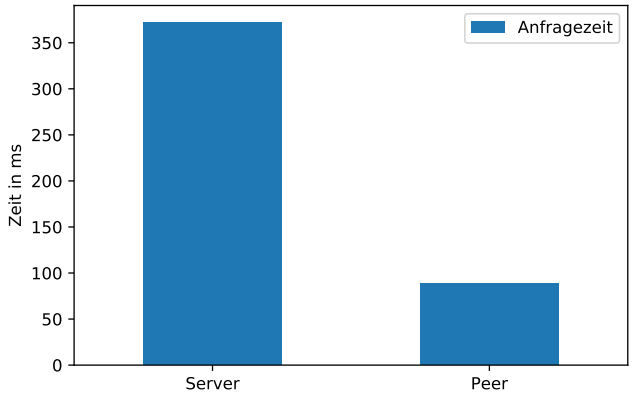

Figure 29: Durchschnittliche Ladezeit von Anfragen nach Bearbeitungsart

Abbildung 28 zeigt einen Vergleich der Seitenladezeiten mit und ohne verfügbare Peers. Hierbei gilt zu beachten das für den ersten Page Load andere Metriken verwendet werden müssen als bei den Navigationen. Turbolinks greift in den Ladeprozess des Browsers ein, wodurch Metriken, die von den Browsern zur Verfügung gestellt werden, im Falle einer Navigation inkorrekte Daten zurückgeben. Umgekehrt stellt Turbolinks zwar ebenfalls Methodiken zur Messung von Metriken bereit, jedoch geben diese für den ersten Page Load keine Ergebnisse zurück, da Turbolinks diese Metriken selbst erfasst. Betrachtet man beide Fälle separat, ist ein Vergleich jedoch möglich. Zur Messung des ersten Page Load wurde die Time to Interactive gemessen. Es ist gut zu sehen das das Peer-to-Peer-CDN einen gewissen Mehraufwand bedeutet. Bei den folgenden Navigationen kann jedoch eine Verbesserung der Ladezeit beobachtet werden. Dies ist vor allem auf die schnellere Beantwortung von Antworten zurückzuführen. Abbildung 29 zeigt deutlich, dass Anfragen, die über das Peer-to-Peer-CDN bearbeitet wurden, deutlich schneller beantwortet werden konnten.

6.3 LIVE-STREAMING IN UNTERNEHMENSNETZWERKEN

Um zu testen, wie sich das CDN innerhalb eines echten Netzwerks mit separaten Rechnern verhält, wurde ein Live-Stream mit

der Infrastruktur von SlideSync aufgesetzt. SlideSync verwendet bei der Verteilung des Videos das Datenformat HLS. Das CDN wurde so konfiguriert, dass ausschließlich die HLS-Segmente über das CDN behandelt werden. Sämtliche im folgenden betrachteten Anfragen sind demnach HLS-Segmente. Der Test wurde

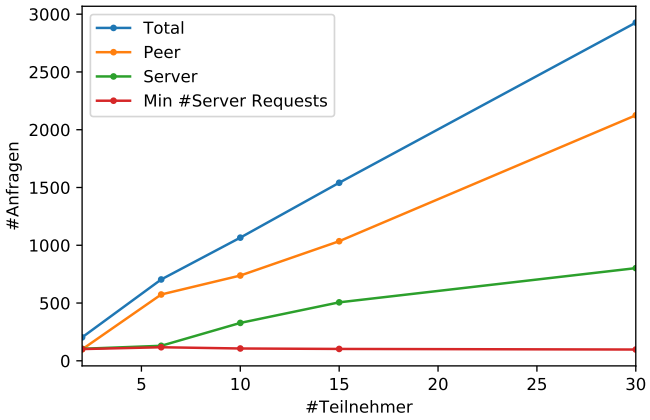

Figure 30: Vergleich der Anfragen nach Anfrageart

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 Eventseite geladen und erst im Anschluss der Live-Stream gestartet. Getestet wurden sechs, zehn, 15, 20 und 30 Clients, die sich alle im selben lokalen Netzwerk befanden. Der Streaming- und der Anwendungsserver befanden 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 demselben Peer Mesh.

Abbildung 30 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 wurden. Bei steigender Anzahl von Teilnehmern, ist zu beobachten, dass eine größere Anzahl 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, dass zwei Teilnehmer sich beide an derselben Stelle im Video befinden, die HLS-Segmente laden müssen, jedoch noch kein Teilnehmer das Segment heruntergeladen hat. Auch wird es wahrscheinlicher, dass eine größere Anzahl an Teilnehmern ein Segment über denselben Teilnehmer laden wollen. Fragen zu viele Teilnehmer eine Ressource bei demselben Teilnehmer an, so kann er nicht mehr alle Anfragen

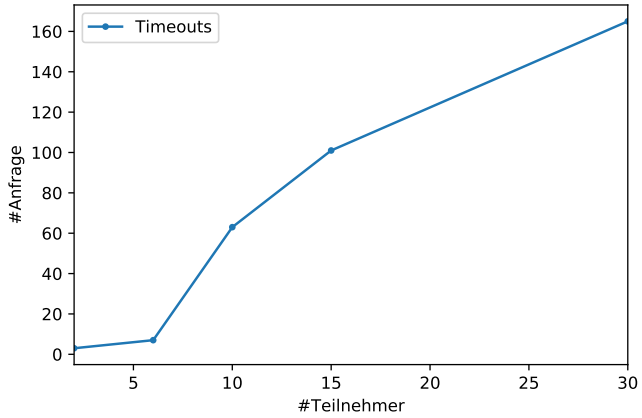

Figure 31: Anzahl der Timeouts des CDNs

bearbeiten. Die Anfrage über das Peer-to-Peer-CDN wird abgebrochen und über den Server geladen. Abbildung 31 zeigt die Anzahl der Timeouts bei steigender Teilnehmerzahl. Dabei ist zu beachten, dass der Timeout so gewählt wurde, dass das Video weiterhin flüssig lief.

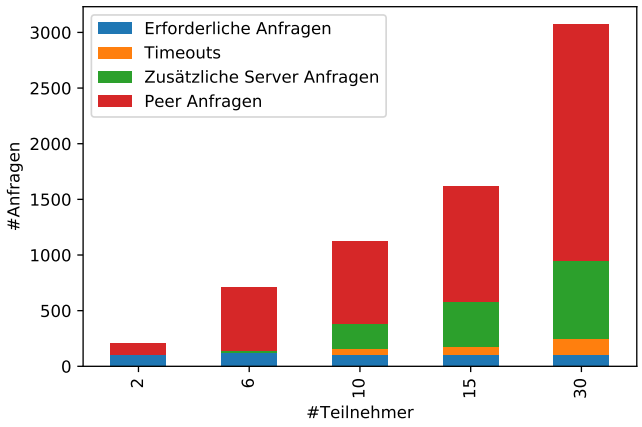

Figure 32: Vergleich der Anfragen nach Anfrageart

Betrachtet man die zusätzlich notwendigen Server-Anfragen, so lässt sich beobachten, dass der Prozentsatz in diesem Test annähernd konstant ist.

Abbildung 33 zeigt, welche Teilnehmer untereinander Daten ausgetauscht haben. Auffällig ist, dass 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, d.h. zwar haben einige Teilnehmer besonders viele Anfragen beantwortet, jedoch haben sie sich nicht als zentrale

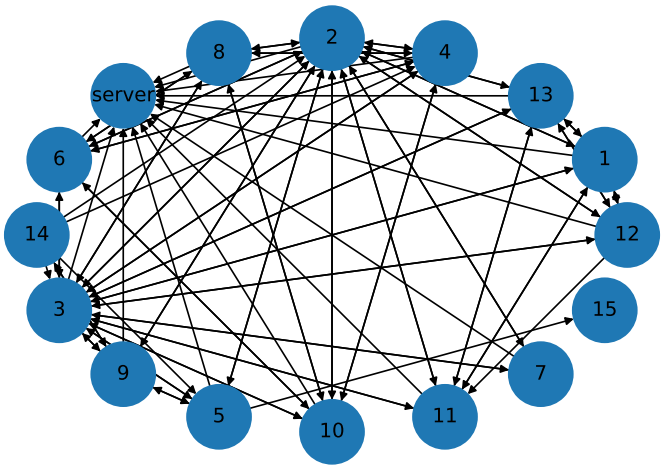

Figure 33: Vergleich der Anfragen nach Anfrageart

Punkte des Netzwerkes etabliert, sondern ebenfalls ihre Daten von anderen Teilnehmern geladen.

gewichtung??

Bearbeitung	Anfragen	Durchschnitt	Standardabweichung
Peer	4571	201.60 ms	190.14 ms
Server	1870	63.61 ms	62.99 ms

Table 5: Verarbeitungsdauer

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. Dies ist auf den geringeren Durchsatz der Webrtc Datachannels zurückzuführen. (Siehe 6.2.1) Damit war das Peer-to-Peer-CDN zwar deutlich langsamer, jedoch schnell genug, eine flüssige Wiedergabe zu gewährleisten. Da eine Anfrage, die länger als 3000ms brauchte, bei dem Peer-to-Peer-CDN zu einem timeout führte, betrug die höchste erfasste Ladezeit 2832,27 ms.

6.4 OFFLINE-SUPPORT

Service Workers werden häufig im Rahmen von Offline-Support für Websites genannt. Mithilfe der Caching-API lassen sich Internetseiten speichern und später, wenn kein Internet vorhanden ist, wieder abrufen. Insbesondere für Schulen ist dies interessant,

da die Internetanbindung in Schulen in vielen Fällen nach wie vor unzureichend ist. Die Sonderstudie Digital der Initiative befragte 2016 1426 Lehrer, von denen 4% angaben, dass es an Ihrer Schule keine Internetverbindung gibt. Rund 50% der befragten Lehrer gaben an, dass es Internetzugriff nur in bestimmten Räumen gebe.[10] In Kombination mit einem Peer-to-Peer-CDN lässt sich dies nutzen, um Internetseiten und Ressourcen auch dann zu laden, wenn die Internetverbindung ausfällt. Insbesondere das Szenario, dass der Lehrer die Seite bzw. Inhalte zu Hause vorlädt und im Anschluss den Schülern bereitstellt, erscheint interessant. Tobias Wollowski beschreibt in seiner Arbeit "Optimierung von Web-Anwendungen für den Einsatz im Klassenzimmer"[45] Möglichkeiten zur Nutzung von Offline-Caches im Unterricht. Im Folgenden wird konzeptionell beschrieben, wie das vorgestellte Peer-to-Peer-CDN angepasst werden kann, um auch im Falle einer nicht vorhandenen Internetanbindung zu funktionieren.

Durch die Verwendung von Service Workers als Proxy und Cache bietet das CDN eine gute Grundlage für Offline-Support. Es stellt die Möglichkeit bereit, Inhalte aus dem Cache an andere Nutzer zu verteilen. Die größte Herausforderung bei der Offline-Nutzung stellt das Signaling, also das Verbinden von Nutzern, dar. Um dies zu bewerkstelligen, ist bei der gewählten Implementierung ein Websocket-Server erforderlich. Ist die Verbindung bereits hergestellt, z.B. vor einem Internetausfall, so können bereits bei der momentanen Implementierung Inhalte in dem Peer Mesh verteilt werden. Ist dies nicht der Fall, so muss im lokalen Netzwerk ein Signaling Server bereitgestellt werden, der auch ohne Internetverbindung erreichbar ist. Auch muss der Schüler bestimmte Ressourcen wie das Peer-to-Peer-CDN schon im Vorfeld geladen und mit Hilfe eines Service Workers gespeichert haben. Sind die IPs einiger Nutzer bereits bekannt, so ist es ebenfalls denkbar, das weitere Signaling über Webrtc auszuhandeln. Dazu ließen sich Algorithmen wie Kademliakademlia zum Auffinden von Peers verwenden. Der größte Nachteil ist jedoch, das anfänglich die IP-Adressen von Nutzern bekannt sein müssen. Hassan Ijaz[3] beschreibt verschiedene Möglichkeiten, Webrtc-Verbindungen auch ohne Internet aufzubauen. Unter anderem ist es möglich Webrtc-Verbindungen mit Hilfe von QR-Codes zu initialisieren. Der Lehrer könnte den Schülern einen QR-Code oder, in einer anderen Codierung, ein SDP Packet bereitstellen, mit dessen Hilfe die Schüler eine Webrtc-Verbindung zum Lehrer herstellen können. Anschließend fungiert der Lehrer als Signaling Server für die Schüler über Webrtc, sodass Verbindungen zwischen den Schülern hergestellt werden können.

Offline-Support ist für Livestreams vor allem interessant, um die Ausfallsicherheit zu erhöhen. SlideSync lädt 18 Sekunden Video vor, bevor es abgespielt wird. Durch Verwendung des Peer-to-Peer-CDN können kürzere Ausfälle überbrückt werden, sodass zumindest ein Teil der Nutzer das Video weiterhin über andere Nutzer laden und verteilen können. Ein Signaling ist nicht notwendig, da die Verbindungen bereits hergestellt wurden. Da Unternehmen in aller Regel über ausreichende Internetverbindungen verfügen und es sich um Live-Inhalte handelt, ist ein reiner Offline-Support eher von geringem Interesse.

6.5 SICHERHEIT

Die Verwendung von Peer-to-Peer-CDNs bringt einige Herausforderungen im Bereich Sicherheit mit sich. Da dies nicht Schwerpunkt der Untersuchungen im Rahmen dieser Arbeit ist, wird eine vertrauenswürdige Umgebung angenommen. Dennoch werden im Folgenden Herausforderungen im Bereich der Sicherheit bei der Verwendung eines Peer-to-Peer-CDNs beleuchtet und mögliche Lösungen vorgestellt.

Eines der Grundprinzipien des Internet Security Models ist, das Nutzer bedenkenlos Internetseiten aufrufen und dort Skripts laden können.[13] Dieses Versprechen an den Nutzer muss der Browser garantieren, um eine sichere Benutzung des Internet zu gewährleisten. Dazu werden Skripts in Sandboxes ausgeführt.[33] Ein Skript kann nur im Kontext der aktuellen Seite ausgeführt werden und Anfragen, die über die aktuelle Seite hinausgehen, müssen vom Quellen-Server explizit dafür freigegeben werden.[9]

Service Worker arbeiten nach demselben Prinzip. Sie haben keinen Zugriff auf das DOM und können nur Anfragen verarbeiten, die im Scope der Seite liegen, die sie geladen hat.[35] Anfragen, die über einen Service Worker behandelt werden und nicht der Same-Origin Policy folgen, müssen mit dem Entsprechenden CORS-Header versehen sein, um geladen zu werden. Service Workers können nur über HTTPS geladen werden. Wird eine Ressource vom Server neu in das Peer-to-Peer-CDN geladen, so muss sie den gleichen Sicherheitsvorkehrungen des Browsers entsprechen, wie jede andere Anfrage. Wird eine Anfrage über das Peer-to-Peer-CDN selbst beantwortet, so greifen diese Mechanismen jedoch nicht. Es ist möglich, dass ein bösartiger Client eine Ressource verändert, bevor er sie an einen Client sendet, der diese dann lädt.

Um die Integrität einer Ressource sicherzustellen, stellen Browserfunktionalitäten bereit, die auf Hashes der Ressource beruhen,

die bei Anfrage mitgesendet werden.[39] Dies wird bei Anfragen automatisch durch den Browser übernommen. Bei der Übertragung mittels Webrtc muss dies die Anwendung selbst übernehmen.

Durch die Verwendung von nativen Browserfunktionalitäten entfällt die Installation von Plugins oder Anwendungen. Dadurch entfällt das Risiko von Malware in Installationsdateien. Umso wichtiger ist jedoch, dass ein vertrauenswürdiger Browser verwendet wird. Wurde der Browser aus einer unsicheren Quelle geladen, so kann eine sichere Verwendung des CDNs nicht gewährleistet werden, da die Sicherheitskonzepte auf der Implementierung des Browser beruhen.[1]

Wird HTTPS bei der zugrundeliegenden Anwendung verwendet, so wird auch der Datenverkehr über Webrtc mittels TLS verschlüsselt.[34] Daten, die mittels Webrtc Datachannels gesendet werden, werden über DTLS (Datagram Transport Layer Security) übertragen. Dies geschieht auch, falls ein TURN-Server verwendet wird. Dadurch sind die Verbindungen Ende-zu-Ende verschlüsselt.

Die Websocket-Verbindung, die für das Signaling verwendet wird, sollte ebenfalls verschlüsselt werden. Die Anwendung muss außerdem sicherstellen, dass sich keine unautorisierten Clients mit dem Websocket Channel verbinden, da sie ansonsten in der Lage sind, Ressourcen anzufragen, für die sie keine Berechtigung haben, oder schadhafte Skripts an Nutzer senden könnten.

Ein weiteres im Schulkontext zu betrachtendes Szenario ist die Verwendung eines Rechners durch mehrere Nutzer. Viele Schulen haben Computerräume, in denen Rechner stehen, die von verschiedenen Klassen und Schülern verwendet werden. Zwar wäre es prinzipiell möglich, den gesamten Cache beim Login/logout zu löschen, jedoch würde das die Cache-Trefferrate negativ beeinflussen. Um dies zu umgehen, können die Ressourcen verschlüsselt gespeichert werden. Wird dazu ein Schlüssel verwendet, der nur im aktuellen Scope verfügbar ist so kann sie nur entschlüsselt werden, falls der Nutzer Zugriff auf den Scope der Ressource hat. Ein unbefugtes Auslesen des Caches ist somit zwar möglich, jedoch sind die Daten für den Angreifer unbrauchbar.

Auch wenn der Signaling Server einen Teil der Nutzer Authentizität sicherstellt, so sollte diesem nicht vollkommen vertraut werden. Der Signaling Server kann lediglich sicherstellen, dass ein Nutzer in eine bestimmte Sicherheitsklasse gehört, da er nur den Zugriff auf einen Websocket-Channel sichert. Ist es darüber hinaus notwendig, die Authentizität eines Nutzers zu

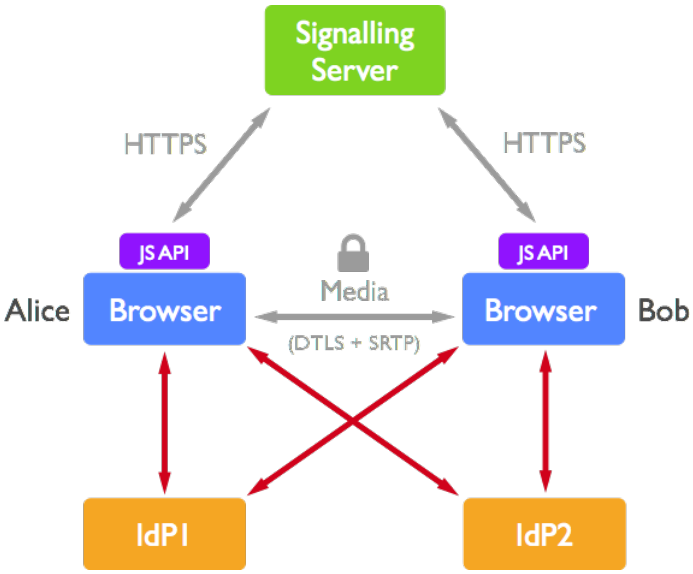

Figure 34: IdP basierter Verbindungsaufbau[34]

gewährleisten, können webbasierte Identitäts-Provider (IdP) verwendet werden. Diese fungieren als vertrauenswürdige Partei und können die Identität eines Nutzers bestätigen. Abbildung 34 zeigt die Struktur einer solchen Verbindung.

Durch die Verwendung von Webrtc bei der Verwendung von STUN-Servern besteht das Risiko eines IP-Leaks.[34] Es ist prinzipiell möglich, die IP-Adresse eines Nutzers zu ermitteln. Eine mögliche Gegenmaßnahme ist die Deaktivierung von Webrtc. Dies kann entweder in der Browserkonfiguration oder durch Plugins geschehen. Jedoch ist dann die Verwendung von Webrtc nicht mehr möglich. Um eine direkte Verbindung zu einem anderen Nutzer aufzubauen ist es notwendig, dessen IP-Adresse zu ermitteln. Stellt die Gefahr eines IP-Leaks ein großes Risiko in der betrachteten Domäne da, so kann das Peer-to-Peer-CDN nicht verwendet werden. Im Rahmen dieser Masterarbeit wird jedoch nur die Verbindung im lokalen Netzwerk ohne die Verwendung eines STUN-Servers untersucht. Es besteht also lediglich

die Gefahr, dass die lokale IP-Adresse eines Nutzers ermittelt werden kann - nicht jedoch dessen öffentliche IP-Adresse.

7

AUSBLICK

- Meshes verbinden
- preloading mitteilungen
- web app zweistufig, kleine lädt vor dann content
-

8

FAZIT

- Browser nutzung

A

AN APPENDIX

Some stuff here

BIBLIOGRAPHY

[1] *A Study of WebRTC Security*. Abgerufen: 09.09.2019. URL: https://webrtc-security.github.io/.

[2] 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/.

[3] *Architecture to establish serverless webrtc connections*. Pub. No.: US 2016/0021148A1. Jan. 2016. URL: https://patentimages.storage.googleapis.com/99/66/5a/c6e189e849027c/US20160021148A1.pdf.

[4] 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/.

[5] 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/.

[6] S. Nandakumar und C. Jennings. *SDP for the WebRTC*. Feb. 2013. URL: https://tools.ietf.org/id/draft-nandakumar-rtcweb-sdp-01.html.

[7] *Can I use Webrtc*. Abgerufen: 10.09.2019. URL: https://caniuse.com/\#feat=rtcpeerconnection.

[8] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System." In: *Lecture Notes in Computer Science* 2009 (Mar. 2001). DOI: 10.1007/3-540-44702-4_4.

[9] *Cross-Origin Resource Sharing*. Abgerufen: 09.09.2019. URL: https://www.w3.org/TR/cors/.

[10] Initiative D21. *"Sonderstudie »Schule Digital«". In: Sonderstudie »Schule Digital« Eine Studie der Initiative D21, durchgeführt von Kantar TNS*. Pub. No.: US 2016/0021148A1. 2016. URL: https://initiatived21.de/app/uploads/2017/01/d21_schule_digital2016.pdf.

[11] 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.

[12] *HTTP Live Streaming*. Abgerufen: 14.09.2019. URL: https://tools.ietf.org/html/draft-pantos-http-live-streaming-23.

[13] Eric Y. und Barth Adam und Rescorla Eric und Jackson Collin Huang Lin-Shung und Chen. "Talking to Yourself for Fun and Profit." In: Carnegie Mellon University, Jan. 2011.

[14] Daniel C. Johnston Alan B. und Burnett. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web - Second Edition*. June 2013. ISBN: 978-0-9859788-5-3.

[15] PricewaterhouseCoopers GmbH WPG atene KOM GmbH. "Aktuelle Breitbandverfügbarkeit in Deutschland(Stand Ende 2018)." In: Bundesministerium für Verkehr und digitale Infrastruktur (BMVI), 2018.

[16] Vana Kalogeraki, Dimitrios Gunopulos, and D. Zeinalipour-Yazti. "A Local Search Mechanism for Peer-to-peer Networks." In: *Proceedings of the Eleventh International Conference on Information and Knowledge Management*. CIKM '02. McLean, Virginia, USA: ACM, 2002, pp. 300–307. ISBN: 1-58113-492-4. DOI: 10.1145/584792.584842. URL: http://doi.acm.org/10.1145/584792.584842.

[17] Xiuqi Li and Jie Wu. "Searching Techniques in Peer-to-Peer Networks." In: *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks* (Aug. 2005). DOI: 10.1201/9780203323687.ch37.

[18] Gurmeet Manku, Mayank Bawa, Prabhakar Raghavan, and Verity Inc. "Symphony: Distributed Hashing in a Small World." In: (Feb. 2003).

[19] Petar Maymounkov and David Eres. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric." In: vol. 2429. Apr. 2002. DOI: 10.1007/3-540-45748-8_5.

[20] Christoph Meinel and Harald Sack. *Kommunikation, Internetworking, Web-Technologien*. Jan. 2004. ISBN: 978-3-642-18963-0.

[21] *Microsoft is building a Chromium-powered web browser that will replace Edge on Windows 10*. Abgerufen: 16.09.2019. URL: https://www.windowscentral.com/microsoft-building-chromium-powered-web-browser-windows-10.

[22] *Millionenschaden durch IT-Ausfälle in Unternehmen*. Abgerufen: 10.09.2019. URL: https://www.springerprofessional.de/informationstechnik/risikomanagement/millionenschaden-durch-it-ausfaelle-in-unternehmen/15429886.

[23] 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.

[24] *Offline Storage for Progressive Web Apps*. Abgerufen: 10.09.2019. URL: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa.

[25] *Policy Settings in Chrome*. Abgerufen: 11.09.2019. URL: http://dev.chromium.org/developers/how-tos/enterprise/adding-new-policies.

[26] *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2018 und 2025 (in Zettabyte)*. Abgerufen: 10.09.2019. URL: https://de.statista.com/statistik/daten/studie/267974/umfrage/prognose-zum-weltweit-generierten-datenvolumen/.

[27] *Quota Management API : Fast Facts*. Abgerufen: 10.09.2019. URL: https://developers.google.com/web/updates/2011/11/Quota-Management-API-Fast-Facts.

[28] M. Ripeanu. "Peer-to-peer architecture case study: Gnutella network." In: *Proceedings First International Conference on Peer-to-Peer Computing*. 2001, pp. 99–100. DOI: 10.1109/P2P.2001.990433.

[29] J. Rosenberg. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*. Apr. 2010. URL: https://tools.ietf.org/html/rfc5245.

[30] 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.

[31] *Ruby on Rails Guides (v6.0.0)*. Abgerufen: 01.09.2019. URL: https://guides.rubyonrails.org/.

[32] 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/.

[33] *Sandbox*. Abgerufen: 09.09.2019. URL: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md.

[34] *Security Considerations for WebRTC*. Abgerufen: 09.09.2019. URL: https://tools.ietf.org/html/draft-ietf-rtcweb-security-05.

[35] *Service Worker API*. Abgerufen: 09.09.2019. URL: https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API.

[36] *Single Page Application*. Abgerufen: 01.09.2019. URL: https://wirtschaftslexikon.gabler.de/definition/single-page-application-54485.

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

[38] Ion Stoica, Robert Morris, David Karger, M. Kaashoek, and Hari Balakrishnan. "Chord: A scalable peer-to-peer lookup service for internet applications." In: vol. 149-160. Jan. 2001, pp. 149–160. DOI: 10.1145/383059.383071.

[39] *Subresource Integrity*. Abgerufen: 09.09.2019. URL: https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity.

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

[41] *Using the Cache API*. Abgerufen: 01.09.2019. URL: https://developers.google.com/web/fundamentals/instant-and-offline/web-storage/cache-api.

[42] *WebRTC Data Channels*. Abgerufen: 09.09.2019. URL: https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13.

[43] *WebRTC Data Channels*. Abgerufen: 09.09.2019. URL: https://tools.ietf.org/html/rfc4960.

[44] *WebRTC in the real world: STUN, TURN and signaling*. Abgerufen: 01.09.2019. URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/#what-is-signaling.

[45] Tobias Wollowski. *Optimierung von Web-Anwendungen für den Einsatz im Klassenzimmers*. Aug. 2019.

[46] Beverly Yang and Hector Garcia-molina. "Improving search in peer-to-peer networks." In: Feb. 2002, pp. 5–14. ISBN: 0-7695-1585-1. DOI: 10.1109/ICDCS.2002.1022237.

[47] 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