

# Web-basierte Systeme – Übung

## 04: Webpack, WebRTC, und Aufgabe 3

---

Wintersemester 2023

Arne Vogel, Maxim Ritter von Onciul



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



Friedrich-Alexander-Universität  
Technische Fakultät

Webpack

WebRTC Grundlagen

PeerJS Library

Aufgabe 3

# Webpack

---

Webpack

WebRTC Grundlagen

PeerJS Library

Aufgabe 3

# Motivation für Module-Bundler wie Webpack

- Ursprünglich wurde JavaScript primär dazu benutzt HTML-Dokumente um animierte Elemente zu ergänzen
- JavaScript wurde/wird direkt in HTML-Dokumenten geschrieben oder verlinkt
- Für komplexe Webanwendungen ist dieses Vorgehen nicht zielführend
  - Kein Package-Manager
  - Kein Modulsystem
  - Kein Buildsystem (Linter, Compiler, usw.)

- JS-Datei wird in der HTML-Datei geladen

---

```
1 ...  
2 <script src="https://awesomeproject/awesome.min.js"><script>  
3 ...
```

---

- JS-Code wird meistens in separater Datei genutzt

---

```
1 const a = awesome_function()
```

---

- Dadurch werden alle Objekte in den **globalen Namespace** geladen
- Gefahr von **shadowing**
- Keine Modulverwaltung (automatische Updates, usw.)

- Module werden über ein Package-Manager verwaltet

---

```
1 npm install awesomeLib
```

---

- Einzelne Funktionen oder ganze Bibliotheken werden über **import statements** in einen **lokalen Namespace** geladen

---

```
1 import awesome from 'awesomeLib'  
2 const a = awesome.doAwesomeStuff()
```

---

# npm (ehemals Node Package Manager)

- Paketmanager für die JavaScript-Laufzeitumgebung Node.js
- Über Bundler wie Webpack auch für clientseitiges JavaScript
- Alle build dependencies werden in der **package.json** verwaltet

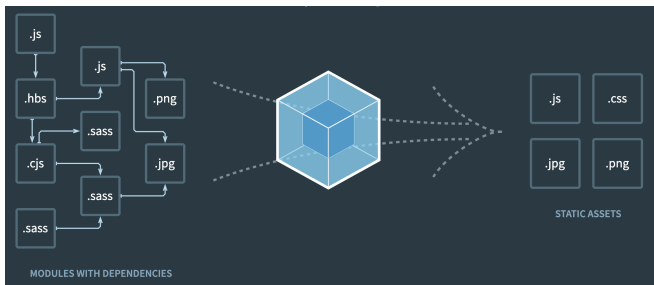
---

```
1 {  
2     "name": "test-project",  
3     "version": "1.0.0",  
4     "main": "src/main.js",  
5     ...  
6     "dependencies": {  
7         "vue": "^2.5.2",  
8         "awesomeLib": "0.5.3"  
9     },  
10 }
```

---



# Webpack - Übersicht



- Statischer Module-Bundler
- Löst Abhängigkeitsbaum auf und geniert statische Assets (Bundles)
- Webpack wird von React, Angular, Vue verwendet

# Webpack Projektstruktur

---

```
1 test-project
2 |- package.json
3 |- webpack.config.js
4 |- node_modules      # externe Libs - dynamisch von npm generiert
5 |- /src
6   |- index.js
7   |- myModule.ts
8 |- /styles
9   |- chat.css
10  |- main.css
11  |- giphy.sass
12 |- /dist             # gebaute Webanwendung - dynamisch von Webpack generiert
13   |- main.js
14   |- index.html
15   |- style.css
```

---

# Webpack import/export

## Export:

---

```
1 export const Count = 5;
2 export function Multiply(a, b) {
3   return a * b;
4 }
5
6 // Default export
7 export default {
8   // Some data...
9 };
```

---

## Import:

---

```
1 import MyModule from './my-module.js';
2 import { NamedExport } from './other-module.js';
```

---

## Example import/export

---

```
1 // Named exports
2 export var Count = 5;
3 export function Multiply(a, b) {
4   return a * b;
5 }
6
7 // Default export
8 export default {
9   add: function (a, b) {
10    return a + b;
11  },
12  subtract: function (a, b) {
13    return a - b;
14  },
15 };
```

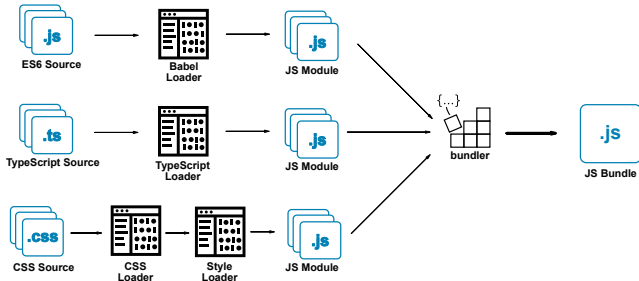
---

---

```
1 // Importing named exports
2 import { Count, Multiply }
3   from './mathOperations.mjs';
4
5 console.log(Count);
6 console.log(Multiply(2, 3));
7
8 // Importing the default export
9 import MathOps
10   from './mathOperations.mjs';
11
12 console.log(MathOps.add(4, 2));
13 console.log(MathOps.subtract(4, 2));
```

---

# Webpack - Funktionsweise



- Webpack besteht aus Plugins und Loadern
- Webpack erlaubt das Importieren verschiedenster Dateitypen
- Abhängigkeiten werden vom **entry point** aus gesucht
- Typabhängige **Loader** formen Code in einheitliche JS-Module um

# WebRTC Grundlagen

---

Webpack

WebRTC Grundlagen

PeerJS Library

Aufgabe 3

- **WebRTC**: offener Standard, der
  - Web Real-Time Communication
  - “Web-Echtzeitkommunikation”
- Kommunikation heißt: Datenübertragung **direkt zwischen Browsern**
  - Video/Voice-Streams
  - Bidirektionale Daten: **DataChannels**
- Möglichst **ohne** System dazwischen
  - WebRTC implementiert **Fallbacks**, damit immer eine Verbindung zustande kommen kann
- Hauptprobleme: NAT, Firewalls

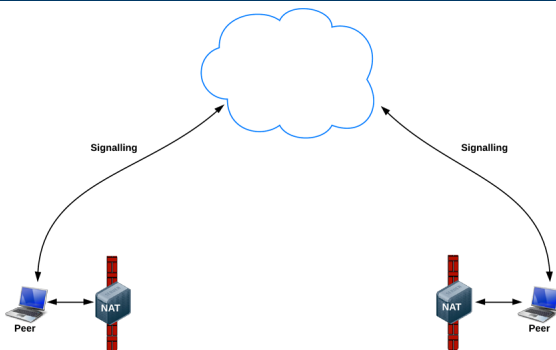


- **Signaling** (Signalisierung) dient der Koordinierung der WebRTC-Kommunikation
- WebRTC-Clients tauschen dabei folgende Informationen aus:
  - Session control messages (Öffnen/Schließen der Verbindung)
  - Fehlernachrichten
  - Media Metadaten: Einigung auf Codecs/Bandbreite
  - Schlüssel für sichere Verbindungen
  - Netzwerk Metadaten (IP Adressen und Ports)
- Unterliegendes Protokoll ist im WebRTC-Standard **nicht definiert**
  - Maximale Kompatibilität mit bestehenden Technologien



- Ideale Welt: Direkte Verbindung zwischen Endgeräten möglich
- Problem: NAT (Network Address Translation) und/oder Firewalls verhindern Verbindungsaufbau
- Lösung: ICE (Interactive Connectivity Establishment)

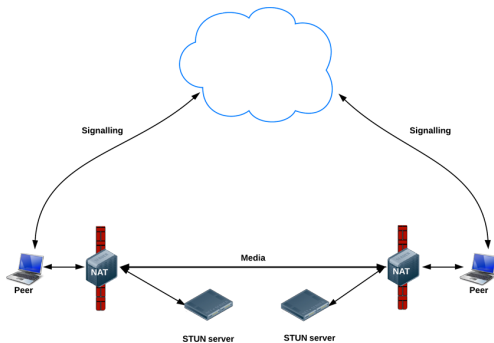
# Verbindungsaufbau in WebRTC



- Ideale Welt: Direkte Verbindung zwischen Endgeräten möglich
- Problem: NAT (Network Address Translation) und/oder Firewalls verhindern Verbindungsaufbau
- Lösung: ICE (Interactive Connectivity Establishment)

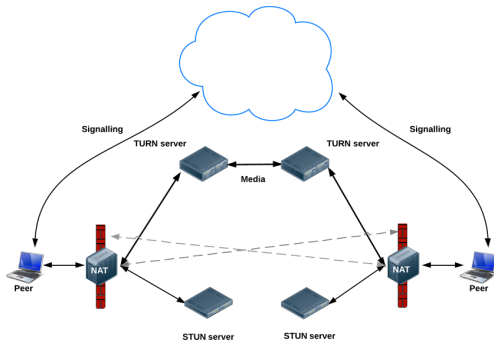
- Verbindungsaufbau mittels ICE (Interactive Connectivity Establishment)
  - 1. Versuch: direkte Verbindung
  - 2. Versuch: Session Traversal Utilities for NAT (STUN)
  - 3. Versuch: Traversal Using Relays around NAT (TURN)
- Weitere Protokolle werden unterstützt

# Session Traversal Utilities for NAT (STUN)



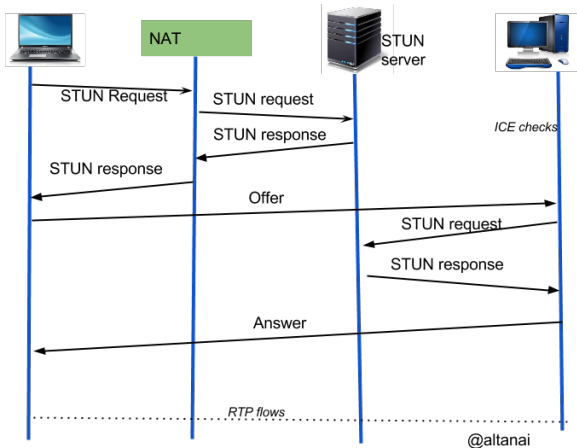
- Über STUN erfahren WebRTC-Clients externe IP+Port
- IP+Port werden dann an den anderen Client weitergeleitet

# Traversal Using Relays around NAT (TURN)



- TURN-Server leiten Verkehr weiter
- Fallback, wenn über STUN keine Verbindung hergestellt werden kann

# Vereinfachter STUN Nachrichtenaustausch



Quelle: <https://telecom.altanai.com/tag/sdp/>

- Hauptobjekt: `RTCPeerConnection`
- API umfangreich und zu Beginn unübersichtlich
- Es gibt mehrere Libraries, die die Komplexität abstrahieren
  - **PeerJS**
  - `simpleWebRTC`
  - `holla`
  - `simple-peer`
  - ...



# PeerJS Library

---

Webpack

WebRTC Grundlagen

PeerJS Library

Aufgabe 3

- **Vereinfacht** die Nutzung von WebRTC (und ICE)
- Implementierung für Node.js und Browser
- Unterstützung für **Voice/Video Streams**
- Unterstützung für **DataChannels**
- Open Source: <https://github.com/peers>
- Unterliegendes Signaling-Protokoll **frei wählbar**
- **Programmierende sind für den Austausch der Peer-IDs zuständig**

# PeerJS Beispiel

- Importieren der PeerJS-Bibliothek (über Webpack)
- Instanziierung des Peer-Objektes
- Peer-Objekt enthält die ID (`peer.id`)

---

```
1 import Peer from 'peerjs';  
2 const peer = new Peer();
```

---

- Herstellen der Verbindung durch den zweiten Client

---

```
1 const conn = peer.connect('dest-peer-id');
```

---

- Erster Client erhält das Connection-Objekt (`conn`) über das Connection-Event

---

```
1 peer.on('connection', function(conn) { ... });
```

---

- `peer.on(event, callback)`  
Methode um Event-Listener einzurichten
  - `open`  
Wird ausgelöst wenn eine Verbindung aufgebaut wurde.
  - `connection`  
Wird ausgelöst wenn eine Datenverbindung zu einem anderen Peer aufgebaut wurde.
  - `close`  
Wird ausgelöst wenn der Peer beendet wurde.
  - `disconnected`  
Wird ausgelöst, wenn die Verbindung zwischen dem Peer und dem Signaling-Server unterbrochen wurde.
  - `error`  
Wird ausgelöst wenn ein Fehler aufgetreten ist. Fehler sind immer kritisch und beenden den Peer.

## Aufgabe 3

---

Webpack

WebRTC Grundlagen

PeerJS Library

**Aufgabe 3**

- Ausliefern des Char-Clients als Bundle
  
- Implementierung eines „Secret Messaging Modus“(SMM)
  - Nachrichten direkt zwischen Clients ausgetauscht



- Anpassung bisheriger Lösung auf lokale Namespaces
  - Insbesondere Giphy-Integration
  - ⇒ Sinnvoller Ansatz:
    - `giphy.js`: Anfragen von Gifs an die Giphy-API
    - `client.js`: Einfügen von Gifs in den DOM
  
- Bundling von JS und CSS Code mit Webpack

- SMM Nachrichten direkt zwischen Clients aus
  - ⇒ WebRTC
    - Keine Speicherung auf Server
- IDs für WebRTC Verbindung über Server ausgetauscht
- Behandlung von Verbindungsabbrüchen

- Slider zum (De-)Aktivieren von SMM
- Geheime Nachrichten hervorgehoben
- CSS wird gestellt!



- Password: *node-red admin init*