

# Möglichkeiten zur asynchronen Kommunikation zwischen Webbrowser und Server

Hauptseminar | Hendrik Wagner





# Wie kann ich meine Website “Live” machen?



# Wo ist Bedarf für asynchrone/bidirektionale Kommunikation im Web?

- Chats, Multiplayer-Games, Online-Aktienhandel...
- Moderne Webseiten und -applikationen sind meist interaktiv
  - Nicht alle Inhalte werden initial geladen
  - Client fordert Aktualisierung/Inhalte über HTTP an - kein Problem!
  - Was, wenn der Server Inhalte von sich aus aktualisieren will?
- HTTP unterstützt keine serverseitigen Aktualisierungen einer Website
  - Wie können wir trotzdem Inhalte bereitstellen/aktualisieren, ohne dass der Nutzer sie anfragt?



---

# Inhalt

Einleitende Fragestellung

Vorstellung der asynchronen Kommunikationsvarianten

Implementierungen

Live Demo

Auswertungen

Zusammenfassung

Diskussion



# Vorstellung der asynchronen Kommunikationsvarianten

# Was ist Polling?

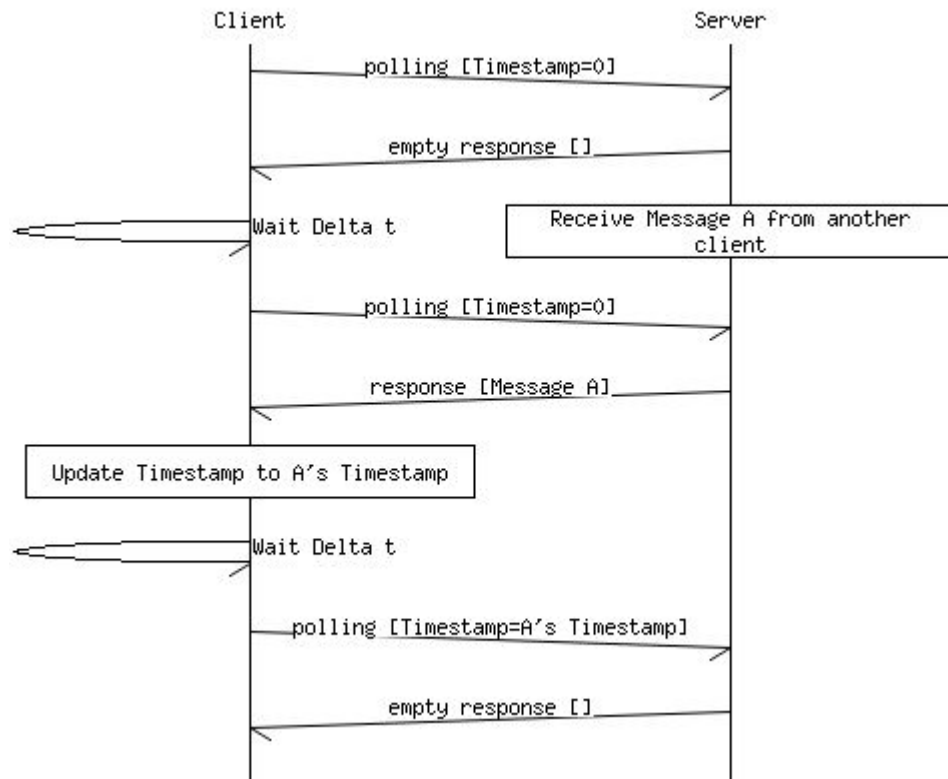


- Client lädt initial Inhalte vom Server
- Danach sendet Client im Intervall  $\Delta t$  eine Anfrage nach neuen Inhalten vom Server
  - Gibt es neuen Inhalt, wird dieser als Antwort übermittelt
  - Falls nicht, ist die Antwort leer



# Polling: Beispiel- kommunikation

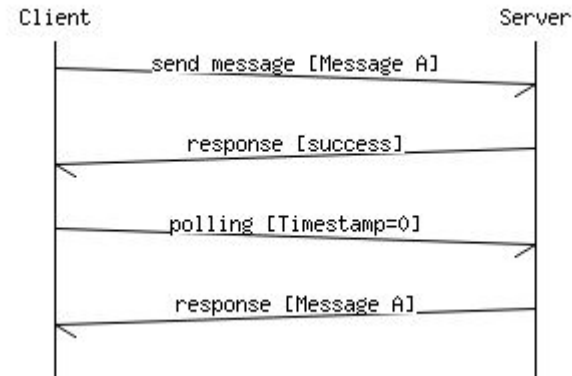
[Bildquelle: eigenes Werk]





# Polling-Varianten: Nachrichten- versand

[Bildquelle: eigenes Werk]





# Was ist Long Polling?



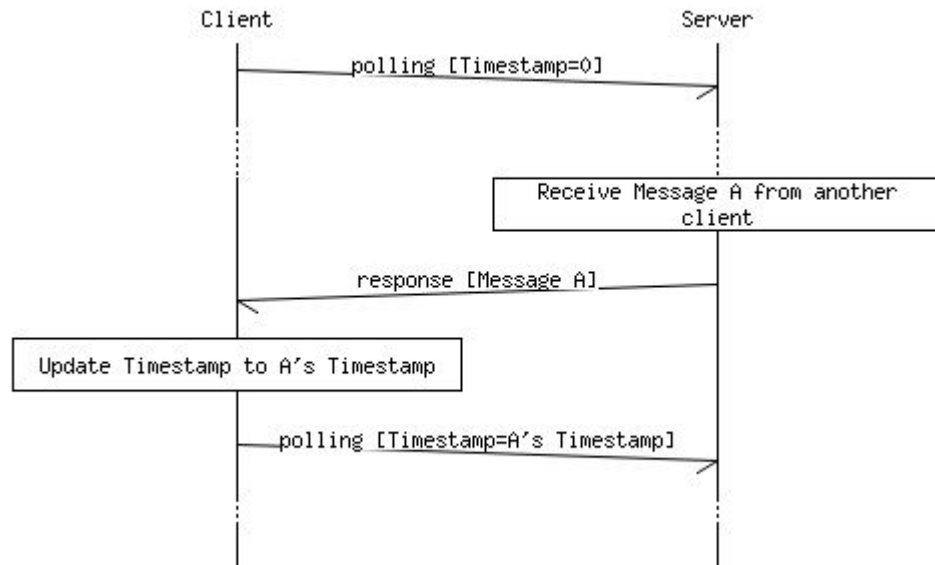
[Bildquelle: <https://tenor.com/en-GB/view/spongebob-waiting-lonely-gif-13695325>]

- Client lädt initial Inhalte vom Server
- Client fragt dann neue Inhalte an
  - Gibt es neuen Inhalt, wird dieser als Antwort übermittelt
  - Falls nicht, **wartet der Server mit seiner Antwort, bis Inhalte existieren!**
- Sobald der Client eine Antwort erhält, sendet er eine neue Anfrage



# Long Polling: Beispiel- kommunikation

[Bildquelle: eigenes Werk]



# Was ist Streaming?

A.k.a Comet oder Server Push

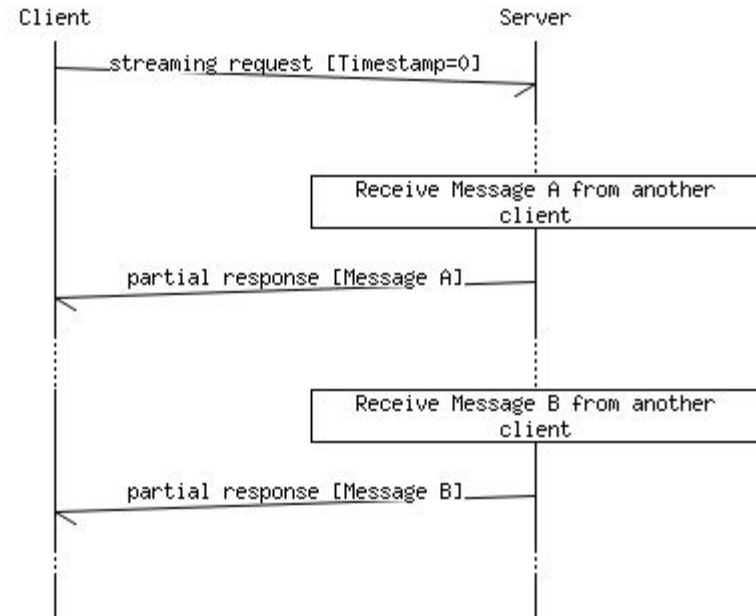


- Client lädt initial Inhalte vom Server
- Client fragt dann neue Inhalte an
  - Gibt es neuen Inhalt, wird dieser als **Teilantwort** übermittelt
  - Späteren Inhalt übermittelt der Server als weitere Teilantwort
- Die Anfrage des Clients wird nie als “beendet” vom Server markiert



# Streaming: Beispiel- kommunikation

[Bildquelle: eigenes Werk]





# Was sind WebSockets?

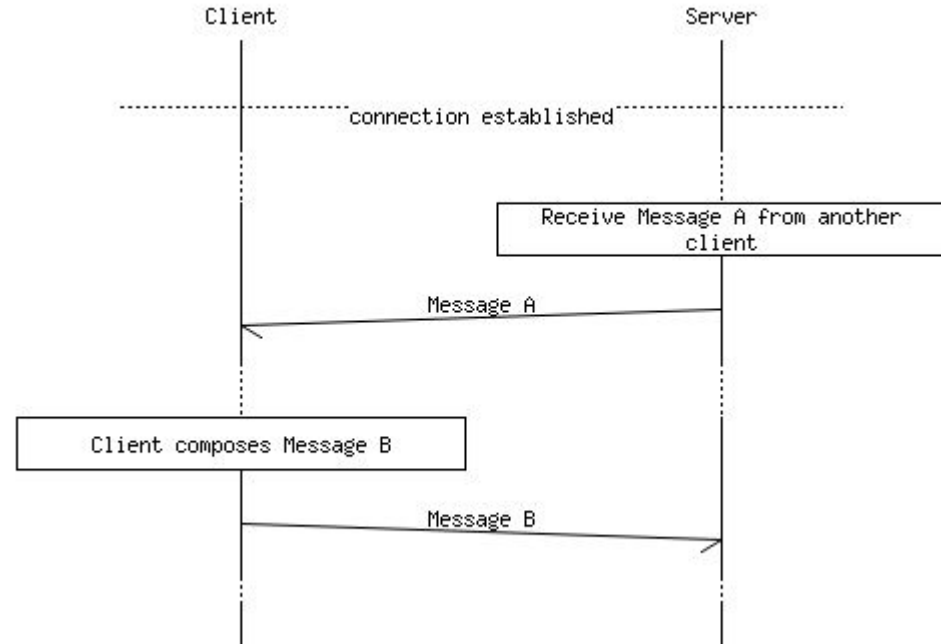


- Client sendet eine Anfrage, bei der er um Upgrade zum WebSocket Protokoll bittet
- Server akzeptiert und baut die Verbindung auf
- Jetzt können Server und Client wie auf TCP kommunizieren



# WebSockets: Beispiel- kommunikation

[Bildquelle: eigenes Werk]







# Implementierungen



# Nachrichtepakete



```
1 {  
2   content: "Hello World!",  
3   sender: "Alice",  
4   timestamp: "2022-12-04T19:29:13.455Z"  
5 }
```

- Wir verwenden für alle Kommunikationsvarianten eine einheitliche Datenstruktur
- Timestamp ist das Sendedatum
  - Wird verwendet, um Chatnachrichten zu sortieren und Clients zu ermöglichen, Folgenachrichten anzufragen



# Polling

```
1 const fetchMessages = async () => {  
2   const response = await fetch("...", {  
3     method: "POST",  
4     body: lastMessageTimestamp  
5   });  
6  
7   if (!response.ok) return;  
8  
9   appendMessages(await response.json());  
10  updateTimestamp();  
11 };  
12  
13 setInterval(fetchMessages, 2000);
```

[Vereinfachter Code]



# Live Demo: Polling





# Long Polling

```
1 const getResponse = async (timestamp) => {  
2   const filteredMessages = filter(timestamp,  
   messages);  
3  
4   if (filteredMessages.length > 0) {  
5     return filteredMessages;  
6   } else {  
7     waitingClients.add(res);  
8   }  
9 }
```

[Vereinfachter Code]



# Streaming

```
1 const receiveMessage = async (message) => {  
2   messages.push(message);  
3   res.end();  
4  
5   storedClients.forEach((client) => {  
6     client.write(JSON.stringify([message]));  
7   });  
8 }
```

[Vereinfachter Code]





# Live Demo: Long Polling & Streaming





# WebSockets

```
1 const sendMessage = async (msg: Message) => {
2   ws.send({
3     type: "message",
4     data: msg,
5   });
6 };
7
8 ws.onmessage = async (event) => {
9   const reader = new FileReader();
10  reader.readAsText(event.data);
11  appendMessages(reader.result);
12 };
```

[Vereinfachter Code]



# Live Demo: WebSockets





# Auswertungen

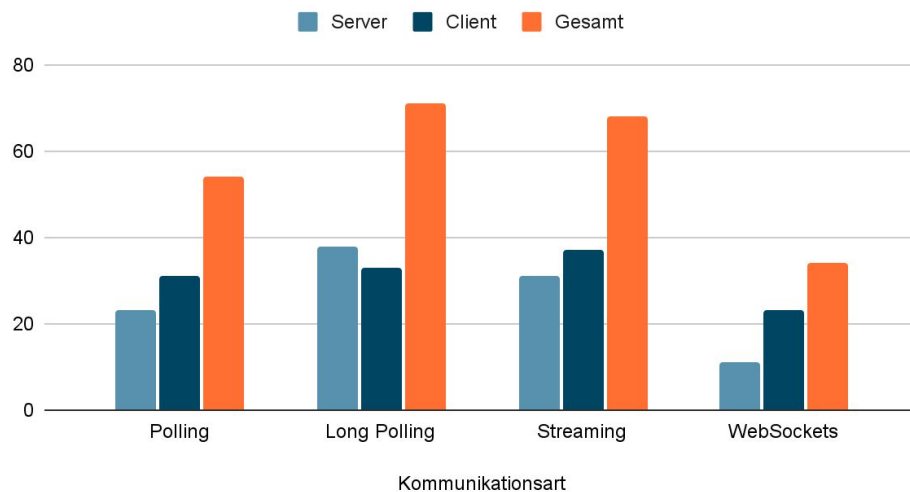


# Wie kann man die Varianten vergleichen?



# Implementierungsaufwand

Implementierungsaufwand in Zeilen Code



- Long Polling und Streaming am meisten Code
- WebSockets am schlanksten
- Begrenzte Bedeutsamkeit



## Performance: Was sendet wann wie viel?

Kommunikationsart	Initial	Regelmäßig	Nachrichtenempfang	Nachrichtenversand
Polling	- / -	711 B	$769 + M$ B	$714 + m$ B
Long Polling	- / -	- / -	$769 + M$ B	$714 + m$ B
Streaming	769 B	- / -	$m$ B	$714 + m$ B
WebSockets	722 B	- / -	$26 + m$ B	$26 + m$ B

- $m$  = Bytes einer Nachricht (inkl. Name und Timestamp)
- $M = m_1 + m_2 + \dots + m_n + n + 1$  = Bytes eines Arrays von Nachrichten



Und was soll ich jetzt verwenden?

**WebSockets...** *wahrscheinlich.*





# Das Problem mit Polling-Varianten

**Polling** ist einfach zu implementieren...

...belastet das Netzwerk aber übermäßig und ist keine Echtzeitübertragung.

**Long Polling** hat weniger Overhead...

...riskiert jedoch Timeouts und blockiert ggf. Systemressourcen.

**Streaming** hat fast keinen unnützen Overhead...

...riskiert aber zu Timeouts und Ressourcen auch Probleme mit Proxys.

**Und sie alle verwenden HTTP auf eine Weise, die das Protokoll nicht vorsieht.**



# Auch WebSockets sind nicht perfekt

## Die Vorteile:

- Für unseren Gebrauchsfall entwickelt
- Kein Overhead
- Geringe Client/Serverlast
- Echtzeitübertragung
- Geringere Latenz als Polling-Varianten

## Ein Nachteil:

- Nicht alle Browser unterstützen Websockets
  - 1.7% aller Nutzer können keine Websockets verwenden

Was, wenn mein Anwendungsfall aber  
100% der Nutzer erreichen soll?

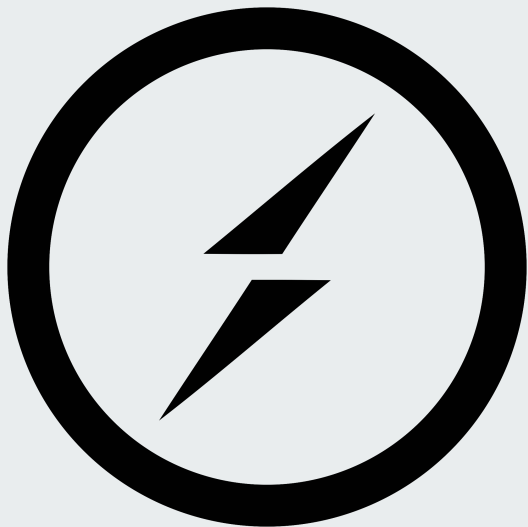
# Frameworks!

---



# Socket.IO

[<https://socket.io/>]



[Bildquelle: <https://socket.io/images/logo.svg>]

- Für eine Vielzahl von Frameworks implementiert
- Als Bibliothek in vielen Programmiersprachen verfügbar
- Baut auf Websockets auf, mit Long Polling als Fallback
- Bietet *Rooms*, in denen Nachrichten an Gruppen von Clients versendet werden
- Performance schlechter als Websockets (duh!)





# Faye

[<https://faye.jcoglan.com/>]



- Unbekanntere Alternative zu Socket.IO
- Weniger unterstützte Frameworks und Programmiersprachen
- Vergleichbare Features
- Fokus auf Publish-Subscribe Pattern



# Zusammenfassung

- Uns stehen **eine Reihe von Lösungsansätzen** zur Verfügung, wenn wir eine Website “Live” gestalten möchten
- Diese wurden auf verschiedenen Kriterien analysiert und verglichen
- Zusätzlich zu den klassischen Ansätzen können wir auf Frameworks ausweichen, um die Vorteile verschiedener Ansätze zu kombinieren





**Gibt es  
Anwendungsfälle, in  
denen Polling die beste  
Wahl sein könnte?**