

# Websockets:

## Leichtgewichtige Verbindungen für Web-Applikationen

- Stefan Neufeind
- Mit-Geschäftsführer der SpeedPartner GmbH aus Neuss ein Internet-Service-Provider (ISP)
  - Individuelle TYPO3-Entwicklungen
  - Hosting, Housing, Managed Services
  - Domains / Domain-Services
  - IPv6, DNSSEC, ...
- Aktive Mitarbeit im Community-Umfeld (PHP/PEAR, TYPO3, Linux)
- Freier Autor für z.B. t3n, iX, video2brain, ...

### Klassischer Fall für eine direkte http-Anfrage

- 1) Verbindung aufbauen
- 2) Anfrage senden



```
GET /2013/ HTTP/1.1
Host: www.linuxtag.org
Accept-Language: de,en;q=0.7,en-us;q=0.3
Accept-Encoding: gzip, deflate
[...]
```

```
HTTP/1.1 200 OK
Date: Sat, 25 May 2013 14:01:00 GMT
Content-Length: 245
Content-Type: text/html; charset=utf-8
[...]
```

- 3) Antwort erhalten
- 4) Verbindung abbauen

**Einschränkungen bei klassischem http/https:**

- Reines Anfrage-Antwort-Modell
- Kommunikation muss durch den Client ausgelöst werden
- Server kann nur eine einzige, unmittelbare Antwort liefern
- Verzögerungen durch z.B. Verbindungsaufbau / -abbau

**Wünschenswert:**

- Flexibilität
  - Kommunikation durch Client oder Server
  - Gesteuert durch Anforderungen (z.B. Ereignisse) statt Anfrage-Antwort
- Effizienz
  - Geringer Overhead
  - Vermeidung/Reduzierung von Verzögerungen

**Lösungsansätze:**

- Polling
- Mehrteilige HTTP-Antworten („multipart“)
- Keepalive
- Alternative Techniken/Protokolle (???)

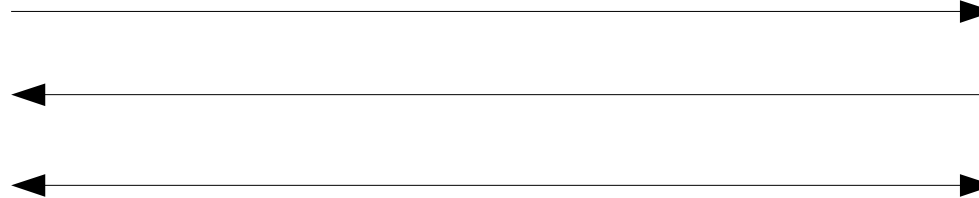
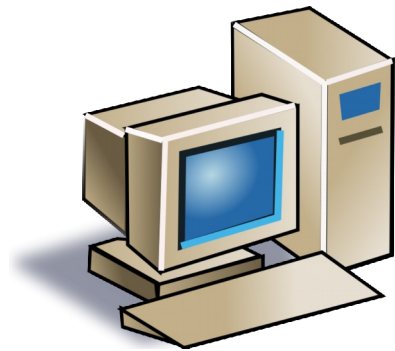
**Lösungsansätze:**

- Polling
  - Neuladen durch Client initiiert
  - ggf. in einem frame/iframe oder per AJAX
- Mehrteilige HTTP-Antworten („multipart“)
  - Server hält Verbindung offen und behält die Steuerung
  - ggf. in einem frame/iframe
  - Bei kleinen Datenmengen/Nachrichten großer Overhead
- Keepalive
  - Nachwievor einzelne Requests notwendig
  - Reduziert jedoch den Overhead für Verbindungsaufbau etc.
- Alternative Techniken/Protokolle
  - Nutzung von z.B. Java oder Flash
  - Kommunikation an http/https „vorbei“

**Verlauf einer WebSocket-Verbindung**

- 1) Verbindung aufbauen
- 2) Anfrage senden

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key:
  dGhllHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, talk
Sec-WebSocket-Version: 13
```



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
  S3pPLMBiTxaQ9kYG
  zzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

- 3) Antwort erhalten
- 4) Nutzung WebSocket-Kanal
- 5) Verbindung abbauen

### Verlauf einer WebSocket-Verbindung

Nach Request Upgrade der  
Verbindung auf direkte Kommunikation

Eindeutiger Key pro Request,  
üblicherweise UUID (128-bit)

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key:
  dGhllHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, talk
Sec-WebSocket-Version: 13
```

### Bestätigung des Upgrade



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
  S3pPLMBiTxaQ9kYG
  zzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Accept-Antwort durch Verknüpfung einer  
festen GUID mit dem eindeutigen Key generiert

```
function key2accept($key) {
  $guid = '258EAF5A5-E914-47DA-95CA-C5AB0DC85B11';
  $composed = $key . $guid;
  $hashed = sha1($composed, TRUE);
  return base64_encode($hashed);
}
```

### Verbindungsaufbau:

- Protokollprefixe „ws://“ (http, Port 80, unverschlüsselt) und „wss://“ (https, Port 443, verschlüsselt)
- Falls Proxy-Konfiguration auf Client erkannt Verwendung von „HTTP CONNECT“ für Tunnel
- Proxy-Support:
  - https generell problemfreier als http (keine Prüfung der TLS-Inhalte durch Proxy)
  - http erfordert für „HTTP Upgrade“ evtl. spezielle Konfiguration
  - Sobald „HTTP Upgrade“ erkannt Durchleitung von Datenverkehr (passthrough, pipe)
  - Nginx 1.4.0 (4/2013): WebSocket-Support wird als „Feature“
  - Varnish: Konfiguration einer Durchleitung per „pipe“ ausreichend (<https://www.varnish-cache.org/docs/3.0/tutorial/websockets.html>)



## Protokollversionen beachten:

- Hixie-75: 2/2010  
Chrome 4, Safari 5.0.0
- Hixie-76, hybi-00: 5/2010  
Chrome 6, Safari 5.0.1, Firefox 4 + Opera 11.00 (standardmäßig deaktiviert)
- Hybi-07: 4/2011  
Firefox 6
- Hybi-10: 7/2011  
Firefox 7 (PC + Android), Chrome 14
- RFC 6455: 12/2011  
IE 10, Firefox 11 (PC + Android), Chrome 16, Safari 6, Opera 12.10

Quelle Übersicht: <http://en.wikipedia.org/wiki/WebSocket>

- Versionen untereinander teils stark inkompatibel (Header, Keys, ...)

**Unterschiedliche Ansätze:**

- Websockets:  
Kommunikation zwischen Client und Server
  - WebRTC:  
Realtime-Kommunikation zwischen Clients (P2P)
- 
- Bereits in 2010: Demo für Client-Server-Videokonferenz per MediaStreamTransceiver
  - Bisher keine allgemeine Lösung für Audio/Video per Websockets verfügbar
  - WebRTC kann Websockets zum Verbindungsaufbau und Statusaustausch nutzen

**Native Nutzung per JavaScript möglich:**

```
try{
  var socket;
  var host = "ws://localhost:8000/socket/server/startDaemon.php";
  var socket = new WebSocket(host);

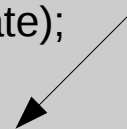
  message('<p class="event">Socket Status: '+socket.readyState);

  socket.onopen = function(){
    message('<p class="event">Socket Status: '+socket.readyState+' (open)');
  }

  socket.onmessage = function(msg){
    message('<p class="message">Received: '+msg.data);
  }

  socket.onclose = function(){
    message('<p class="event">Socket Status: '+socket.readyState+' (Closed)');
  }
} catch(exception){
  message('<p>Error'+exception);
}
```

0 = CONNECTING  
1 = OPEN  
2 = CLOSED



Quelle: <http://net.tutsplus.com/tutorials/javascript-ajax/start-using-html5-websockets-today/>

**z.B. per Socket.IO:** <http://socket.io/>

- Server (Node.js) sowie Client
- Fallbacks-Lösungen für auch für ältere Browser
  - WebSocket
  - Adobe Flash Socket
  - AJAX long polling
  - AJAX multipart streaming
  - Forever Iframe
  - JSONP Polling
- Breite Browser-Unterstützung:
  - Internet Explorer 5.5+
  - Safari 3+
  - Google Chrome 4+
  - Firefox 3+
  - Opera 10.61+

**z.B. per Wrench:** <http://wrench.readthedocs.org/en/latest/>

- Server (PHP)
- Client nativ

```
#!/usr/bin/env php
<?php

require(__DIR__ . '/lib/SplClassLoader.php');

$classLoader = new SplClassLoader('Wrench', __DIR__ . '/lib');
$classLoader->register();

$server = new \Wrench\Server('ws://0.0.0.0:8080/', array(
    'allowed_origins' => array('bohuco.net'),
));

$server->registerApplication('echo', new \Wrench\Application\EchoApplication());
$server->run();
```

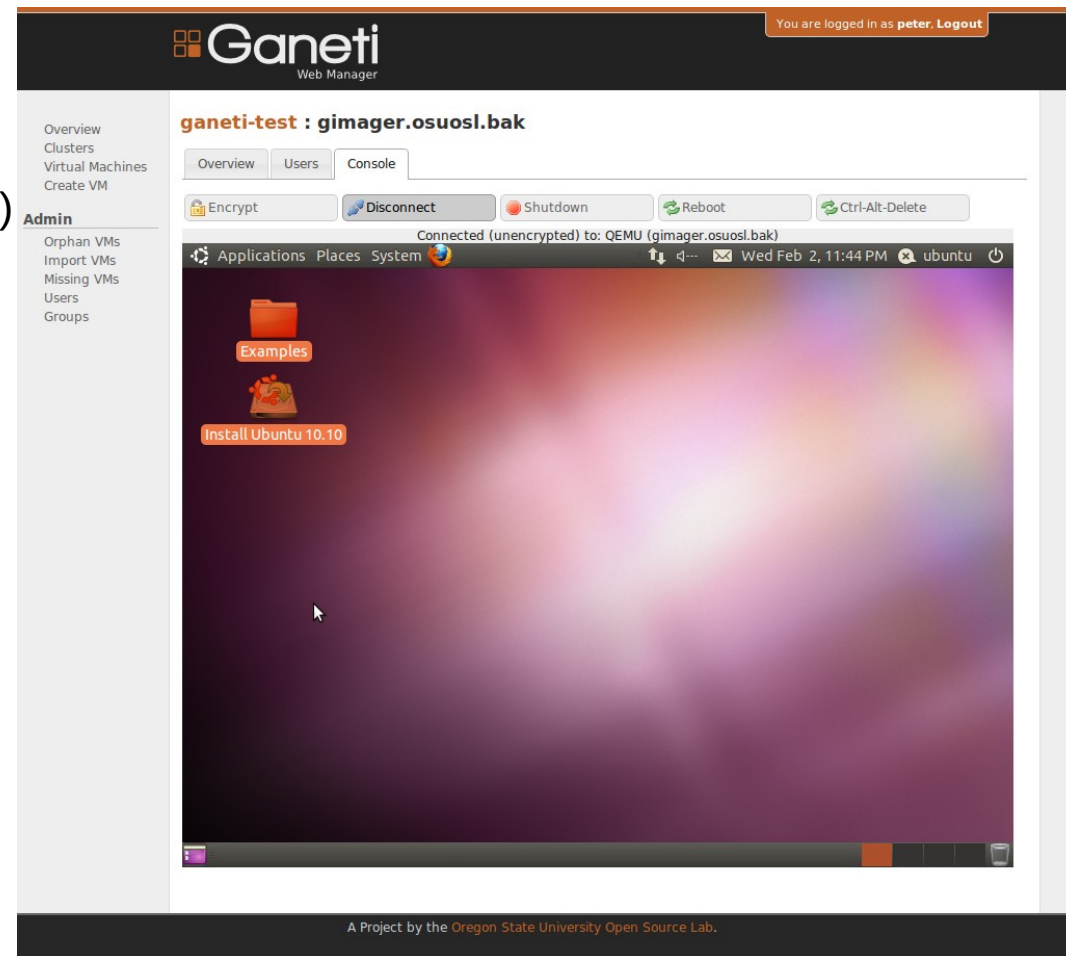
Quelle Übersicht: <http://bohuco.net/blog/2012/12/wrench-the-new-html5-websocket-class-hero-for-php/>

**z.B. per Ratchet:** <http://socketo.me/>

- Server (PHP)
- Client nativ; Flash-Fallback z.B. per web-socket-js (<https://github.com/gimite/web-socket-js>)

noVNC: <http://kanaka.github.io/noVNC/>

- Nutzung von Websockets und Canvas
- Verschlüsselte Übertragung
- Keine Plugins o.ä. erforderlich
- Verwendung z.B. innerhalb des Ganeti Web Manager (<https://code.osuosl.org/projects/51/wiki/VNC>)



### **Vorteile:**

- Flexible Möglichkeit zur Kommunikation
- Geringe Datenmenge
- Geringe Last für Server und Client (im Gegensatz zu Polling)
- Direkte Möglichkeit für Rückmeldungen durch den Server (dauerhafte Verbindung)

### **Kritik:**

- Browser-Unterstützung noch nicht „überall“ gegeben
- Unterschiedliche Protokollversionen (ältere Browser)
- Security bei „Passthrough“-Verbindungen (?)

### **Lösungen/Kompromisse:**

- Nutzung von Frameworks mit „Fallback-Lösungen“
- Nutzung in kontrollierter Client-Umgebung (Intranet, ...)

Danke fürs Zuhören  
sowie  
viel Erfolg beim Testen

Link zu den Slides: <http://talks.speedpartner.de/>

Bei Fragen stehen wir selbstverständlich gerne zur Verfügung:

Stefan Neufeind, [neufeind@speedpartner.de](mailto:neufeind@speedpartner.de)  
SpeedPartner GmbH, <http://www.speedpartner.de/>