Is WebSockets the future of the World Wide Web?

The Real Time Web



Kristian Johannessen  
1.4.2013

# Abstract

Contents

[Abstract i](#_Toc346781683)

[1 Introduction 1](#_Toc346781684)

[1.1 HTTP 1.0 1](#_Toc346781685)

[1.2 HTTP 1.1 1](#_Toc346781686)

[2 The Real Time Web with HTTP 1](#_Toc346781687)

[2.1 Polling 2](#_Toc346781688)

[2.2 Long-Polling 2](#_Toc346781689)

[2.3 HTTP Streaming (Forever Frame) 2](#_Toc346781690)

[2.4 Server-Sent Events 2](#_Toc346781691)

[3 WebSockets 3](#_Toc346781692)

[3.1 The protocol 3](#_Toc346781693)

[3.2 The API 3](#_Toc346781694)

[4 Discussion 3](#_Toc346781695)

[4.1 HTTP was never designed for real time web 3](#_Toc346781696)

[5 Conclusion 3](#_Toc346781697)

# Introduction

The World Wide Web has been available to the masses for 20 years [history of the web], and is still considered as a young technology. In the mid 90s, people even thought that the World Wide Web was just some fashion and that it would slowly die out [internett er en flopp!]. Those peoples believes are probably not the same today as the Web 2.0 is powering the World Wide Web into a new era. But the web today is a long way away from what it was 20 years, or even 10 years ago, and those are the aspects I will be addressing in this essay.

## HTTP 1.0

* http 1.0 kom i internettets barndom
* Nettsider var i stor grad tekstbaserte, eller hadde i det minste ikke mye objekter i seg
  + Grunnet mye bruk av forskere osv
* Flere og flere begynte dog med bilder og annet innhold for å dekorere.
* CSS fantes ikke på denne tiden
* Videre var ikke http 1.0 designet for å håndtere mange objekter i et html dokument[Network performance tingen]
* http 1.0 gjorde én tcp request (med åpning og lukking) pr. objekt
  + Hadde derfor ofte flere TCP connections oppe (på dial up!)
  + Trengte en forbedring

## HTTP 1.1

* Bruker persistent connections
* http 1.1 design goals fra Network performance tingen side 2
* Lar flere objekter hentes ned med en og samme TCP request før den lukkes
* 1.1 har mulighet for å motta deler av et objekt, lagre det i cache og så fortsette der den slapp ved f.eks disconnect.
  + Chuncked encoding (se the forever-frame technique og ref til protokollen)
* Introduserte upgrade request headeren for å gi mulighet til å oppgradere til en tenkt fremtidig oppgradering. Meget fremtidsrettet og smart! Handshake WebSockets [key differences]
* Vise til den artikkelen som indikerer at 1.1 var et relativt stor steg opp.
* Men ikke så stort som neste

Forklare om html 1.0 og 1.1, samt AJAX teknologier som kom i forhold til å hente data. Få frem problemstillingen rundt om Websockets er det nye bra og kule.

Bør fokusere en del på begrensningene i http. Få frem at alle de andre benytter http. Undersøk om du finner noe mer i forhold til store datamengder. Store datamengder er dog ikke aktuelt for real time apper uansett..

# The Real Time Web with HTTP

* Hva vil real time web si?
* Oppdatert innhold til alle klienter med en gang de skjer.
* Steg en: AJAX
  + Gjorde at man kunne oppdatere innhold uten å laste en hel side på nytt
* Applikasjoner som drar nytte av (er nærmest avhenig av) Real Time: Aksjetikkere, auksjonshus, Facebook, Twitter osv..
* Http ble ikke designet for realtime, men det finnes diverse måter å oppnå det (nesten)

## Polling

* Enkleste form for å hente data. Foregår ved at klienten gjør en request etter ny data. Serveren svarer med en gang enten med ny data eller en tom request [Stream updates with...].
* Kan benyttes til real time, men er oftest siste fallback instans.
* Må gjøres ved et høyt intervall hvis det er ukjent når ny data ankommer.
* Nyttig i applikasjoner hvor man vet i hvilket intervall man får nye data. En værapplikasjon for eksempel.

## Long-Polling

* Ha long-polling og Streaming under COMET? Evt til 2. utkast
* Som polling, men drar nytte av keep-alive muligheten i http 1.1
* Hvis det ikke er noen ny data, holdes connectionen åpen til den
  + Timer ut. Klienten blir da bedt om å koble seg på igjen av serveren
  + Det er data som så sendes over connectionen.
* Bruker mer server ressurser
* Hvis data oppdateres hyppig er det ikke stor forskjell på long-polling og vanlig polling siden serveren aldri får tid til å holde på en request [Html web sockets: quantum].

## HTTP Streaming (Forever Frame)

* Gammelt - Netscape 1992 [A comparison of push and pull tech for ajax]
* Page streaming
  + Streamer innhold i en lengelevende TCP connection.
  + Serveren sender til klienten når en tilstandsendring oppstår.
  + Browseren må oppdatere innhold samtidig som den lytter etter flere endringer.
* Service streaming
  + En XMLHttpRequest som holdes åpen.
  + Mer fleksibilitet i forhold til lengde på connection og hvor ofte nye requests skal gjøres.
* Mest vanlig er å benytte en iFrame for å simulere server push via en åpen TCP connection
* Kalles en forever frame fordi det er en iframe som er uendelig lang grunnet bruk av chuncked encoding (implisitt uendelig).
* Fjerner mye overhead ved å drive å sette opp og ta ned TCP connections hele tiden
* Scripts som sendes til en iFrame utføres umiddelbart.
* Man må håndtere flushing sånn at innholdet ikke blir veldig stort.

## Server-Sent Events

* Beveger oss over i Web 2.0 land - HTML5
* Gir serveren mulighet til å sende data til klienten uten at klienten har gjort en request.
* Benytter Http 1.1.
* Få inn i en diskusjon under konklusjonen eller websockets avsnittet (eller eget?) hvordan SSE kanskje er bedre i de tilfellene du egentlig ikke trenger en toveis kanal.
* Tilbys et eget API gjennom HTML5
* Har automatisk reconnect. Når connectionen er lukket, vil browseren koble seg til igjen etter en viss tid (3 sekunder er standard).
* Man kan også spesifisere at klienten ikke skal koble seg på igjen med Http response 204 [http://dev.w3.org/html5/eventsource]
* Håndterer beskjeder fra serveren med events i browseren. Kan lage egne av disse, noe som gir stor fleksibilitet.
* Kan sette ID på hvert event.
  + Lar browseren benytte Http header "Last-Event-ID" for å koble seg på igjen og fullføre forrige handling [http://www.html5rocks.com/en/tutorials/eventsource/basics/]
* Stor ulempe: proxy'er og brannmurer kan buffere responsen (siden den er så lang), noe som skaper latency [http://www.websocket.org/quantum.html]

# WebSockets

* TCP basert
* Full-duplex på en og samme connection

## The protocol

* Ble laget fordi HTTP ikke er egnet for full-duplex [http://datatracker.ietf.org/doc/rfc6455/?include\_text=1]
  + Mye overhead i HTTP-headere
  + Måtte kanskje benytte flere TCP-connections
  + Tidligere nevnte teknologier holder ikke mål
  + Mottatt data kunne være utadert pga latency allerede ved mottak
* Skal være minimal - ønsker ikke overhead!

"Basically it is intended to

be as close to just exposing raw TCP to script as possible given the

constraints of the Web."

* Designet for å være bakoverkompatibel - dyktig design av http 1.1 med upgrade header!
* Handshake
  + Hente figurer
  + Klienten ber om å oppgradere til WS
  + Hvis serveren sier ja, bryter HTTP connectionen ned og WS tar over på samme TCP/IP connection
* Benytter samme porter som HTTP og HTTPS (80 og 443)
* Kan sende både binærdata og tekst
* Skaper er "tunnel" gjennom brannmurer og proxier sånn at data ikke blir bufferet [ws.org]
* Frames
  + Kan sendes full-duplex begge veier samtidig
  + Har bare 2 bytes med kontrolldata i motsetning til HTTP-resonser som kan være flere 100 [http://www.websocket.org/quantum.html]
* Definerer en ny type URI: ws:// og wss://

## The API

* Slik som SSE er det definert et eget API
* Har diverse events og metoder for enkel tilkobling og frakobling
* Ser ikke ut til å støtte custom events på samme måte som SSE

# Discussion

Blabla bla

## HTTP was never designed for real time web

# Conclusion