Period-5 WebSockets/Real Time Communication

Name attributes of HTTP protocol makes it difficult to use for real time systems

Http er dårligt til "real time", da det er baseret på et request → response system. Hvert request og response indeholder meget boilerplate data, I form af headers.

```
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: W3 Total Cache/0.8
Pragma: public
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag: "pub1259380237;gz"
Cache-Control: max-age=3600, public
Content-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
X-Pingback: http://net.tutsplus.com/xmlrpc.php
Content-Encoding: gzip
Vary: Accept-Encoding, Cookie, User-Agent
```

Header eksempel

Alt denne header data, er ikke nødvendigvis interessant, hvis man f.eks laver et online spil. Der vil man måske gerne bare sige: "*spiller A flytter sig 1 unit ad x-aksen"* måske blot formuleret som [A, 0,1] eller noget lignede. Det siger sig selv at, hvis man blot er interesseret I at clienterne skal få adgang til ovenstående array (måske kommer der et nyt array for, HVER ENESTE tryk på en spillers piletast), giver det en kæmpe overhead at sende httpHeaders med hver gang (ovenstående array indeholder ca 10 bytes + eventuel housekeeping stuff, ovenstående http header fylder derimod næsten en kilobyte).

Med websockets kan man nøjes med, blot at sende et par bytes med til housekeeping, hver gang du pusher data til clienterne.

Explain polling and long-polling strategies, their pros and cons

Polling er et koncpet, der kan sammenlignes lidt med observer pattern, som vi kender det fra java, blot imellem aktører på et nætværk. Det foregår I praksis ved at, clienten spørger serveren om der er updates, hver gang den har mulighed for det. Eventuelle updates sendes så I et response til clienten.

Long polling er meget ligesom polling, I modsætning til polling (hvor clienten sender req(har du noget data) og serveren instantly sender res(nope, ikke endnu)), fungerer long polling dog således, at http connectionen holdes åben.
Det vil altså sige at clienten spørger om følgende req(har du noget nyt data), hvis svaret er nej, holder serveren da requestet åbent indtil der er noget data, således at følgende kan gælde:
req(noget data?) → arbitrært tidsrum indtil data er ready → res(her er din data(-:).

Long polling virker I min optik, til altid at være bedre end polling I forhold til performance.
Dog er long polling nok mere komplekst at implementere.

What is WebSocket protocol, how is it different from HTTP communication, what advantages it has over HTTP?

Websockets er I modsætning til http full duplex, det vil sige at de tillader bidirektionel kommunikation, fremfor envejs kommunikation (half-duplex) som http giver (req → res → req → res osv).

Derudover slipper websockets, som tidligere nævnt, uden om alt den overhead der er associeret med http headers. For at benytte sig af websockets, skal man sende en request header med en forspørgsel om at upgrade connectionen, sjovt nok virker det for mig, nærmere som om at det er en en downgrade af connectionen, fra http til det lavere liggende tcp-lag.

Accept-Encoding:gzip, deflate, sdch, br
Accept-Language:en-US,en;q=0.8,da;q=0.6
Cache-Control:no-cache
Connection*:*Upgrade
Cookie:io=R_yig1P_OIU_o81MAAAG
Host:localhost:3000
Origin:http://localhost:3000
Pragma:no-cache
Sec-WebSocket-Extensions:permessage-deflate; client_max_window_bits
Sec-WebSocket-Key:iwLpxNn+A75qsldmd1z29g==
Sec-WebSocket-Version:13
Upgrade:websocket
User-Agent:Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.76 Mobile Safari/537.36

"Eksempel på et request der beder om en upgrade fra http til websocket"

Websockets opfører pretty much ligesom en almindelig tcp-socket.

Explain what the WebSocket Protocol brings to the Web-world.

En måde at lave full-duplex kommunikation, uden brug af plugins.

Explain and demonstrate, using a package like Socket.io (on the Server), the process of WebSocket communication - From connecting client to server, through sending messages, to closing connection:

Server (socket.io npm pakken):
Når der bliver established en connection til serveren laves en socket, associeret med den nye connection

```
io.on("connection", (socket) =>
```

første parameter er det event som pakken skal lytte efter (her connection).

```
io.emit("update", voting.results);
```

Emit er et kald der sender til alle connectede sockets.

```
socket.on("disconnect", () =>
                {
console.log("a user disconnected!!");
                });
```

Ovenstående lukker forbindelsen.

Klienten:

Først skal vi bruge en socketFactory:

```
app.factory("mySocket", function (socketFactory, $location)
                {
        return socketFactory();
                });
```

Vi kan så bruge sockets I eksempelvis en angular ctrl:

```
app.controller("VoteCtrl", function ($scope, mySocket)
```

Da socketen kommer fra samme pakke som på serversiden, benytter den samme terminologi, hvorfor man kan gøre følgende:

```
mySocket.on("update", function (res)


    mySocket.emit("vote", vote);
```

Osv.

What's the advantage of using libraries like Socket.IO, Sock.JS, WS, over pure WebSocket libraries in the backend and standard APIs on frontend? Which problems do they solve?

Fordelen ved ovenstående pakker, er primært at de selv fortager en detection af klientens browser, hvorefter de benytter polling/long polling, hvis browseren er så gammel, at den ikke understøtter websockets. Da man gerne vil have at ens webpages rammer så stort et markede som muligt, er det nice at man ikke selv behøver stå for detection og håndtering af legacy browsere.