WebSockets

2014-09-17

Lars Tiede

Motivation

- Web apps communicate like this: clients sends request, server answers, communication is done.
  - —> server can not initiate communication, ever
- So what if a web app wants the server to communicate something to the client (“push”) without the client sending a request?
  - Old answer: cheat with long polling
  - New answer: use WebSockets

What are “WebSockets”?

- Protocol on top of TCP that provides a full-duplex *message* oriented communication channel: RFC 6455

- Connection initiated with a HTTP request, which is “upgraded” to a WebSocket connection

- W3C standard API: http://www.w3.org/TR/websockets/

- Designed to be implemented in web browsers and web servers, but can of course be used in other scenarios as well

Demo

Example codes

- Code (and slides) from the demo will be on source.uit.no

- Another example using Python on the server side: https://source.uit.no/star-apt/websocket-demo/tree/master

WebSocket != Socket

- A WebSocket implementation might use a TCP socket under the hood, but is itself something else.
- Consequence: it's not trivial to make browser frontends for legacy apps with WebSockets
  - Either need a server that supports WebSocket connections (for example most newer VNC servers)
  - Or wrap a legacy protocol in WebSockets: for example https://github.com/kanaka/websockify

Programming with WebSockets - server side

- General network programming: WebSockets have similar characteristics as sockets. Differences include:
  - Fewer options: comm is always message based, no UDP or raw IP sockets, no domain sockets…
  - naming scheme: use URIs instead of port names
  - HTTP: can get proxy traversal and encryption from your webserver
- Web programming: WebSockets break with the traditional web programming idea of answering a request quickly and then moving on.
  - Many popular web programming frameworks rely on the traditional request-response model and don’t work well with WebSockets.

Web programming models - server side

- Traditional web programming model: request handling routine answers a request very quickly, closes the connection and terminates. Common implementations:
  - one thread per request handler, with few threads running concurrently. Okay approach when request handlers don't run long or load is low. Often called *"blocking"* or *"threaded"*. (Example: Django)
  - many request handlers in one thread using non-blocking I/O. Good *if requests don't need much CPU time*. Often called *"asynchronous"* or *"event-driven"*. (Example: Tornado, Twisted, "anything" written in JavaScript)
- WebSockets don't fit: connection lives long, WebSocket I/O can happen at any time —> handling routine lives long and might do a lot of I/O. Implementation options:
  - one thread per request handler is only possible if we can run *many* threads concurrently. Only few systems can do this. (Example: Go)
  - Non-blocking I/O as in traditional model

Programming with WebSockets - client side

- Browser
  - JavaScript is the only option.
  - I/O is asynchronous.
- Other programs
  - WebSocket libraries are available for many languages.
  - Can likely choose between blocking and non-blocking implementations.

Programming with WebSockets - summary

- Server side
  - web programming: you need asynchronous I/O or very cheap threads, threaded frameworks don’t work.
  - general network programming: WebSockets have similar characteristics as sockets.
- Client side
  - web programming: in the browser, JavaScript’s asynchronous I/O is the only option.
  - general network programming: WebSockets have similar characteristics as sockets.

WebSockets > TCP Sockets ?

- WebSockets will allow you to include a browser into your design at any time without hassle.

- WebSockets on top of HTTP: URIs instead of port numbers, probably get proxy traversal and encryption cheaply with <your favourite HTTP server/reverse proxy>

- Your communication is message-based? WebSockets give you a general purpose message-based protocol for free.

- WebSocket client and server libraries are available for every major language, and WebSockets are not harder to use than sockets.

- So, for general network programming: consider using WebSockets instead of TCP sockets?

Bonus: WebSocket quirks (1)

- Specification has for a while been in flux, several different protocols (all issued by IETF) were floating around until things settled.
  - hixie-75, hixie-76, hybi-00, hybi-07, hybi-10
  - All of them were issued between 2010 and 2011.
- Development has since december 2011 settled on RFC 6455. But keep an eye open when you choose a library, not all of them implement all of RFC 6455.

Bonus: WebSocket quirks (2)

- r/w buffer sizes, max message length can be specified anywhere, you think? lol no
    - Every implementation will do different things, and you might have no way of even knowing what the actual limits are
- Control messages for connection keepalive exist (“ping” and “pong”), but APIs are not supposed to expose any of that
    - Again, every implementation will do different things and you don’t have any control over it

Thanks!