Node.js Vorstellung

… aber eigentlich geht es um etwas anders

Tim Becker tim@press-any-key.de
@a2800276

Node

- server-seitiges JavaScript
- stable & populär
- seit 2009
- ausgereifte Infrastruktur
- diverse “seriöse” Nutzer

Beispiel

```
var http = require('http');

var handler = function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n')
}

var server = http.createServer(handler)
server.listen(1337, "127.0.0.1");

console.log('Server running at http://127.0.0.1:1337/');
```

JavaScript ist irrelevant

- Das Wesen von Node is Event Driven IO.
- Um zu verstehen, was das bedeutet, muss man zuerst verstehen, wie web unter der Haube funktioniert.

Wie funktioniert HTTP?

Client:

```
GET /
```

Server:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 11 Nov 2014 11:35:27 GMT
Connection: close

Hello World
```

Demo

Wie funktioniert HTTP II? (Webframework Edition)

Client:

```
GET /
user_agent: Wget/1.15 (darwin13.2.0)
```

Wie funktioniert HTTP II? (Webframework Edition)

Client:

```
GET /
user_agent: Wget/1.15 (darwin13.2.0)
```

```
{ "GATEWAY_INTERFACE"=>"CGI/1.1",
  "PATH_INFO"=>"/",
  "QUERY_STRING"=>"",
  "HTTP_USER_AGENT"=>"Wget/1.15 (darwin13.2.0)"
  "REMOTE_ADDR"=>"127.0.0.1",
  "REMOTE_HOST"=>"localhost",
  "REQUEST_METHOD"=>"GET",
  "REQUEST_URI"=>"http://localhost:8080/",
  "SCRIPT_NAME"=>"",
  "SERVER_NAME"=>"localhost",
  "SERVER_PORT"=>"8080",
  "SERVER_PROTOCOL"=>"HTTP/1.1",
  "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/2.1.2/2014-05-08)",
  "rack.version"=>[1, 2],
  "rack.input"=>#<StringIO:0x007fb53b151da8>,
  …
```

Wie funktioniert HTTP II? (Webframework Edition)

Client:

```
GET /
user_agent: Wget/1.15 (darwin13.2.0)
```

```
{ "GATEWAY_INTERFACE"=>"CGI/1.1",
  "PATH_INFO"=>"/",
  “QUERY_STRING"=>"",
  "HTTP_USER_AGENT"=>"Wget/1.15 (darwin13.2.0)"
  "REMOTE_ADDR"=>"127.0.0.1",
  "REMOTE_HOST"=>"localhost",
  "REQUEST_METHOD"=>"GET",
  "REQUEST_URI"=>"http://localhost:8080/",
  "SCRIPT_NAME"=>"",
  "SERVER_NAME"=>"localhost",
  "SERVER_PORT"=>"8080",
  "SERVER_PROTOCOL"=>"HTTP/1.1",
  "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/2.1.2/2014-05-08)",
  "rack.version"=>[1, 2],
  "rack.input"=>#<StringIO:0x007fb53b151da8>,
  …
```

Wie funktioniert HTTP II? (Webframework Edition)

Client:

```
GET /
user_agent: Wget/1.15 (darwin13.2.0)
```

```
{ "GATEWAY_INTERFACE"=>"CGI/1.1",
  "PATH_INFO"=>"/",
  "QUERY_STRING"=>"",
  "HTTP_USER_AGENT"=>"Wget/1.15 (darwin13.2.0)"
  "REMOTE_ADDR"=>"127.0.0.1",
  "REMOTE_HOST"=>"localhost",
  "REQUEST_METHOD"=>"GET",
  "REQUEST_URI"=>"http://localhost:8080/",
  "SCRIPT_NAME"=>"",
  "SERVER_NAME"=>"localhost",
  "SERVER_PORT"=>"8080",
  "SERVER_PROTOCOL"=>"HTTP/1.1",
  "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/2.1.2/2014-05-08)",
  "rack.version"=>[1, 2],
  "rack.input"=>#<StringIO:0x007fb53b151da8>,
  …
```

Wie funktioniert HTTP II? (Webframework Edition)

Client:

```
GET /
user_agent: Wget/1.15 (darwin13.2.0)
```

```
{ "GATEWAY_INTERFACE"=>"CGI/1.1",
  "PATH_INFO"=>"/",
  "QUERY_STRING"=>"",
  "HTTP_USER_AGENT"=>"Wget/1.15 (darwin13.2.0)"
  "REMOTE_ADDR"=>"127.0.0.1",
  "REMOTE_HOST"=>"localhost",
  "REQUEST_METHOD"=>"GET",
  "REQUEST_URI"=>"http://localhost:8080/",
  "SCRIPT_NAME"=>"",
  "SERVER_NAME"=>"localhost",
  "SERVER_PORT"=>"8080",
  "SERVER_PROTOCOL"=>"HTTP/1.1",
  "SERVER_SOFTWARE"=>"WEBrick/1.3.1 (Ruby/2.1.2/2014-05-08)",
  "rack.version"=>[1, 2],
  "rack.input"=>#<StringIO:0x007fb53b151da8>,
  …
```

Wie funktioniert HTTP II? (Webframework Edition)

Server:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 11 Nov 2014 11:35:27 GMT
Connection: close

Hello World
```

Wie funktioniert HTTP II? (Webframework Edition)

Server:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 11 Nov 2014 11:35:27 GMT
Connection: close

Hello World
```

```
[200,
  {"Content-Type" => "text/plain",
   "Date" => “Tue, 11 Nov 2014 11:35:27 GMT”},
  [“Hello World”]
]
```

Wie funktioniert HTTP II? (Webframework Edition)

Server:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 11 Nov 2014 11:35:27 GMT
Connection: close

Hello World
```

```
[200,
  {"Content-Type" => "text/plain",
   "Date" => “Tue, 11 Nov 2014 11:35:27 GMT”},
  [“Hello World”]
]
```

Wie funktioniert HTTP II? (Webframework Edition)

Server:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 11 Nov 2014 11:35:27 GMT
Connection: close

Hello World
```

```
[200,
  {"Content-Type" => "text/plain",
   "Date" => “Tue, 11 Nov 2014 11:35:27 GMT”},
  [“Hello World”]
]
```

Das war Rails…

Gilt aber im wesentlichen auch für Python (WSGI), Java Servlets und andere Sprachen/Frameworks.

```
@Override
protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    getServletContext().log("service() called");
    count++;
    response.getWriter().write("Incrementing the count: count = " + count);
}
```

Das Problem

- Der Rest der Welt bleibt stehen, während ich einen Request bearbeite

- Kann man aber leicht mit Threads oder Prozessen lösen …

Das Problem II

- Thread verbrauchen viele Server-Resourcen
  - 1 bis 2MB - Thread
- Prozesse umso mehr

http://www.kegel.com/c10k.html

Das Problem III

- Der Rest der Welt bleibt stehen, während ich einen ~~Request bearbeite~~ warte

Non-Blocking / Async IO

https://gist.github.com/hellerbarde/2843375

http://norvig.com/21-days.html

Non-Blocking / Async IO

- immer nur dann arbeiten, wenn gerade was da ist:

```
data = read(); arbeiten(data)
```

vs

```
read (function (data){arbeiten(data)} )

// was anderes machen oder rumsitzen
```

noch mehr Probleme…

- Wie gehe ich mit bruchstückhaften Daten um?
  z.B.

- **`read()`** liefert mir nicht “**`GET /`**“ sondern “**`GE`**” weil gerade noch nicht mehr angekommen ist…

- http_parser.c

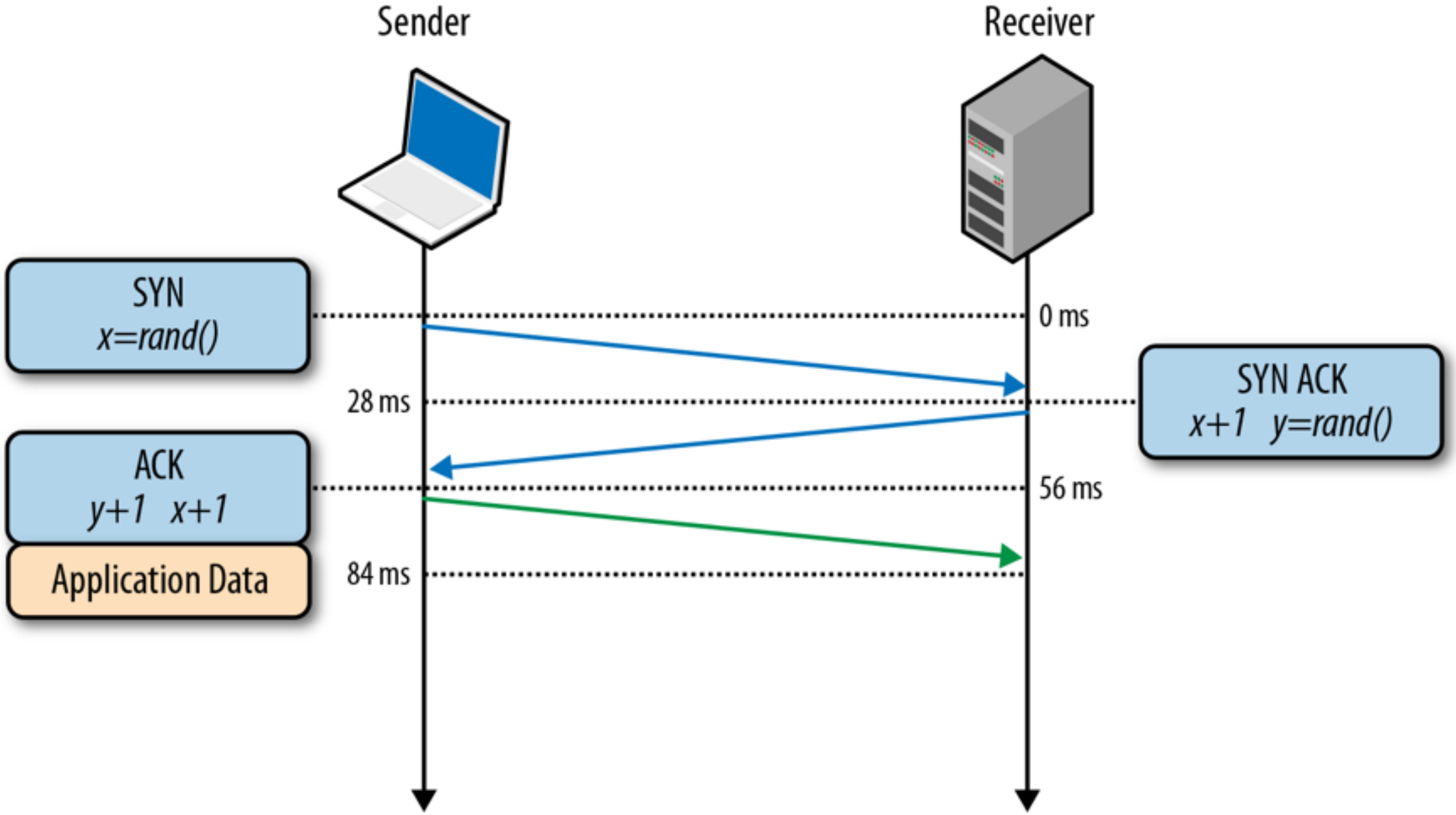
noch mehr Probleme

- welche Operationen blockieren?
- deswegen JavaScript.
  - JavaScript kennt keine blockierenden Funktionen
  - Entwickler sind Callbacks gewohnt (`onmouseover`)

Websockets & Co.

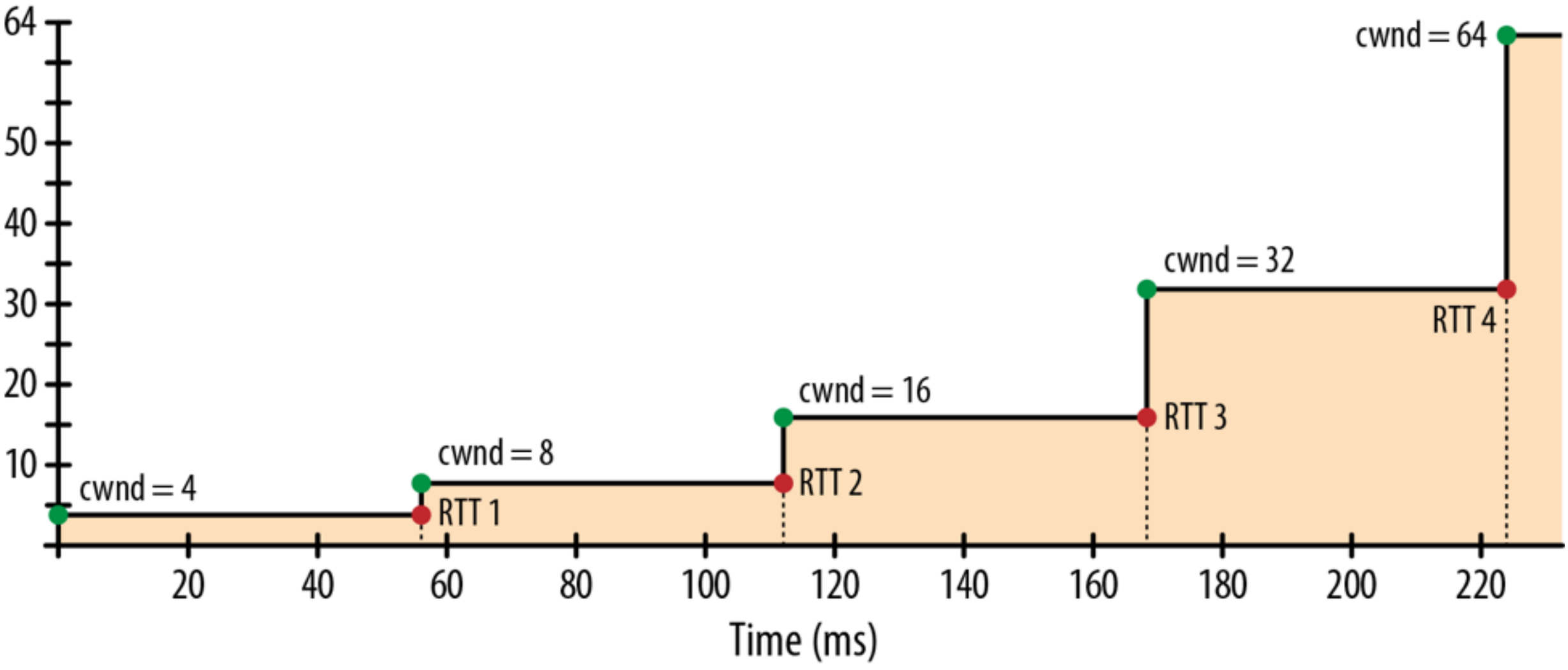
- 2004 : Type ahead search
- AJAX: long poll -> websockets
- Single Page Apps
- keep-alive
- HTTP2.0 / SPDY

Wie funktioniert TCP/IP? (3 Way Handshake)



High Performance Browser Networking Chap.2

Wie funktioniert TCP/IP? (Congestion Window)



High Performance Browser Networking Chap.2

Wie funktioniert TCP/IP?

Hello, would you like to hear a TCP joke?

Yes, I'd like to hear a TCP joke.

Ok, I'll tell you a TCP joke.

Ok, I will hear a TCP joke.

Are you ready to hear a TCP joke?

Yes, I am ready to hear a TCP joke.

Ok, I am about to send the TCP joke. It will be 144 words long, it has two characters, it does not have a setting, and it ends with a punchline.

Ok, I am ready to get your TCP joke that will be 144 words long, has two characters, does not have an explicit setting, and ends with a punchline.

I'm sorry, your connection has timed out. Hello, would you like to hear a TCP joke?

nützliches zu Node.js

- npm : package manager
- node-inspector : `https://github.com/node-inspector/node-inspector`
- express : web framework, usw. usf.

Alternativen

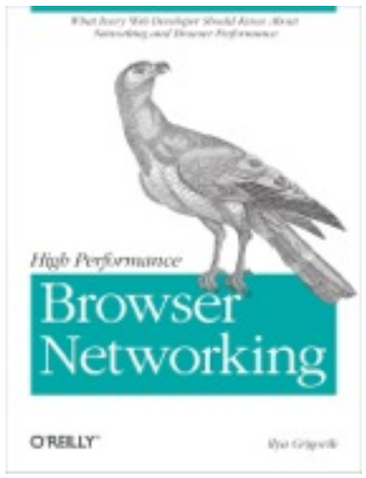
- Alternative Event Driven Frameworks
  - Ruby: EventMachine
  - Python: Twisted
  - usw.
- “Light-Weight Concurrency”
  - Go
  - Erlang

nützliches zu Netzwerk Performance

- Guillermo Rauch: 7 Principles of Rich Web Applications [http://rauchg.com/2014/7-principles-of-rich-web-applications/]

- Paul Irish: Delivering the Goods [https://docs.google.com/presentation/d/1MtDBNTH1g7CZzhwlJ1raEJagA8qM3uoV7ta6i66bO2M]

- Ilya Grigorik: High Performance Browser Networking [http://chimera.labs.oreilly.com/books/1230000000545/]



Fragen?

Danke.

tim@press-any-key.de
@a2800276