AMP, PWAs, HTTP/2 and Service Workers: A New Era of Web Performance?

Mobile Track
Who is talking today?

Felix Gessert

PhD Thesis
Web Performance
Cloud Data Management

CEO & Co-Founder
Baqend Platform
Speed Kit Plugin

Universität Hamburg

Baqend
Presentation is loading
Average: 9.3s

- **100 ms**
- **500 ms**
- **5 sec**

```
Loading...
```

- ✔️ **+80% Traffic**
- ⬇️ **-20% Traffic**
- ⬇️ **-1% Revenue**
Page Load Time

+80% Traffic = Money

-20% Traffic

-1% Revenue
What causes slow page loads?
There are 3 performance problems.
There are 3 performance problems.

Is this really a problem in practice?
How can we build FASTER websites?
What we are going to cover.

**Frontend**
- Google AMP
- Instant Articles
- Progressive Web Apps

**Network**
- HTTP/2
- Service Workers

**Backend**
- Cloud & NoSQL
- Speed Kit
## What is the goal?

<table>
<thead>
<tr>
<th>Delay</th>
<th>Perception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 100 ms</td>
<td>Instant</td>
</tr>
<tr>
<td>100 – 300 ms</td>
<td>Small perceptible delay</td>
</tr>
<tr>
<td>300 – 1000 ms</td>
<td>Machine is working</td>
</tr>
<tr>
<td>1+ s</td>
<td>Mental context switch</td>
</tr>
<tr>
<td>10+ s</td>
<td>Task abandoned</td>
</tr>
</tbody>
</table>

**Load time < 1s**

1. Frontend Performance
Frontend: Critical Rendering Rendering Path

```
<!doctype html>
<link href=all.css rel=stylesheet />
<script src=app.js ></script>
<body>
  <h1>Web Performance</h1>
</body>
```

elem.style.width = "50px";
document.write("test");

```
body { background-color: green; }
H1 { padding: 10px; }
```
Frontend: Critical Rendering Path
Best Practices

1. Minimize **Length** *(Round-Trips)*
2. Minimize **Size** *(Critical Resources)*
3. Minimize **Weight** *(Critical Bytes)*
Inlining critical CSS and JS “above the fold”

Load CSS first, JS last

Load non-critical CSS and JS asynchronously

Compress images

CSS & JS: UglifyJs & cssmin

Single-page application

Test your performance

PageSpeed Insights

Process html

Minify and concatenate CSS and JS
Google’s vision for a better web: AMP
Accelerated Mobile Pages (AMP)

How AMP works:

- **Stripped down** HTML + AMP tags (e.g. img) → rendered asynchronously by runtime
- **CSS** must be **inlined** + <50 KB
- **No custom JS** (except in iframes)
- Only static sizes → **no repaints**
- **No Forms** & only for **mobile**
- Pre-Loaded in **Google Results**
- **Cached** in Google CDN, as long as it is crawled the next time

https://www.ampproject.org/docs/reference/spec.html
Implementing AMP for a website

1. **Link** to AMP Version:

```
<link rel="amphtml" href="full-url-to-amp-version"/>
```

2. Use **HTML** Boilerplate:

```
<!doctype html>
<html ⚡ lang="en">
  <head> …
```

3. **AMP Tags**:

```
<amp-img src="logo.png" width="100" height="40">
```
AMP: the Good

- Fast Mobile Loads
- Google Result Carousel
- Works Well for Static Sites
AMP: the Bad

Google AMP Case Study – Leads Dropped by 59% (How to Disable It)

By Brian Jackson, Updated: September 17, 2017

- 59% Leads

- 17% Signups
Google May Be Stealing Your Mobile Traffic

October 15, 2016 by Alex Kras — 223 Comments
AMP: the Ugly↓

Kill Google AMP before it KILLS the web

By Scott Gilbertson 19 May 2017 at 08:25

- No JS: only Google’s
- Bad UX: iOS Scrolling
- No Custom Analytics
- Assumes Dumb Devs
AMP: the Ugly

Google AMP is bad for E-commerce

By Lesley Paone | August 16th, 2017 |

No Chat or Payment

No Search or Login
Google AMP is bad for E-commerce.

AMP started as a good idea but it is too limiting.

No Chat or Payment

No Search or Login
Facebook’s Alternative: Instant Articles
Facebook Instant Articles

- Single **HTML Document**
- **No CSS/JS**
- **Designed** in FB Editor
- **Crawled** from RSS Feed

```html
<head>
  <meta property="op:markup_version" content="v1.0">
  <!-- The URL of web version-->
  <link rel="canonical" href="http://example.com/article.html">
  <meta property="fb:article_style" content="myarticlestyle">
</head>
<body>
  <article> ... </article>
</body>
```
Instances Articles: the Good

Fast Mobile Loads

Good UX for Facebook Users
INSTANT RECALL

Facebook's Instant Articles promised to transform journalism — but now big publishers are fleeing

by Casey Newton | @CaseyNewton | Apr 16, 2017, 11:01am EDT

FB makes the rules

Users stop visiting real site
Why not apply the *good* ideas to any website?

**Progressive Web Apps**
What are Progressive Web Apps?
Try this:

codetalks.baqend.com
Progressive Web Apps (PWAs)

- Fast **Loads** through Caching
- **Offline** Mode (Service Workers)
- Add-to-Homescreen and **Push**
Implementing PWAs

- PWAs are best practices not a technology
- Progressively enhance when supported

1. **Manifest** declares Add-to-Homescreen:

```html
<link rel="manifest" href="/manifest.json">
{
    "short_name": "Codetalks PWA",
    "icons": [
        {"src": "icon-1x.png", "type": "image/png", "sizes": "48x48"}],
    "start_url": "index.html?launcher=true"
}
```
Implementing PWAs

- PWAs are best practices, not a technology
- Gracefully degrade when not supported

2. **Service Workers** for caching & offline mode:

![Diagram showing web app connecting to website, SW.js, cache, and network]
Implementing PWAs

- PWAs are best practices not a technology
- Progressively enhance the user experience

3. Add Web Push and Background Sync:

[Diagram showing interaction between Web App, Website, SW.js, Sync, and Push.]
Typical Architecture: **App Shell Model**

**App Shell**: HTML, JS, CSS, images with app logic & layout

**Content**: Fetched on demand & may change more often
Why **PWAs** over AMP & Instant Articles?

- **Independent Technology**
- **Work across Devices**
- **No Restrictions on Development**
Why PWAs over AMP & Instant Articles?

What is the future of Progressive Web Apps?

Independent Technology

Work across Devices

No Restrictions on Development
The Future of **PWAs** is bright.

### Payment Request API

- **Goal:** replace traditional **checkout** forms
- **Just ~10 LOC for a payment**
- **Vendor- & Browser-Agnostic**
The Future of **PWAs** is bright.

**Credentials Management API**

1. Click **Sign-in** → Native Account Chooser
2. Credentials API stores information for future use
3. **Automatic** Sign-in afterwards
The Future of **PWAs** is bright.

**Web Speech API**

Native Speech Recognition in the Browser:

```javascript
annyang.addCommands({
    'Hello Code.talks': () => {
        console.log('Hello you.');
    }
});
```
The Future of **PWAs** is bright.

**Web Share API**

- **Share** site through native share sheet UI
- Service Worker can register as a **Share Target**
2. Network Performance
Network Performance in a Nutshell

**DNS Lookup**
- Every domain has its own DNS lookup

**Initial Connection**
- TCP makes a three-way handshake $\rightarrow$ 2 roundtrips (1 with TCP Fast Open)
- SSL connections have a more complex handshake $\rightarrow$ +2 roundtrips (only 1 with TLS False Start or Session Resumption)

**Time to First Byte**
- Depends heavily on the distance between client and the backend
- Includes the time the backend needs to render $\rightarrow$ Session lookups, Database Queries, ...

**Content Download**
- New connections are slow (initial congestion window is small) $\rightarrow$ many roundtrips

Maximum 6 parallel connections
Latency vs Bandwidth

Latency vs Bandwidth

\[ 2 \times \text{Bandwidth} = \text{Same Load Time} \]

\[ \frac{1}{2} \text{ Latency} \approx \frac{1}{2} \text{ Load Time} \]
How can network performance be tackled?
Common Tuning Knobs:

- Avoid redirects, when necessary serve from CDN
- Heavy browser and CDN caching
- Persistent backend connections and IP anycasing
- HTTP/2 with optimized SSL and TCP
- Gzip compression for text-based files
- Minimize DNS lookups
Why HTTP/2 Matters

HTTP

3.22s

HTTPS

4.03s

HTTP with CDN

0.44s

HTTPS with CDN and HTTP/2

0.35s
<table>
<thead>
<tr>
<th>Request</th>
<th>HTTP/1.1</th>
<th>HTTP/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>524 ms</td>
<td>268 ms</td>
</tr>
</tbody>
</table>

HTTP/1.1 vs HTTP/2
HTTP/1.1  vs  HTTP/2

<table>
<thead>
<tr>
<th>Request</th>
<th>Response Size</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET main.js</td>
<td>453 B</td>
<td>453 ms</td>
</tr>
<tr>
<td>GET jQuery.min.js</td>
<td>29 B</td>
<td>53 ms</td>
</tr>
<tr>
<td>GET bootstrap.min.js</td>
<td>15 B</td>
<td>29 ms</td>
</tr>
<tr>
<td>GET 2elalad90h.com</td>
<td>34 B</td>
<td>308 ms</td>
</tr>
<tr>
<td>HTTP/1.1</td>
<td>524 ms</td>
<td>268 ms</td>
</tr>
</tbody>
</table>

What makes HTTP/2 faster?
Optimizations in **HTTP/2**

- **Multiplexing** (1 Connection)
- **Server Push**
- **Resource Prioritization**
- **Header Compression**
Adding a Content Delivery Network (CDN)
Adding a Content Delivery Network (CDN)

- Low latency to client
- Caching on the edge
- DDoS protection
- Failover & Stale-on-error
- Warm backend connections
Hooking Into the Network: **Service Workers**

```javascript
navigator.serviceWorker.register('/sw.js');
//In sw.js:
self.addEventListener('fetch', (event) => {});  //...
```
Hooking Into the Network: **Service Workers**

- Cache Data (CacheStorage)
- Store Data (IndexedDB)
- Receive Push
- Respond when Offline
Hooking Into the Network: **Service Workers**

- **Rewrite** HTTP Requests
- **Sync** Data in Background
- Hide **Flaky Connectivity** from the User
Browser Support for **Service Workers**

<table>
<thead>
<tr>
<th>Browser</th>
<th>11</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>54</th>
<th>55</th>
<th>56</th>
<th>57</th>
<th>58</th>
<th>60</th>
<th>61</th>
<th>10.1</th>
<th>11</th>
<th>11</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>10.2</th>
<th>10.3</th>
<th>4.4</th>
<th>4.4</th>
<th>56</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>IE</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Edge</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>60</td>
<td>61</td>
<td>10.1</td>
<td>11</td>
<td>11</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>10.2</td>
<td>10.3</td>
<td>4.4</td>
<td>4.4</td>
<td>56</td>
<td>61</td>
</tr>
<tr>
<td>Firefox</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safari</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iOS Safari</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opera Mini</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android Browser</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chrome for Android</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Supported by **75%** of browsers.
Browser Support for Service Workers

Service Workers
A method for browsers to run JavaScript in the background to handle network requests and manage cached responses. Service Workers offers a replacement for Application Cache.

Reference: w3c.github.io...
Contact: @bradeeoh - Brady Eidson

- Safari: In Development
- Edge: Implemented, but Toggled
Implementing Service Workers

- Requires SSL
- Hard to debug
- Sw.js must be served top-level (root scope)
Major Challenge: **Cache Coherence**

- Cache just stores *(Req, Res)*-Pairs
- **HTTP browser cache** always exists, too

→ App decides when to **evict** cache
Major Challenge: Cache Coherence

Usual pattern:

Cache Misses → sw.js → Network → Clear SW Cache

→ Does not improve initial page load time
3. Backend Performance
Backend Performance in a Nutshell

- Load Balancing
- Auto-scaling
- Failover

- Stateless Sessions
- Efficient Code & IO

- Horizontally scalable databases (e.g. “NoSQL”)
  - Replication
  - Sharding
  - Failover
Backend Performance in a Nutshell

How can you implement a fast backend?

- Load Balancing
- Auto-scaling
- Failover

- Stateless Sessions
- Minimize shared state
- Efficient Code & IO

- Horizontally scalable databases (e.g. “NoSQL”)
- Replication
- Sharding
- Failover
Option 1: Build a Scalable Architecture

Load Balancer → Application Server → Database

Windows Azure, Amazon web services, VoltDB, MongoDB, Cassandra, redis, Open Telekom Cloud, Google Cloud, Amazon DynamoDB, riak, Amazon web services, S3, HBase
Option 2: Use a Backend Platform

Progressive Web App, AMP, ...

Backend-as-a-Service or Serverless Platform

Firebase, Kinvey, Microsoft Azure, Parse, BaQend
Read More on **Backend Performance**

Articles on **medium.baqend.com**

- **Scalable Stream Processing: A Survey of Storm, Samza, Spark and Flink**
  With this article, we would like to share our insights on real-time data processing we gained building Baqend. This is an updated version...

- **Web Performance in a Nutshell: HTTP/2, CDNs and Browser Caching**
  Successful websites need to be fast, scalable and secure. In this article we survey the state of the art of high-performance websites, in...

- **The AWS and MongoDB Infrastructure of Parse: Lessons Learned**
  This is the extended form of a comment that got some interest on Hackernews. After a grace period of one year, Parse is now offline. This...

- **NoSQL Databases: a Survey and Decision Guidance**
  Together with our colleagues at the University of Hamburg, we — that is Felix Gessert, Wolfram Wingerath, Steffen Friedrich and Norbert...

- **Lessons Learned Building a Backend-as-a-Service: A Technical Deep Dive**
  In this post we share our technical learnings from building a multi-tenant Backend-as-a-Service (BaaS). We cover how a BaaS works, how it...

- **Building a Shop with Sub-Second Page Loads: Lessons Learned**
  Here is the story of how we leveraged research on web-caching and NoSQL systems to prepare a webshop for hundreds of thousands of visitors...
Now, we have a PWA, HTTP/2, etc.

How do we measure web performance?
Page Speed Analyzer

https://www.codetalks.de/de/2017/programm

Domains: 10
Requests: 23
Response Size: 965.17 KB

Your Website

https://www.codetalks.de/de/2017/programm
14737 ms Speed Index 314 ms
8909 ms Time To First Byte 142 ms
18344 ms DOMContentLoaded 406 ms
18615 ms FullyLoaded 1014 ms
18.8 Last Visual Change 0.5

46.93x Faster
Donnerstag, 28.09.2017
08:00 - 09:30
Einlass & Frühstück

Your Website with Speed Kit

https://makefast.speed-kit.com/
18.8

46.93x Faster
Donnerstag, 28.09.2017
08:00 - 09:30
Einlass & Frühstück

46.93x Faster
Begegnung & Vorstellung der Themen Tracks
0.5
Measuring Web Performance

- TTFB
- DOMContentLoaded
- Last Visual Change
- Fully Loaded

<table>
<thead>
<tr>
<th>Last Visual Change</th>
<th>14737ms</th>
<th>Speed Index</th>
<th>314ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>46.93x Faster</td>
<td>62.74x Faster</td>
<td>45.18x Faster</td>
</tr>
<tr>
<td></td>
<td>8909ms</td>
<td>Time To First Byte</td>
<td>142ms</td>
</tr>
<tr>
<td></td>
<td>18314ms</td>
<td>DOMContentLoaded</td>
<td>406ms</td>
</tr>
<tr>
<td></td>
<td>515ms</td>
<td>FullyLoaded</td>
<td>1014ms</td>
</tr>
<tr>
<td></td>
<td>18.8s</td>
<td>Last Visual Change</td>
<td>0.5s</td>
</tr>
</tbody>
</table>

Even as asynchronous JS has completed, no visual changes have been observed.
Measuring Web Performance

How can we measure user-perceived performance?

Even asynchronous JS has completed
The Speed Index

Visual Completeness

$\int_0^\infty$
What we Learned: Wrap-up

- **AMP** and **Instant Articles**: Fast but very limited
- **PWAs** bring native qualities to the web: offline, fast loads, push notifications
What we Learned: **Wrap-up**

- **HTTP/2** is much faster due to multiplexing and push
- **CDNs** tackle latency & caching
- **Service Workers** can modify the browser’s requests

Network
What we Learned: **Wrap-up**

- **Cloud Providers** make scaling out easier
- **Servers** and **Database Systems** need to support scalability and failover
What we Learned: Wrap-up

How can we improve the performance of existing sites?

- Cloud Providers make scaling out easier
- Servers and Database Systems need to support scalability and failover

Backend
Speed Kit
Turning Websites into Instantly-Loading Progressive Web Apps
What **Speed Kit** does.
What **Speed Kit** does.
What Speed Kit does.

Backed by 30 Man-Years of Research
Adding Speed Kit to a Site
1. Configure Domain

Set which URLs Baqend should accelerate.
2. Include Code Snippet

Add the Speed Kit Service Worker to the website.
3. Requests Accelerated

Speed Kit routes the requests through Baqend’s CDN.
How it **works** under the hood

Website with Snippet → Speed Kit Service Worker → Baqend Service

- Requests
- Fast Requests
- Pull
- Push
- 3rd Party Services
- Existing Backend
- Other
Speed Kit works across tech stacks.
Works for Publishers.

kicker.de
Works for **Landing Pages**.

molsoncoors.com
Works for Portals.

realtor.com
Works for E-Commerce.

alibaba.com
Works for **Conference Websites**.

**codetalks.de**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Original Website</th>
<th>Improved Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Index</td>
<td>14737ms</td>
<td>314ms</td>
</tr>
<tr>
<td>Time To First Byte</td>
<td>8909ms</td>
<td>142ms</td>
</tr>
<tr>
<td>DOMContentLoaded</td>
<td>18344ms</td>
<td>406ms</td>
</tr>
<tr>
<td>FullyLoaded</td>
<td>18615ms</td>
<td>1014ms</td>
</tr>
<tr>
<td>Last Visual Change</td>
<td>18.8s</td>
<td>0.5s</td>
</tr>
<tr>
<td>46.93x Faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>62.74x Faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45.18x Faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18.36x Faster</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38.38x Faster</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Works for **Aggregators**.

news.google.com
Does it work for Your Site?

www.example.com

Go

test.speed-kit.com
What we develop at **Baqend**

**Speed Kit**
- Turns Existing Sites into **PWAs**
- **50-300% Faster** Loads
- **Offline** Mode

**Platform**
- Platform for building (Progressive) **Web Apps**
- **15x** Performance Edge
- Faster **Development**
Web Performance Literature

Good Resources

- High Performance Mobile Web
- Building Progressive Web Apps
- Browser Networking

- https://developers.google.com/web/fundamentals/performance/?hl=en
- https://hpbn.co/
- https://medium.baqend.com/

Performance Tools

- PageSpeed Insights: https://developers.google.com/speed/pagespeed/
- Page Speed Analyzer: https://test.speed-kit.com
- BaQend: https://www.baqend.com/
- WebPageTest: http://www.webpagetest.org/
We are hiring.

Frontend Developers
Mobile Developers
Java Developers
Web Performance Engineers

Contact us.

Felix Gessert · fg@baqend.com · www.baqend.com
Questions?

Our other talks:

**Th. 16:00** Real-Time Databases Explained: Why Meteor, RethinkDB, Parse and Firebase Don't Scale

**Fr. 10:00** Real-Time Anwendungen mit React und React Native entwickeln

**Fr. 17:00** Wie man ein Backend-as-a-Service entwickelt: Lessons Learned