HTTP/2

De wachttijd is over!

De inkt van de *final spec* van het HTTP/2 protocol is net een jaar droog. De hoogste tijd dus om de opvolger van HTTP/1.1 eens aan de tand te voelen en te kijken wat dit betekent voor ontwikkelaars. HTTP/2 brengt vernieuwing in het fundament van het web. Niet zo fundamenteel als IPv6 en zeker niet met zo'n lang migratiepad. Je wist het vast niet, maar je hebt al maanden, zo niet jaren, met HTTP/2 of voorloper SPDY gebrowsed.

Al enige jaren geleden stelden grootverbruikers van het web (met Google als aanvoerder) vast, dat HTTP/1.x tekenen van slijtage begon te vertonen op het gebied van performance en efficiëntie. Zo gebruikt HTTP/1.x onnodig veel bandbreedte. Het is namelijk een *plaintext* protocol. Dat is handig voor debuggen, maar niet erg efficiënt. Daarnaast is HTTP/1.x is een *chatty* protocol. Vooral de headers bevatten erg veel redundantie. Ze kunnen 200 Bytes tot zo'n 2kB groot zijn; informatie die vaak bij *elke* request opnieuw ongewijzigd wordt verstuurd.

TCP kent een mechanisme dat *slow start* heet: als de connectie net gestart is, worden er kleine pakketten gestuurd. Deze pakketten worden steeds groter, totdat *ofwel* een ingesteld maximum bereikt is, *ofwel* het pakket verloren gaat. In dat laatste geval begint de slow start weer van voren af aan (zie **figuur 1**). Gevolg is dat vaak alleen voor de headers van een HTTP request al 7 of 8 roundtrips op IP-pakketniveau nodig zijn..

HTTP/1.1 vermindert het slow start effect door middel van KeepAlive en Pipelining. Beide lijden echter aan Head of Line Blocking; een fenomeen waarbij het eerste request de daaropvolgende requests vertraagt. Vanwege Head of Line Blocking openen browsers meerdere parallelle verbindingen. In RFC 7230 wordt aangeraden dit aantal te beperken (voorheen werd zelfs 2 als maximum genoemd). Elke nieuwe verbinding lijdt natuurlijk wel onder TCP slow start. Daarnaast kost elke verbinding server resources.

'In den beginne' bestond een pagina uit enkele bestanden: HTML, JS, CSS en wat plaatjes. Inmiddels gaat het vaak om een honderdtal resources en vele MB's, alleen al voor de eerste pagina. Dan begint HTTP/1.x echt te 'kraken' en de page load tijd flink te stijgen.

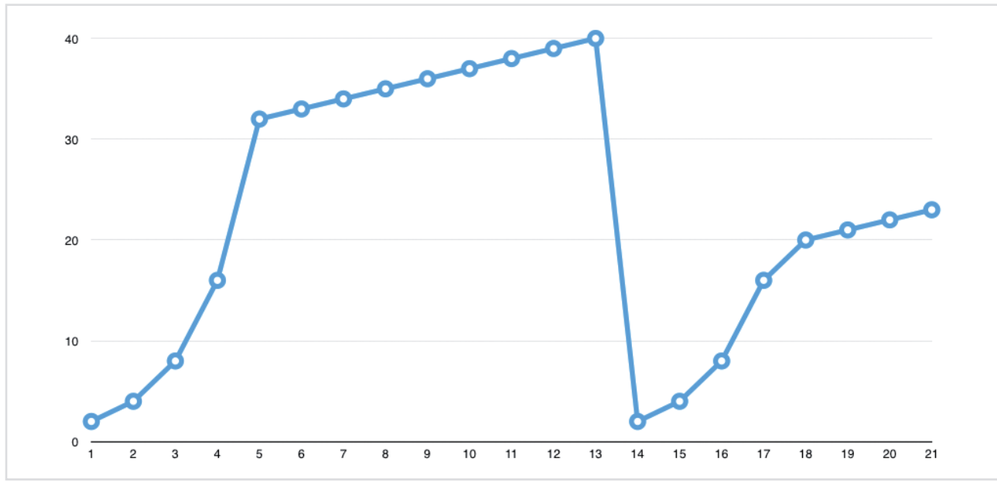
HTTP/2 biedt je de mogelijkheid om eindgebruikers de rijke gebruikerservaring, waaraan ze gewend zijn, te blijven bieden en deze nog verder uit te bouwen. Het HTTP/2 protocol (RFC 7540) is gepubliceerd op 14 mei 2015. Het is grotendeels gebaseerd op Google's SPDY/4. Veel browsers en diverse servers ondersteunden dat protocol al. Dat is de reden dat HTTP/2 relatief snel uitgerold kan worden.

Doel van HTTP/2 is het verkleinen van latency, het verbeteren van de page load tijd en het verbeteren van network & server resourcegebruik. Hiervoor is het gebruik van de onderliggende TCP-verbinding (het zogenaamde "over the wire" protocol) compleet herzien.

HTTP/2 in een notendop:

- Een **binair** protocol.
- **Multiplexed**: geen Head of Line Blocking. Een request/response-paar heet een stream en heeft een eigen stream id. Op deze manier kunnen meerdere streams afwisselend pakketjes ('frames') sturen. Grote stukken data worden in meerdere frames opgesplitst. Elke stream heeft daarvoor individuele flow control instelling, waarmee back pressure toe te passen is. Een frame benoemt naast zijn stream id ook zijn frame type, bijvoorbeeld, HEADERS of DATA.
- **Één persistente verbinding per host** is voldoende: TCP slow start is nauwelijks meer een probleem.
- **Comprimeert headers** om bandbreedte te sparen.

Hedzer Westra is Java-ontwikkelaar bij Ordina J-Technologies. Zijn carrière met Java begon in 1996, tijdens een studie Informatica. Je kunt hem bereiken op **hedzer.westra@ordina.n**l. Hij presenteert bij Ordina onder andere de cursus Web sockets & server push.



*Figuur 1: TCP slow start. Op de x-as het IP-pakketnummer, op de y-as de pakketgrootte in kB. Bron: presentatie Mark Nottingham: "What to Expect from HTTP/2", sheet 16 (**www.mnot.net/talks/http2-expectations**)*

- Het stelt servers in staat **"push" berichten** te sturen en de browser zet deze meteen in z'n cache. Server Push is voorlopig de enige functie, die toegankelijk is in applicaties.
- **Prioriteit** en **onderlinge afhankelijkheid** van requests zijn een optionele hint naar de server. De browser kan hiermee bijvoorbeeld voorrang vragen voor HTML of plaatjes pas als laatste laten opleveren. De waarden zijn dynamisch aanpasbaar, bijvoorbeeld als de gebruiker scrolt tijdens het renderen of naar een andere tab switcht.

HTTP methods, status codes, header velden, URL'S, standaardpoorten en protocol prefixes (http:// & https://) blijven ongewijzigd.

Op HTTP/2 sites worden HTTP/1.x workarounds overbodig en sommige gevallen zelfs nadelig. Je kunt ze (na volledige migratie) verwijderen uit je applicatie.

Het gaat dan om:

- **Domain sharding**: statische content wordt over meerdere hostnames verspreid. Vaak wordt dit toegepast in combinatie met een CDN (Content Delivery Network). Hiermee wordt om de per-host browserlimiet van 2-8 connecties heen gewerkt. Bij HTTP/2 geeft één persistente connectie vanwege multiplexing al voldoende capaciteit.
- **Sprites**: het combineren van vele kleine icoontjes in één plaatje, waar vervolgens met CSS het juiste icoon weer wordt 'uitgesneden'. Ook hier geeft de enkele multiplexed verbinding, in combinatie met (header) compressie, voldoende capaciteit om zelfs vele kleine plaatjes allemaal apart op te halen.
- **Resource inlining** & **concatenation**, ook wel bekend als **minification**. Dit wordt vaak toegepast bij Javascript en CSS, waarbij meerdere kleine files tot één grote worden samengevoegd en gecomprimeerd. Alle noodzaak hiertoe vervalt, dankzij de veel grotere throughput van vele kleine bestanden.

Early adopters hebben een SEO (Search Engine Optimization) voordeel, want Google zal je site hoger ranken als de page load tijd kleiner is.

Iedereen veert natuurlijk op bij het magische woord 'push'. HTTP/2 biedt het en servers ontsluiten dit tevens naar applicaties. Verwacht er echter geen wonderen van, want HTTP/2 definieert bijvoorbeeld *geen* Javascript API. Wil je pushberichten afhandelen, dan blijf je de keuze houden uit Server Sent Events (SSE), Ajax of Websockets.

Je moet de toepassing van Server Push eerder zoeken in het voorspellend opsturen van statics. Bij initiële page load kan de server alle benodigde static content opsturen. Dit kan het aantal round trips en daarmee de browser rendertijd flink verkleinen.

DE LATENCY KILLER VAN 2016

In Servlet 4.0 (JSR 369), die hopelijk in 2017 beschikbaar is in JEE8, komt een PushBuilder om Server Push mogelijk te maken. De appserver zal dan op eigen initiatief een 'virtueel' request naar je applicatie sturen. Jetty biedt nu al een eigen push API-implementatie, die zeer waarschijnlijk in JSR 369 gestandaardiseerd wordt.

Zoals **figuur 2** aantoont, is de HTTP/2-compatible browser dekking (globaal 70%, NL 85%) voor de meeste applicaties voldoende om te veronderstellen, dat de gebruiker op dit moment HTTP/2 tot zijn of haar beschikking heeft.

In principe is HTTP/2 volledig backwards compatible. Als de browser of server (of de tussenliggende proxy of load balancer) geen HTTP/2 spreekt, wordt teruggevallen op HTTP/1.1.

Server- en middlewareleveranciers zijn druk bezig om hun software en appliances geschikt te maken voor HTTP/2. Httpd, nginx, WildFly, Tomcat, Jetty, Netty en NodeJs zijn gereed. De F5 BIG-IP load balancer, IIS en HAProxy eveneens. CDN leveranciers CloudFlare en Akamai bieden ook HTTP/2. Microservice stacks, zoals Spring boot en Dropwizard bieden eveneens ondersteuning, maar de reactive stacks Play, Vert.x en Ratpack nog niet.

Fabian Stäber beschrijft op zijn blog unrestful.io de open source commandline tool h2c die hij ontwikkeld heeft. Geschreven in Go en voor vele platforms als executable te downloaden van github.

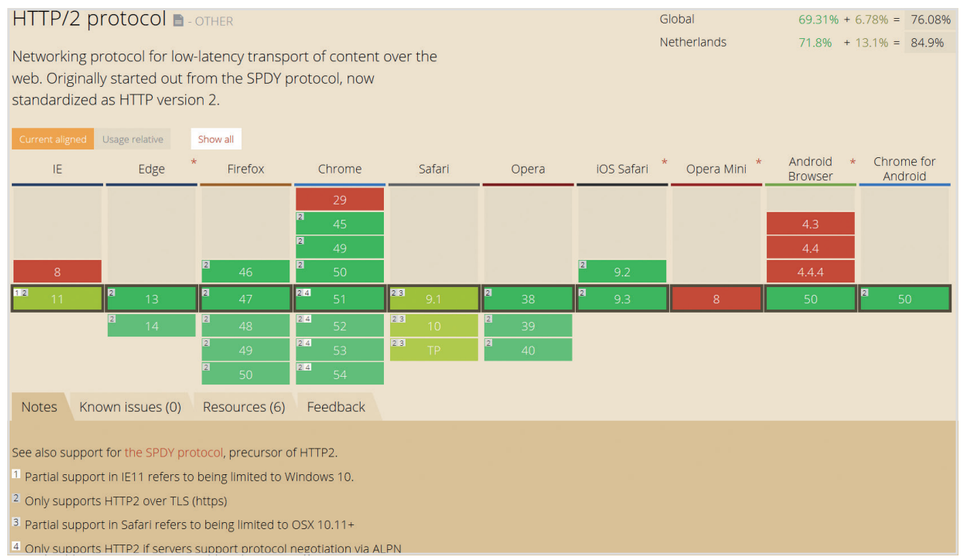
Een voorbeeld:
- start in console 1 `h2c start --dump`
- start in console 2 `h2c get https://webtide.com`.
Alle content verschijnt dan in console 2, inclusief eventuele pushed data.

In console 1 scrolt heel veel informatie langs, onder andere je eerste echte Push Promise:

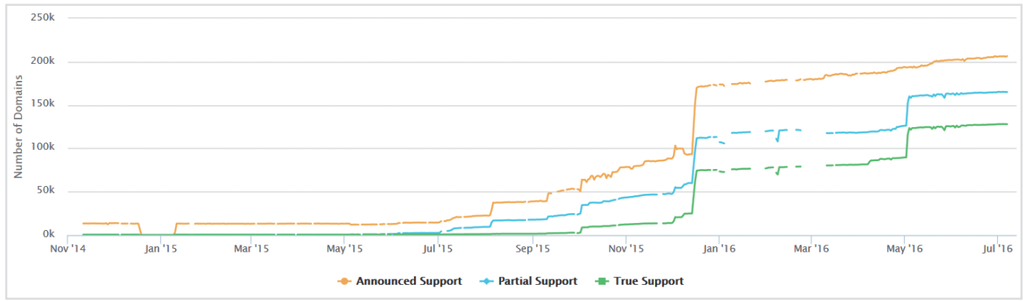
```
-> HEADERS(1)
+ END_STREAM
+ END_HEADERS
:method: GET
:scheme: https
:authority: webtide.com
:path: /

<- PUSH_PROMISE(1)
+ END_HEADERS
Promised Stream Id: 2
:scheme: https
:method: GET
:authority: webtide.com
:path: /wp-content/plugins/crayon-syntax-highlighter/
fonts/monaco.css?ver=_2.7.2_beta
host: webtide.com
```

De push promise hoort bij stream 1 en geeft aan dat er een stream 2 response komt op de - nog niet door de browser opgevraagde! - GET URL zoals genoemd bij `:path:`.



*Figuur 2: HTTP/2 ondersteuning, zoals gerapporteerd door **caniuse.com***



Figuur 3: Ontwikkeling van het aantal sites, dat HTTP/2 ondersteunt (bron: isthewebhttp2yet.com).

Volgens **isthewebhttp2yet.com** zijn er al meer dan 125.000 sites, die HTTP/2 volledig ondersteunen.

Vanwege het binaire karakter van HTTP/2 moeten debugging tools geüpgraded worden. Bestaande tools werken niet meer, of vallen in het beste geval terug op HTTP/1.1. Zo vallen Fiddler en Charles terug op HTTP/1.1; .Net SDK support ontbreekt. De netwerkanalysetool Wireshark ondersteunt HTTP/2 volledig.

In Firefox kun je een HTTP/2 verbinding herkennen aan 'HTTP/2.0 200 OK' en 'X-Firefox-Spdy: h2' in de response header. Chrome toont meer informatie op chrome://net-internals/#http2. Curl ondersteunt het protocol ook sinds versie 7.43.0.

Verder is er nog een handige open source tool, genaamd h2c. Fabian Stäber beschrijft op **unrestful.io** deze tool (zie kader).

In de HTTP/2 Working Group was geen consensus om TLS verplicht te stellen in HTTP/2. Echter, alleen Android Browser ondersteunt *plaintext* HTTP/2 (ook wel 'h2c'). In de praktijk betekent dit, dat HTTP/2 browserverkeer altijd encrypted ('h2') is.

Wel is afgesproken om de NPN handshake te vervangen voor ALPN en om de minimaal vereiste algoritmen en sleutelgroottes 'op te krikken'.

JSE 9 wordt HTTP/2 compatible door toevoeging van ALPN en een HTTP/2 client. Jetty, Netty en OkHttp leveren nu al een asynchrone client. Vanwege HTTP/2's strengere encryptie-eisen heb je Java8 of Java7 met Bouncy Castle nodig. Daarnaast is ook de Jetty ALPN bootclasspath extensie benodigd. Deze kun je activeren met het commandline argument '-Xbootclasspath/p:alpn-boot-VERSION.jar'.

Een probleem is dat de versie van deze extensie heel direct gekoppeld is aan de exacte versie van de Java 7 of 8 runtime. Zo moet je bij Java 1.8.0u92 bijvoorbeeld op zoek naar 'org.mortbay.jettyalpn:alpnboot:8.1.8.v20160420'.

Ik kan het volgende concluderen: HTTP/2 pakt succesvol het latency probleem aan waar HTTP/1.1 last van heeft. De uitrol is in volle gang. Als Java-ontwikkelaars zullen we de komende tijd gaan uitvinden hoe we er in onze applicaties het meest effectief gebruik van kunnen maken. Kunnen we er bijvoorbeeld krachtiger REST interfaces mee bouwen? Er ligt een spannende tijd voor ons! ■

REFERENTIES

HTTP/2 spec home **https://http2.github.io**
JEP 110: HTTP/2 Client **http://openjdk.java.net/jeps/110**
Servlet 4.0 JSR-369 **https://jcp.org/en/jsr/detail?id=369**
Fabian Stäbers blog **http://unrestful.io**
Caniuse HTTP/2 **http://caniuse.com/#feat=http2**
Akamai latency demo **https://http2.akamai.com/demo**
Jetty PushBuilder javadoc **http://download.eclipse.org/jetty/stable-9/apidocs/org/eclipse/jetty/server/PushBuilder.html**
Jetty ALPN bootclasspath extensie **http://www.eclipse.org/jetty/documentation/current/alpn-chapter.html**