http2 explained

Daniel Stenberg

فهرست مطالب

مقدمه و معرفی	1.1
پیش‌زمینه	1.2
امروز HTTP	1.3
کارهایی که برای غلبه بر تأخیرها انجام شده	1.4
آپدیت‌کردن HTTP	1.5
مفاهیم http2	1.6
پرتکل http2	1.7
افزونه‌ها	1.8
دنیایی با http2	1.9
در فایرفاکس http2	1.10
در کرومیوم http2	1.11
curl در http2	1.12
بعد از http2	1.13
خواندن بیشتر	1.14
تقدیر و تشکر	1.15
واژه‌نامه	1.16

شرحی بر http2

این نوشته به زبان ساده‌تری استاندارد RFC 7540 (HTTP/2), پیش‌زمینه‌ی آن, مفاهیم, پرتکل، پیاده‌سازی‌های انجام‌شده و آینده‌ی آن را توصیف می‌کند.

صفحه‌ی https://daniel.haxx.se/http2/ را برای توضیحات جامع‌تر درمورد این پروژه ببینید.

همچنین، برای سورس این کتاب، صفحه‌ی https://github.com/bagder/http2-explained را ببینید.

همکاری‌کردن

من از همه‌ی کمک‌ها و مشارکت‌کنندگان برای بهبود این پروژه استقبال می‌کنم! ما Pull Request هم قبول می‌کنیم، اما شما می‌توانید برایمان در قسمت issues هم بنویسید یا اصلا به ایمیلم به آدرس daniel-http2@haxx.se پیشنهادتان را بفرستید!

/ دنیل استِنبرگ

یادداشت مترجم

در این ترجمه سعی کرده‌ام بدون استفاده از واژه‌های نامأنوس و ناآشنا، ترجمه‌ای روان و همه‌گیر ارائه دهم؛ همچنین در واژه‌نامه نیز واژه‌ها و اصطلاحات جدیدتر را آورده‌ام.

این ترجمه، ترجمه‌ای تحت‌اللفظی نیست که تک‌تک جملات در نوشته‌ی اصلی با جملات ترجمه‌شده یکسان باشند، اما تلاش کردم که مقصود اصلی نویسنده منتقل شود و محتوای نوشته‌ی اصلی حفظ شود.

قطعا این ترجمه بی‌نقص نیست و جای بهترشدن دارد. خوش‌حال می‌شوم که پیشنهادات خودتان را در توئیتر به اکانت @ehsaan_me یا ایمیل the.black.suited در جیمیل بفرستید.

امیدوارم با خواندن این کتاب، چیزهای جدیدی یاد بگیرید :)

۱. پیش‌زمینه

این نوشته http2 را از یک نگرش تکنیکی و پرتکلی توصیف می‌کند. نوشتن آن، زمانی که آغاز شد که دنیل آن را به صورت یک ارائه در استکهلم در آپریل ۲۰۱۴ عرضه کرد که پس از آن، گسترش یافت و با توضیحات بیشتری به یک نوشته با جزئیات بیشتر تبدیل شد.

RFC 7540 نام رسمی مشخصات نهایی http2 است که در پانزدهم می ۲۰۱۵ منتشر شد: https://www.rfc-editor.org/rfc/rfc7540.txt

تمامی خطاها در این نوشته، حاصل قصور من *[یا مترجم]* است. لطفا اگر خطایی دیدید، به من اطلاع دهید تا در نسخه‌های بعدی آن‌ها را رفع کنم.

در این نوشته، به طور ثابت از واژه‌ی http2 برای خطاب این پرتکل جدید استفاده کرده‌ام، در حالی که از نظر فنی، نام صحیح آن HTTP/2 می‌باشد. من این تصمیم را برای بهبود خوانایی نوشته و تطابق آن با زبان گرفته‌ام.

۱.۱. نویسنده

نام من، دنیل استنبرگ است و برای موزیلا کار می‌کنم. من در حدود ۲۰ سال است که با پروژه‌های اپن‌سورس و شبکه در پروژه‌های مختلف کار کرده‌ام. احتمالا به این علت بیشتر شناخته می‌شوم که توسعه‌دهنده‌ی راهبر در پروژه‌های curl و libcurl بوده‌ام. همچنین من در کارگروه IETF HTTPbis نیز برای چند سال حضور داشتم و سعی در به‌روزنگه‌داشتن پرتکل HTTP 1.1 می‌کردم، همچنین در کار استانداردسازی http2 نیز مشارکت داشته‌ام.

ایمیل من: daniel@haxx.se

توئیتر: @bagder

وب‌سایت: daniel.haxx.se

وبلاگ: daniel.haxx.se/blog

۱.۲. کمک!

اگر هر اشتباهی، نارسایی، خطا و حقایق فنی نادرست مشاهده کردید، لطفا برای من یک نسخه‌ی تصحیح‌شده از پاراگراف مربوطه را ارسال کنید و نسخه‌ی اصلی را تصحیح خواهم کرد. همچنین از کسانی که کمک می‌کنند، به درستی نام خواهم برد! امیدوارم که بتوانیم این نوشته را به مرور زمان بهتر کنیم.

این نوشته در https://daniel.haxx.se/http2 دردسترس است.

۱.۳. لایسنس

۱.۴. تغییرات این نوشته

تغییرات نوشته‌ی اصلی را می‌توانید در این لینک مشاهده کنید. https://http2-explained.haxx.se/content/en/part1.html#14-document-history

تغییرات ترجمه فارسی در همین صفحه نوشته خواهد شد.

۲. HTTP امروز

HTTP 1.1 تبدیل به پرتکلی شده
که این روزها، تقریبا برای همه‌چیز در اینترنت استفاده می‌شود. سرمایه‌گذاری‌های عظیمی در پرتکل‌ها و زیرساخت‌هایی که
از HTTP 1.1
بهره می‌برند شده است، به دلیل این‌که اغلب اوقات، اجراکردن چیزی روی HTTP
راحت‌تر از ساختن چیزی از نو است.

۲.۱. HTTP 1.1 بسیار بزرگ است

هنگامی که HTTP ساخته شده و به دنیا عرضه شد، بسیاری آن را یک پرتکل ساده و سرراست یافتند، ولی زمان ثابت کرد که این دیدگاه نادرست است. HTTP 1.0 در استاندارد RFC 1945 تنها در ۶۰ صفحه توصیف شده است که در سال ۱۹۹۶ منتشر شد. RFC 2616 که HTTP 1.1 را توضیح می‌دهد، در یک رشد قابل‌توجه، به ۱۷۶ صفحه هم می‌رسد. هنوز هم هنگامی که در IETF روی استانداردهای مربوط به آن کار می‌کنیم، [ناچارا] آن را به ۶ سند، که تعداد صفحات آن‌ها روی هم بسیار بیشتری می‌شود، تقسیم کردیم که حاصل آن، استاندارد RFC 7230 و خانواده شد. به هر حال، HTTP 1.1 بزرگ است و دارای جزئیات و ظریف‌کاری‌های بسیار، و البته امکانات اختیاری است.

۲.۲. دنیایی از گزینه‌ها

طبیعت HTTP 1.1، داشتن جزئیات بسیار و گزینه‌های موجود برای افزونه‌های بعدی، تبدیل به یک اکوسیستم نرم‌افزاری شد که تقریبا هیچ پیاده‌سازی، همه‌چیز را پیاده‌سازی نمی‌کند، و حتی ممکن نیست که دقیقا بگوییم که این «همه‌چیز» چه چیزهایی هستند. این ویژگی باعث به‌وجودآمدن شرایطی شد که قابلیت‌هایی که در ابتدا، بسیار کم‌استفاده بودند، به ندرت پیاده‌سازی شدند و کسانی که این قابلیت‌ها را پیاده‌سازی کردند، متوجه شدند که کاربردهای بسیار کمی برای آن‌ها وجود دارد.

بعدها، این ویژگی‌ها باعث ایجاد ناهماهنگی بین کلاینت‌ها و سرورهایی که از این قابلیت‌ها استفاده می‌کردند شد. HTTP pipelining از نمونه‌های بارز این قابلیت‌ها است.

۲.۳. استفاده‌نکردن از ظرفیت TCP

HTTP 1.1 به سختی از همه‌ی مزایا، قدرت و کارایی TCP استفاده می‌کند. کلاینت‌های
HTTP و مرورگرها باید در پی یافتن راه‌های خلاقانه برای کاهش زمان بارگذاری
صفحات باشند.

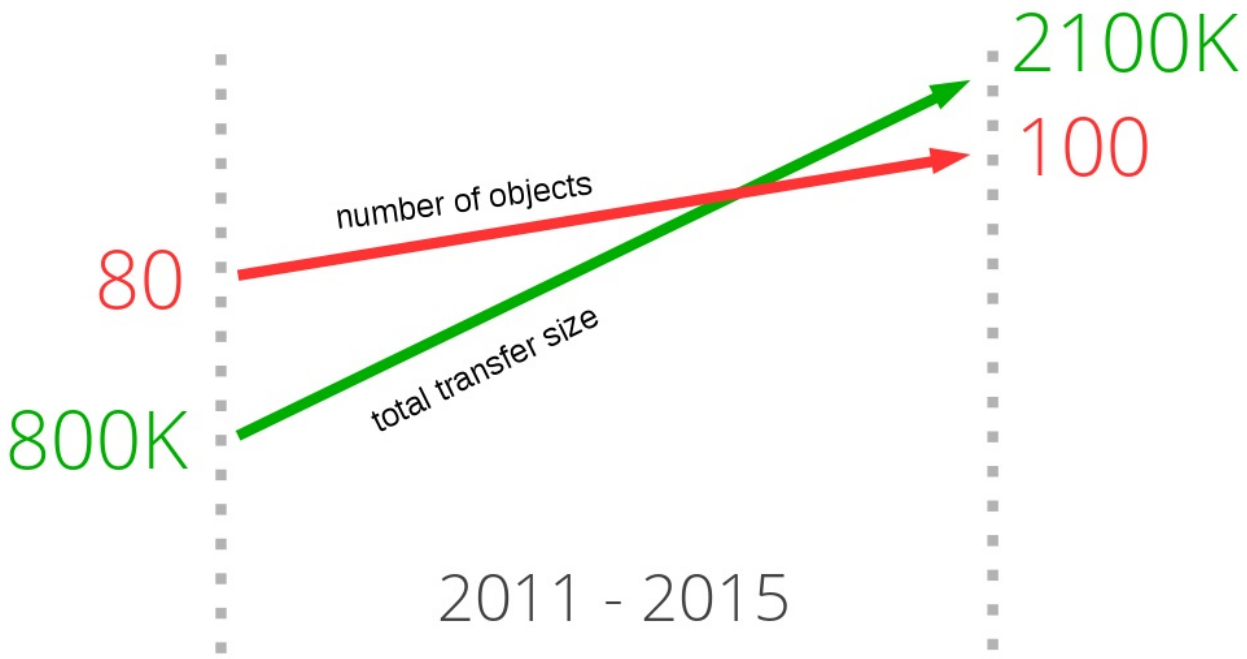
تلاش‌های دیگری که در طول این سال‌ها به طور موازی پیگیری می‌شدند هم نشان داده‌اند که جایگزین‌کردن TCP کار راحتی نیست، به همین دلیل ما روی بهبود TCP و پرتکل‌های وابسته به آن کار می‌کنیم.

به عبارت دیگر، TCP می‌تواند به گونه‌ای استفاده شود که وقفه‌ها را کمتر کند یا از بازه‌های زمانی تلف‌شده برای ارسال و دریافت داده‌های بیشتری استفاده شود. قسمت‌های بعدی بعضی از این کاستی‌ها را نمایان می‌کنند.

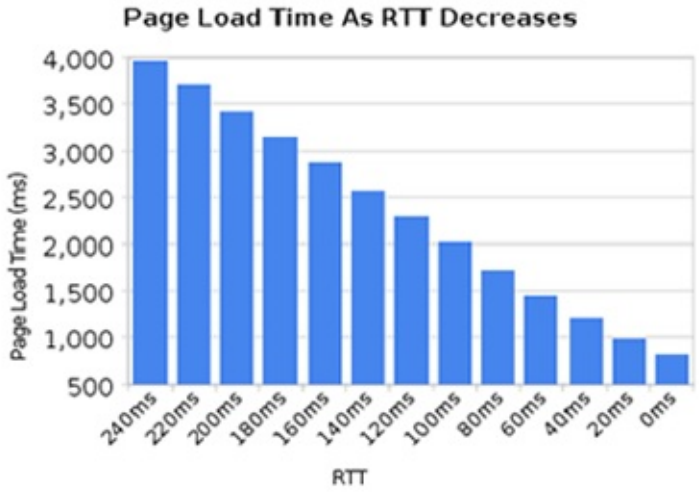
۲.۴. حجم مبادل‌ها و تعداد فایل‌ها

وقتی به آمارهای امروزی درمورد پرطرفدارترین وب‌سایت‌ها و صفحه‌های اول آن‌ها نگاه می‌کنیم، یک الگوی مشخص را می‌توان برداشت کرد. در طول این سال‌ها، حجم داده‌هایی که باید برای بازکردن این صفحات مبادله کرد، به ۱.۹ مگابایت رسیده است. چیزی که دراین‌باره مهم‌تر است، این است که بیش از ۱۰۰ فایل مختلف برای نمایش هر صفحه لازم است.

همان‌طور که نمودار زیر نشان می‌دهد، این روند به همین منوال در حال پیش‌رفت است و کمتر نشانه‌ای از تغییر در آینده‌ای نزدیک به چشم می‌خورد. این نمودار، رشد اندازه‌ی داده‌های مبادله‌شده (با رنگ سبز) و میانگین تعداد درخواست‌ها (قرمز) را در وب‌سایت‌های پرطرفدار اینترنت نشان می‌دهد و این‌که این ارقام چگونه در یک بازه‌ی ۴ ساله تغییر کرده‌اند.



۲.۵. تأخیرها



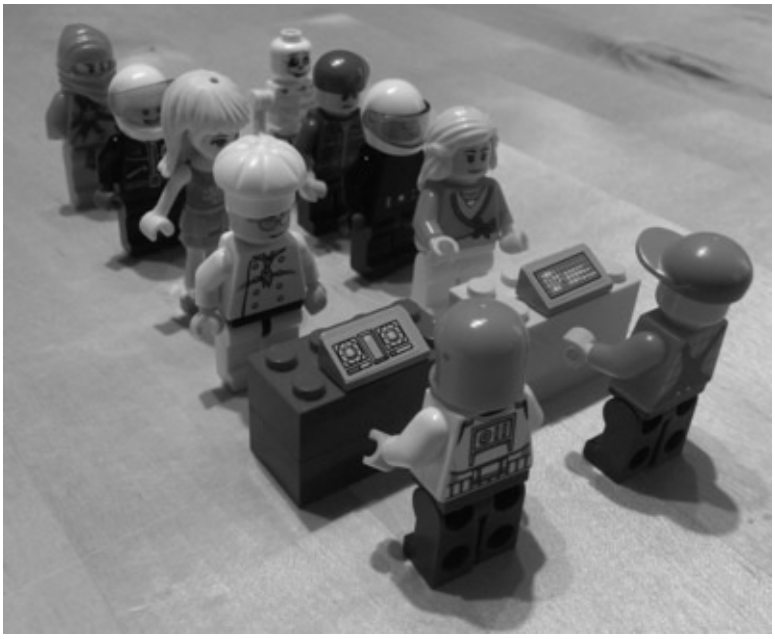
HTTP 1.1 بسیار به تأخیر حساس است، می‌توان گفت بخشی از این حساسیت به دلیل این است که HTTP pipelining هنوز آن‌قدر مشکل دارد که اکثر کاربران نتوانند از آن استفاده کنند.

با این که شاهد رشد عظیمی در پهنای‌باند دردسترس مردم در چند سال اخیر بوده‌ایم، ولی به همان نسبت،، اقدامی برای کاهش این تأخیرها نشده است. ارتباطات با تأخیر بالا، مانند فناوری‌های کنونی موبایل، داشتن یک تجربه‌ی وب‌گردی خوب و سریع را سخت می‌کند، حتی اگر پهنای باند بسیار بالایی هم در اختیار داشته باشید.

مورد استفاده‌ی دیگری از تأخیرهای پایین، نوع‌های خاصی از ویدیو هستند، مانند ویدیوکنفرانس، گیمینگ و مشابه‌های آن که هیچ جریان از پیش‌ساخته‌ای برای ارسال وجود ندارند.

۲.۶. انتخاب صف درست (Head-of-line blocking)

HTTP pipelining یک راه برای فرستادن یک درخواست دیگر، در حالی که منتظر پاسخ درخواست دیگری هستیم، است. این مفهوم بسیار شبیه به صف‌های بانک یا سوپرمارکت است. شما نمی‌دانید که نفر اول صف یک آدم تر و فرز است یا یک آدم مزاحم که وقت زیادی را تلف می‌کند تا کارش را انجام دهد.



البته، شما می‌توانید صفی را انتخاب کنید که فکر می‌کنید سریع‌تر است یا حتی صف خودتان را تشکیل بدهید. ولی در نهایت، شما نمی‌توانید تصمیم نگیرید. و وقتی تصمیم گرفتید، نمی‌توانید تصمیم خود را عوض کنید.

تشکیل یک صف جدید با هدررفت منابع و کارایی همراه است، پس هنگامی که تعداد صف‌ها بیشتر می‌شود دیگر کارایی ندارد. در واقع، هیچ راه‌حل بی‌نقصی برای این مشکل وجود ندارد.

حتی امروزه هم بیشتر مرورگرهای دسکتاپ با HTTP pipelining به صورت غیرفعال عرضه می‌شوند.

جزئیات بیشتر درباره‌ی این موضوع را می‌توانید در گزارش 264354 باگ‌زیلای فایرفاکس بخوانید.

۳. کارهایی که برای غلبه بر تأخیرها انجام شده

وقتی ما با مشکلات مواجه می‌شویم، گرد هم می‌آییم تا برای آن‌ها راه‌حل‌هایی پیدا کنیم. بعضی از این راه‌حل‌ها کاربردی و هوشمندانه هستند، و بعضی دیگر فقط باعث درست‌شدن موانع بیشتر می‌شوند.

Spriting .۳.۱

Spriting روشی است که چندین عکس کوچک‌تر را در قالب یک عکس
بزرگ جا می‌دهند. سپس، با جاوا اسکریپت یا CSS،
می‌توانید عکس‌های کوچک‌تر را از این عکس بزرگ ببرید و آن‌ها را نمایش دهید.

وب‌سایت‌ها از این روش برای افزایش سرعت استفاده می‌کنند. دانلود یک عکس بزرگ از طریق HTTP 1.1، بسیار سریع‌تر از دریافت ۱۰۰ عکس کوچک‌تر است.

البته، این روش معایبی هم برای صفحاتی که صرفا می‌خواهند ۲ یا ۳تا از این عکس‌ها را نشان دهند هم دارد. همچنین Spriting باعث می‌شود که هنگام پاک‌کردن حافظه‌ی Cache، همه‌ی عکس‌ها با هم پاک شوند، به جای این‌که عکس‌های پراستفاده‌تر بمانند.

Inlining .۳.۲

Inlining هم یک ترفند دیگر برای جلوگیری از فرستادن عکس‌های
تکی است که با جاسازی داده‌های عکس در قالب URL
کار می‌کند. مزایا و معایب این روش، مشابه Spriting است.

```
.icon1 {
    background: url(<data>) no-repeat;
}

.icon2 {
    background: url(<data>) no-repeat;
}
```

۳.۳. ادغام‌کردن

یک وب‌سایت بزرگ، چندین فایل جاوااسکریپت متفاوت دارد. توسعه‌دهندگان از ابزارهای Front-End تا این فایل‌ها را ادغام یا ترکیب کنند تا مرورگر به جای دریافت چندین فایل کوچک، یک فایل بزرگ را دریافت کند. ولی، در این روش تنها وقتی داده‌های بسیار کمتری نیاز است، داده‌های بسیاری فرستاده می‌شود و همچنین داده‌های بسیاری باید بارگذاری شوند تا تغییرات اعمال شوند.

البته این روش، صرفا برای توسعه‌دهندگان درگیر در پروژه، مشکل ایجاد می‌کند.

۳.۴. توزیع‌کردن

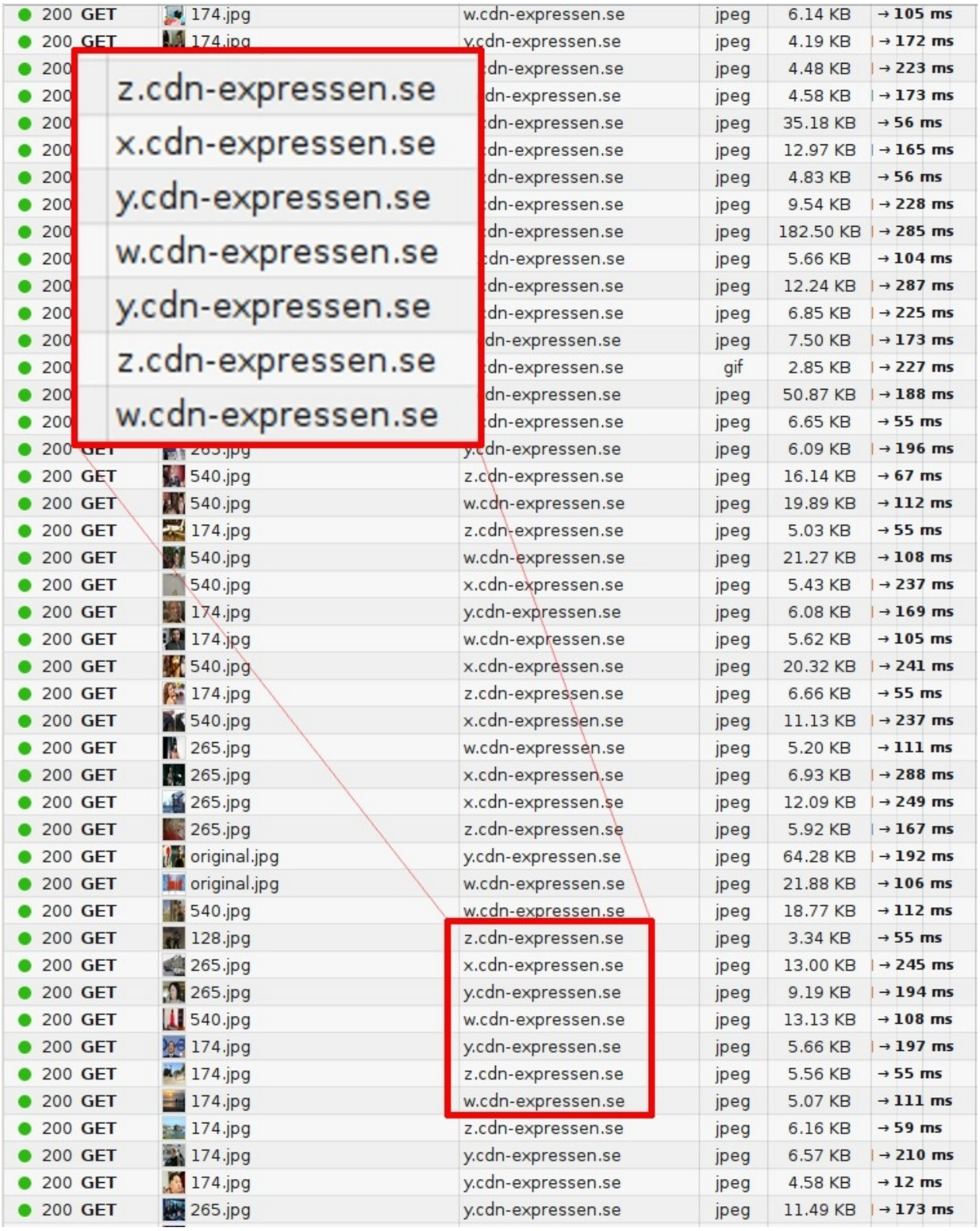
آخرین ترفندی که برای افزایش کارایی ذکر می‌کنم، معمولا با نام توزیع‌کردن (Sharding) شناخته می‌شود. اساسا، به این معنی است که جنبه‌ها و بخش‌های مختلف سرویس را روی چندین میزبان (Host) مختلف بارگذاری کنند. در نگاه اول، ممکن است که این کار عجیب به نظر برسد، ولی دلیل پشت آن، قانع‌کننده است.

در ابتدا، استاندارد HTTP 1.1 مشخص کرده بود که کلاینت‌ها فقط می‌توانند از ۲ کانکشن TCP برای هر Host استفاده کنند. پس برای زیرپاگذاشتن این قانون، سایت‌های باهوش‌تر از host nameهای جدید استفاده کردند و بنابراین، تعداد کانکشن‌ها و در نتیجه سرعت بارگذاری صفحات بیشتر می‌شد.

به مرور زمان، این محدودیت نیز حذف شد. و امروز کلاینت‌ها می‌توانند به راحتی ۶ تا ۸ کانکشن به هر host name ایجاد کنند. ولی آن‌ها همچنان محدودیت دارند، پس سایت‌ها از این تکنیک برای افزایش تعداد کانکشن‌ها استفاده می‌کنند. از آن‌جایی که تعداد فایل‌ها به ازای هر درخواست افزایش می‌یاشد، و همان‌طور که قبلا نشان داده‌ام، کانکشن‌های بیشتر باعث می‌شود که مطمئن شویم که HTTP به خوبی کار می‌کند و صفحات سریع لود می‌شوند. این عجیب نیست که سایت‌ها از بیش از ۵۰ یا حتی ۱۰۰ کانکشن با این تکنیک استفاده کنند. آمارهای اخیر از httparchive.org نشان می‌دهد که ۳۰۰ هزار URL پربازدید جهان به طور متوسط به ۴۰ کانکشن TCP نیاز دارند! و آمار می‌گوید که این تعداد کانکشن‌ها به مرور زمان در حال افزایش است.

علت دیگر برای استفاده از تکنیک توزیع، قراردادن عکس‌ها و منابع مشابه در یک host name جداگانه‌ای که از Cookies استفاده نمی‌کنند است، چرا که امروزه حجم Cookies افزایش چشمگیری داشته است. با استفاده از میزبان‌هایی که کوکی ندارند، می‌توانید کارایی را با کاهش حجم درخواست‌های HTTP بالا ببرید!

عکس زیر، یکی از سایت‌های پرطرفدار سوئد را نشان می‌دهد که چگونه منابع مختلف خود را در چندین host name توزیع کرده است.



200 GET	174.jpg	w.cdn-expressen.se	jpeg	6.14 KB	→ 105 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.19 KB	→ 172 ms
200		dn-expressen.se	jpeg	4.48 KB	→ 223 ms
200		dn-expressen.se	jpeg	4.58 KB	→ 173 ms
200		dn-expressen.se	jpeg	35.18 KB	→ 56 ms
200		dn-expressen.se	jpeg	12.97 KB	→ 165 ms
200		dn-expressen.se	jpeg	4.83 KB	→ 56 ms
200		dn-expressen.se	jpeg	9.54 KB	→ 228 ms
200		dn-expressen.se	jpeg	182.50 KB	→ 285 ms
200		dn-expressen.se	jpeg	5.66 KB	→ 104 ms
200		dn-expressen.se	jpeg	12.24 KB	→ 287 ms
200		dn-expressen.se	jpeg	6.85 KB	→ 225 ms
200		dn-expressen.se	jpeg	7.50 KB	→ 173 ms
200		dn-expressen.se	gif	2.85 KB	→ 227 ms
200		dn-expressen.se	jpeg	50.87 KB	→ 188 ms
200		dn-expressen.se	jpeg	6.65 KB	→ 55 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	6.09 KB	→ 196 ms
200 GET	540.jpg	z.cdn-expressen.se	jpeg	16.14 KB	→ 67 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	19.89 KB	→ 112 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.03 KB	→ 55 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	21.27 KB	→ 108 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	5.43 KB	→ 237 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.08 KB	→ 169 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.62 KB	→ 105 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	20.32 KB	→ 241 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.66 KB	→ 55 ms
200 GET	540.jpg	x.cdn-expressen.se	jpeg	11.13 KB	→ 237 ms
200 GET	265.jpg	w.cdn-expressen.se	jpeg	5.20 KB	→ 111 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	6.93 KB	→ 288 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	12.09 KB	→ 249 ms
200 GET	265.jpg	z.cdn-expressen.se	jpeg	5.92 KB	→ 167 ms
200 GET	original.jpg	y.cdn-expressen.se	jpeg	64.28 KB	→ 192 ms
200 GET	original.jpg	w.cdn-expressen.se	jpeg	21.88 KB	→ 106 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	18.77 KB	→ 112 ms
200 GET	128.jpg	z.cdn-expressen.se	jpeg	3.34 KB	→ 55 ms
200 GET	265.jpg	x.cdn-expressen.se	jpeg	13.00 KB	→ 245 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	9.19 KB	→ 194 ms
200 GET	540.jpg	w.cdn-expressen.se	jpeg	13.13 KB	→ 108 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	5.66 KB	→ 197 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	5.56 KB	→ 55 ms
200 GET	174.jpg	w.cdn-expressen.se	jpeg	5.07 KB	→ 111 ms
200 GET	174.jpg	z.cdn-expressen.se	jpeg	6.16 KB	→ 59 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	6.57 KB	→ 210 ms
200 GET	174.jpg	y.cdn-expressen.se	jpeg	4.58 KB	→ 12 ms
200 GET	265.jpg	y.cdn-expressen.se	jpeg	11.49 KB	→ 173 ms

۴. آپدیت‌کردن HTTP

بهتر نیست که یک پرتکل بهتر بسازیم؟ پرتکلی که...

به تأخیرها کمتر حساس باشه
مشکل HTTP Pipelining رو و Head-of-line blocking رو حل کنه
نیازی به افزایش تعداد Host name نداشته باشه
از همین تعامل‌ها، محتوا و ساختارهای URI پشتیبانی کنه
و توسط کارگروه HTTPbis در IETF ساخته شده باشه!

۴.۱. IETF و کارگروه HTTPbis

IETF یک سازمان برای توسعه و ترویج استانداردهای اینترنت در سطح پرتکل است. این سازمان بیشتر به خاطر سری استانداردهای RFC شامل TCP، DNS، FTP و از همه بهتر HTTP و یکسری پرتکل‌های دیگر که هیچجا شناخته نشده‌اند، مشهور است.

در IETF، کارگروه‌های اختصاصی با اختیارات محدود برای رسیدن به یک هدف مشخص کار می‌کنند. آن‌ها یک منشور مشخص می‌کنند تا خطمشی‌ها و محدودیت‌ها برای چیزی که تولید می‌کنند را مشخص کنند. همه‌ی افراد اجازه‌ی مشارکت در بحث و توسعه را دارند. هر کسی که شرکت می‌کند و چیزی می‌گوید، گفته‌ی او، اهمیت یکسانی نسبت به گفته‌های دیگران دارد و هر کسی به عنوان یک فرد مستقل شناخته می‌شود، بدون درنظرگرفتن شرکتی که او در آن‌جا کار می‌کند.

کارگروه HTTPbis در تابستان ۲۰۰۷ شکل گرفت و وظیفه دارد تا استانداردهای HTTP را آپدیت کند. در این گروه، بحث درمورد نسخه‌ی بعدی HTTP در اواخر سال ۲۰۱۲ شکل گرفت. کار آپدیت HTTP 1.1 در اوایل ۲۰۱۴ تمام شد که نتیجه‌ی آن را در سری RFC 7230 می‌بینید.

آخرین نشست فنی کارگروه HTTPbis در ژوئن ۲۰۱۴ در شهر نیویورک برگزار شد. بحث‌های باقی‌مانده و روند رسمی IETF انجام شدند تا این استاندارد RFC به طور رسمی سال بعد عرضه شود.

بازیگران بزرگ‌تر در عرصه‌ی HTTP در جلسات و گفتگوهای این کارگروه غایب بودند. نمی‌خواهم نام هیچ شرکت یا محصول خاصی را ببرم، اما واضح است که بعضی از بازیگران اصلی اینترنت امروز، مطمئن هستند که IETF بدون آن‌ها هم عملکرد خوبی خواهد داشت...

۴.۱.۱. پسوند bis

نام گروه HTTPbis است که پسوند bis از در لاتین به معنای دو است. پسوند Bis معمولا در IETF برای هر آپدیت یا نسخه‌ی دوم هر چیزی استفاده می‌شود؛ مثلا همین به‌روزرسانی HTTP 1.1.

۴.۲. http2 از SPDY شروع شد

SPDY یک پرتکل است که توسط گوگل توسعه داده و توزیع شد. آن‌ها، این پرتکل را در یک محیط باز توسعه دادند و از همگان دعوت کردند که شرکت کنند ولی روشن است که آن‌ها با کنترل‌کردن پیاده‌سازی یک مرورگر پرطرفدار و همچنین سرورهای پرجمعیتی که از سرویس‌ها استفاده می‌کردند، سود می‌بردند.

هنگامی که گروه HTTPbis تصمیم گرفت که روی http2 کار کند، SPDY قبلا ثابت کرده بود که یک طرح عملی است. SPDY نشان داده بود که استفاده از آن در اینترنت ممکن است و آماری هم وجود دارد که تا چه حد خوب کار می‌کند. کار http2 با پیش‌نویس SPDY/3 شروع شد که به سادگی، تبدیل به پیش‌نویس صفر (HTTP2 (draft-00 با کمی تغییر شد.

۵. مفاهیم http2

http2 چه فایده‌ای دارد؟ مرزهای تعیین‌شده برای کارگروه HTTPbis چه هستند؟

این مرزها در واقع بسیار محدود بودند و اختیارات کمی به گروه برای نوآوری می‌دادند:

http2 باید از پارادایم‌های HTTP پشتیبانی کند. یعنی همچنان پرتکلی است که کلاینت از طریق TCP به سرور درخواستی می‌فرستند.

پیش‌وندهای `http://` و `https://` نباید تغییر کنند. هیچ پیشوند تازه‌ای نمی‌توان ایجاد کرد. محتوایی که تحت این پیشوندها دردسترس هستند، بیشتر از آن هستند که بتوان تغییرشان داد.

سرورها و کلاینت‌های HTTP1 تا دهه‌ها وجود خواهند داشت، پس باید بتوان آن‌ها را با سرورهای http2 پراکسی کرد.

تبعا، پراکسی‌ها باید بتوانند قابلیت‌های http2 را به کلاینت‌های HTTP 1.1، یک‌به‌یک مربوط کنند.

حذف یا کاهش قسمت‌های اختیاری پرتکل. این کار واقعا لازم نبود، در واقع یک حرکت بود که از گوگل و SPDY شروع شد. وقتی مطمئن باشیم که همه‌چیز ضروری هستند، می‌توانید بدون این‌که چیزی جا بیندازید، آن را پیاده‌سازی کنید که بعدا گرفتار نشوید.

نسخه‌های جزئی نداشته باشیم. ما تصمیم گرفتیم که کلاینت‌ها یا سرورها یا با http2 سازگارند یا نیستند. اگر نیاز به توسعه‌ی پرتکل وجود داشت، نسخه‌ی بعدی، http3 خواهد بود. هیچ نسخه‌ی جزئی (Minor) در http2 نخواهیم داشت.

۵.۱. http2 برای پیشوندهای URI موجود

همان‌طور که قبلا هم اشاره شد، پیشوندها و ساختارهای کنونی URI را نمی‌توان تغییر داد، پس http2 باید از آن‌هایی که الان هستند استفاده کند. از آن‌جایی آن‌ها در HTTP 1.x استفاده می‌شوند، به یک راه نیاز داریم که پرتکل را به http2 ارتقا دهیم یا از سرور بخواهیم که از http2 به جای پرتکل‌های قدیمی استفاده کند.

HTTP 1.1 قبلا یک راه برای این منظور تعریف کرده: استفاده از هدر
Upgrade: که به سرور اجازه می‌دهد که
پاسخ درخواست را با پرتکل جدید بدهد، که هزینه‌ی این کار، پذیرش دو درخواست به جای یکی
است.

این دوبرابر شدن درخواست‌ها، چیزی نبود که تیم SPDY بتواند قبول کند، و از آن‌جایی که آن‌ها SPDY را تنها بر روی TLS پیاده کرده بودند، یک افزونه برای TLS توسعه دادند که ارتباط اولیه (Negotiation) را به طور چشمگیری سریعتر و کوتاهتر می‌کرد. با این افزونه، که NPN نام دارد، سرور به کلاینت اطلاع می دهد که چه پرتکل‌هایی را می‌شناسد و کلاینت می‌تواند از پرتکلی که ترجیح می‌دهد استفاده کند.

۵.۲. http2 برای https://

تلاش‌های زیادی انجام شده که http2 بر TLS به درستی رفتار کند. SPDY به TLS نیاز دارد و این تصمیم مهمی بود که TLS را برای الزامی کنیم، ولی به یک نتیجه‌ی جمعی نرسیدیم، بنابراین http2 با TLS به صورت اختیاری منتشر شد. با این حال، دو پیاده‌سازی برجسته اعلام کردند که http2 تنها از طریق TLS دردسترس خواهد بود: فایرفاکس از موزیلا و کروم از گوگل، دو مرورگر پیشروی امروز.

دلایل اجباری‌کردن TLS، احترام به حریم‌خصوصی کاربر است، همچنین آزمایش‌های اولیه نشان داد که پرتکل‌های جدید، میزان موفقیت بیشتری دارند، اگر بر مبنای TLS باشند. این به این‌خاطر است که معمولا فرض بر این گذاشته می‌شود که ترافیکی که از پورت ۸۰ عبور می‌کند، با پرتکل HTTP 1.1 کار می‌کند. وقتی پرتکل‌های دیگری از این پورت استفاده می‌کنند، بعضی از واسطه‌ها (مثلا آنتی‌ویروس‌ها) ممکن است اختلال ایجاد کنند یا حتی داده‌های رسیده را از بین ببرند.

موضوع اجباری‌شدن TLS مناقاشات زیادی را در لیست‌های ایمیل و دیدارها به‌وجود آورد - آیا این کار درست است یا غلط؟ موضوعی که بسیار جدال‌آمیز است - مواظب باشید که آن را ناگهانی از یکی از اعضای کارگروه HTTPbis نپرسید!

به طور مشابه، بحثی هم درمورد این وجود داشته که آیا http2 باید لیستی از روش‌های رمزنگاری را برای TLS اجباری کند، یا لیست‌سیاهی از این الگوریتم‌ها درست کند، یا شاید هیچ‌کاری به لایه‌ی TLS نداشته باشد و اجازه دهد که کارگروه TLS کار خودشان را انجام دهند. نتیجه آن شد که استاندارد تعیین کرد که نسخه‌ی TLS باید حداقل ۱.۲ باشد و همچنین در انتخاب روش‌های رمزنگاری (Cipher Suite) نیز محدودیت‌هایی وجود دارد.

۵.۳. ارتباط اولیه http2 روی TLS

NPN پرتکلی بود که SPDY برای مذاکره یا همان ارتباط اولیه با سرورهای TLS استفاده می‌کرد. از آن‌جایی که این پرتکل استاندارد نبود، NPN از IETF گذشت و نتیجه شد: ALPN. ALPN در حال‌حاضر برای http2 استفاده می‌شود، در حالی که کلاینت‌ها و سرورهای SPDY هنوز از NPN استفاده می‌کنند.

در حقیقت، NPN اول وجود داشته و در زمانی که ALPN در فرآیند استانداردسازی قرار داشته، کلاینت‌ها و سرورهای http2 بسیاری به‌وجودآمدند که از هردوی آن‌ها پشتیبانی می‌کردند. همچنین، از NPN در SPDY استفاده می‌شود و سرورهای بسیاری SPDY و http2 ارائه می‌دهند. پس پشتیبانی همزمان از NPN و ALPN در این سرورها، کاملا منطقی است.

ALPN با NPN تفاوتشان در در این است که چه کسی تصمیم می‌گیرد که با چه پرتکلی صحبت کند. در ALPN، کلاینت لیستی از پرتکل‌هایی که پشتیبانی می‌کند را به سرور می‌دهد و سرور برحسب کارایی و اولویت، یکی را انتخاب می‌کند، در حالی که در NPN، کلاینت تصمیم نهایی را می‌گیرد.

۵.۴. http2 برای http://:

همان‌طور که قبلا اشاره کردم، برای HTTP 1.1 که از متن ساده (plain text) استفاده می‌کند، ارتباط اولیه‌ی http2 با هدر Upgrade: انجام می‌شود. اگر سرور به http2 صحبت می‌کند، با کد "Switching 101" پاسخ می‌دهد و پس از آن، با کلاینت http2 صحبت می‌کند. البته، این فرآیند، باعث دوبرابرشدن درخواست‌ها و پاسخ‌ها می‌شود، ولی مزیت آن این است که معمولا ممکن است یک کانکشن http2 را مدت بیشتری زنده نگه داشت و از آن استفاده بیشتری نسبت به یک کانکشن HTTP1 کرد.

در حالی که بعضی از سخنگوهای مرورگرها اعلام کردند این مورد را پیادهسازی می‌کنند، تیم Internet Explorer یک‌بار اعلام کردند که این کار را می‌کنند، هر چند تا به الان به قول خود عمل نکردند. همچنین curl و چند کلاینت دیگر که مرورگر نیستند اعلام کردند که از http2 به صورت متن‌ساده (clear-text/رمزنگاری‌نشده) پشتیبانی می‌کنند.

امروزه، هیچ مرورگری بدون TLS از http2 پشتیبانی نمی‌کند.

۶. پرتکل http2

فکر می‌کنم بحث درمورد پیش‌زمینه و اتفاقات گذشته بس باشد. برویم به سراغ استانداردهای پرتکل: چیزهایی که http2 را ساختند.

۶.۱. باینری

http2 یک پرتکل باینری است.

اگر شما با پرتکل‌های اینترنتی قبلا کار کرده باشید، احتمالا نسبت به این مورد واکنش نشان می‌دهید و شروع به ارائه‌ی دلایل مبنی بر این که چرا پرتکل‌های متنی (text/ascii) بهترند چون انسان‌ها می‌توانند آن‌ها را بخوانند، در telnet از آن‌ها استفاده کنند و ...

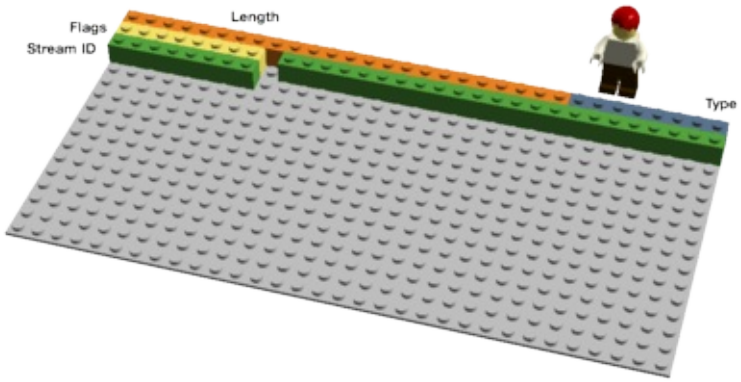
http2 باینری است تا فریم‌بندی را راحت‌تر کند. تشخیص اول و آخر یک فریم
در HTTP 1.1
و بقیه‌ی پرتکل‌های متنی، کار پیچیده‌ای بود. با دورشدن از فضاهای خالی اختیاری و راه‌های متفاوت برای پیاده‌سازی یک چیز [1]، پیاده‌سازی حالا راحت‌تر خواهد شد.

همچنین، جداکردن فریم‌های مربوط به قرارداد و پرتکل از فریم‌های مربوط به داده‌های پاسخ، راحت‌تر خواهد شد - قبلا این کار در HTTP1 بسیار گیج‌کننده بود.

در حقیقت، پرتکل قابلیت فشرده‌سازی را دارد و اجرا شدنش روی TLS، متن را مخفی می‌کند، بنابراین شخص ثالثی نمی‌تواند متن را از روی ترافیک جابجاشده بخواند. در واقع، باید عادت کنیم که از برنامه‌هایی مثل Wireshark برای خواندن داده‌های مبادله‌شده در سطح پرتکل http2 استفاده کنیم.

دیباگ‌کردن این پرتکل به ابزارهایی مثل curl یا برای آنالیز به Wireshark و مشابه‌های آن نیاز دارد.

۶.۲. قالب باینری



http2 فریم‌ها را به صورت باینری می‌فرستد. نوع فریم‌ها
ممکن است مختلف باشد، ولی همه‌ی آن‌ها یک‌نوع مشخصات دارند:
اندازه (Length)، نوع (Type)، نشان‌ها (Flags)،
شناسه‌ی استریم (Stream Identifier) و داده‌های فریم.

در قرارداد http2، ده نوع فریم مختلف تعریف شده و اساسی‌ترین آن‌ها که به قابلیت‌های HTTP 1.1 نیز مربوط هستند، DATA و HEADERS است. بعضی از این نوع‌ها را در ادامه بررسی می‌کنیم.

۶.۳. تسهیم‌سازی (Multiplexing)

شناسه‌ی استریم که در فریم قبلی به آن اشاره شد، با یک «استریم» در هر فریم همراه است. یک استریم، مجموعه‌ای از فریم‌های داده به طور مستقل و دوطرفه است که کلاینت و سرور می‌توانند در یک کانکشن http2 آن‌ها را مبادله کنند.

یک کانکشن مستقل http2 می‌تواند دارای چندین استریم‌های باز به طور همزمان باشد، و یا یکی از طرفین، استریم‌های مختلف را با هم ترکیب کند تا چندین فریم را بسازد.

تسهیم‌سازی استریم به این معنی است که بسته‌های چندین استریم با هم ترکیب شده و در قالب یک کانکشن مبادله می‌شودن. دو (یا بیشتر) قطار از داده‌ها، تبدیل به یک قطار می‌شوند و در طرف دیگر، دوباره جدا می‌شوند. مثلا:

حالا اگر این دو قطار با هم ترکیب شوند یا اصطلاحا تسهیم‌سازی (Multiplex) شوند:

۶.۴. اولویت‌ها و وابستگی‌ها

هر استریم دارای اولویت است (همچنین آن را به عنوان weight هم می‌شناسند)، که به طرف مقابل اطلاع می‌دهد که کدام استریم مهم‌تر است تا با توجه به محدودیت منابع، کدام را زودتر دریافت کند یا کدام را زودتر درخواست کند.

با استفاده از فریم PRIORITY، کلاینت می‌تواند به سرور اطلاع دهد که یک استریم، به چه استریم‌های دیگری وابسته است. این قابلیت به کلاینت اجازه می‌دهد که یک درخت از وابستگی‌ها ایجاد کند که هر «استریم فرزند» به «استریم‌های پدر» وابسته است.

اولویت‌ها و وابستگی‌ها می‌توانند در زمان اجرا به صورت پویا عوض شوند، که باعث می‌شود مرورگرها این توانایی را داشته باشند که هنگامی که کاربر در یک صفحه‌ی پر از عکس، پیمایش (Scroll) می‌کند، مرورگر به سرور اطلاع دهد که کدام عکس‌ها مهم‌تر هستند، یا اگر به یک Tab دیگر بروید، مرورگر به استریم‌های دیگری اولویت بالاتر می‌دهد که مربوط به صفحه‌ی جدید هستند.

۶.۵. فشرده‌سازی هدرها

HTTP یک پرتکل Stateless
است، یعنی هر درخواست باید هدرهایی را همراه داشته باشد تا سرور بتواند به آن پاسخ دهد، بدون اینکه نیاز باشد سرور اطلاعات بسیاری را از درخواست‌های قبلی ذخیره کند. از آن‌جایی
ک
ه
http2 این پارادایم را تغییر نمی‌دهد، همچنان به همین صورت کار
می‌کند.

این روش باعث تکراری‌شدن درخواست‌های HTTP می‌شود. وقتی یک کلاینت از همان سرور، چندین بار درخواست عکس برای یک صفحه می‌کند، چندین درخواست تقریبا یکسان را می‌فرستند. این نوع داده‌هایی که تقریبا یکسان هستند، مناسب فشرده‌سازی هستند.

همان‌طور که قبلا هم اشاره کردیم، تعداد فایل‌های موردنیاز هر صفحه بالاتر رفته، همچنین اندازه‌ی Cookies و درخواست‌ها هم به مرور زمان در حال افزایش هستند. Cookies باید در هر درخواست فرستاده شوند، معمولا در چندین درخواست یکی هستند.

اندازه‌ی درخواست‌های HTTP 1.1 در واقع این‌قدر بزرگ شده‌اند که گاهی وقت‌ها بیش از حجم مقبول TCP می‌شوند که باعث می‌شود ارسال‌شان بسیار کندتر شود، چون یک‌بار ناقص فرستاده می‌شوند تا از سرور ACK بگیرند و بعد از آن، به طور کامل ارسال می‌شود. این دلیل دیگری برای نیازداشتن به فشرده‌سازی است.

۶.۵.۱. فشرده‌سازی موضوع ریسک‌داری است

فشرده‌سازی‌های HTTPS و SPDY دربرابر حملات BREACH و CRIME آسیب‌پذیر بودند. با قراردادن یک متن شناخته در استریم و یافتن این‌که چگونه تغییر می‌کند، حمله‌کننده می‌توانست بفهمد که چیزی به صورت Encrypt شده ارسال می‌شده است.

فشرده‌سازی محتوای پویا در یک پرتکل - بدون این‌که به یکی از این حملات آسیب‌پذیر باشد - به فکر و تصمیم‌های محتاطانه نیاز دارد. این همان چیزی است که کارگروه HTTPbis سعی کردند انجام دهند.

[HPACK] وارد می‌شود!(https://www.rfc-editor.org/rfc/rfc7541.txt). Header Compression for HTTP/2 با فشرده‌سازی هدرها
برای HTTP/2، یک فرمت فشرده‌سازی است که مخصوصا برای
هدرهای http2 طراحی شده است و در یک پیش‌نویس اینترنتی جداگانه تبیین
شده.

به گفته‌ی Roberto Peon (یکی از سازندگان HPACK): *(مترجم: این نقل‌قول به طور تحت‌اللفظی ترجمه نشده).*

> HPACK
> احتمال ناامن‌بودن پیاده‌سازی و در نتیجه نشت اطلاعات را کاهش می‌دهد روند رمزنگاری/رمزگشایی را آسان‌تر و به‌صرفه‌تر می‌کند، و به دریافت‌کننده این امکان
> را می‌دهد که
> ب
> ر
> Context Size فشرده‌سازی کنترل داشته باشد یا اجازه‌ی ایجاد پراکسی برای
> Re-Indexing
> (مثل
> وضعیت مشترک بین فرانت‌اند و بک‌اند در یک پراکسی) را بدهد و همچنین امکان مقایسه‌ی سریع رشته‌هایی که به روش هافمن رمزنگاری شده‌اند را
> می‌دهد.

۶.۶. ریست - نظرم عوض شد

یکی دیگر از معایب HTTP 1.1، عدم وجود پشیمانی است! یعنی وقتی یک درخواست با Content-Length مشخص فرستاده شد، نمی‌توانید به این راحتی متوقفش کنید. البته، شما می‌توانید کانکشن TCP را قطع کنید، ولی باید دوباره یک ارتباط TCP جدید ایجاد کنید.

راه‌حل بهتر این است که درخواست قبلی را متوقف کنید و یک درخواست جدید بفرستید. این کار در http2 با ارسال فریم RST_STREAM ممکن است که از هدررفت پهنای باند و همچنین از بین رفتن کانکشن‌ها جلوگیری می‌کند.

۶.۸. Push از سرور

این ویژگی را با نام «cache push» نیز یاد می‌کنند. ایده‌ی اصلی این است که هنگامی که کلاینت درخواست فایل X را می‌کند، سرور می‌داند که احتمالا فایل Z را هم می‌خواهد و آن را به کلاینت، بدون این‌که درخواست کرده باشد، می‌فرستد. این ویژگی باعث می‌شود که کلاینت ، فایل Z را هم در Cache قرار دهد و هنگامی که نیاز است از آن استفاده کند.

Server push ویژگی‌ای است که کلاینت باید به
سرور اجازه‌ی انجام آن را بدهد. در آن صورت، کلاینت می‌تواند یک استریم پوش‌شده را در هر زمانی
با RST_STREAM کنسل کند.

۶.۸. کنترل جریان

هر استریم http2، ظرفیت جریان را تعیین می‌کند تا سمت دیگر مطلع باشد که چه مقدار می‌تواند داده بفرستید. اگر با پرتکل SSH و چگونگی کار آن آشنا باشید، می‌دانید که این ویژگی بسیار شبیه به SSH است.

در هر استریم، هر دو طرف باید به طرف دیگر اطلاع دهند که فضای کافی برای دریافت داده‌ها را دارند و طرف دیگر تنها در صورتی اجازه‌ی ارسال دارد که ظرفیت اجازه می‌دهد. تنها فریم‌های DATA کنترل می‌شوند.

[1]. منظور از این فریم، معایب متنی‌بودن پرتکل است. مثلا در هدر، هم `Content-Type:text/html` مقبول است و هم `content-type: text/html` . ↩

۷. افزونه‌ها

پرتکل http2 فرض می‌کند که گیرنده باید همه‌ی فریم‌های ناشناخته را بخواند و نادیده بگیرد (فریم‌هایی که نوع ناشناخته‌ای دارند). دو طرف می‌توانند نوع فریم‌های جدیدی را تعیین کنند، اما این فریم‌ها نمی‌توانند وضعیت خود را تغییر دهند و جریان آن‌ها کنترل نمی‌شوند.

موضوع این که آیا http2 اجازه‌ی افزونه‌ها را بدهد یا نه، در طول توسعه‌ی پرتکل سر آن بحث‌های زیادی انجام گرفت که توسعه‌دهندگان نظر خود را مرتبا عوض می‌کردند. بعد از پیش‌نویس ۱۲، در نهایت افزونه‌ها در http2 مجاز شدند.

افزونه‌ها جزئی از خود پرتکل نیستند و مستندات آن‌ها در خارج از هسته‌ی استانداردهای پرتکل قرار دارند. قبلا دو نوع فریم در پرتکل به عنوان افزونه‌ها تعریف شده. آن‌ها را اینجا به دلیل محبوبیت و این‌که قبلا جزئی از پرتکل بودند، توضیح می‌دهم.

۷.۱. سرویس‌های جایگزین

با ورود http2، کانکشن‌های TCP می‌توانند به هر دلیلی، حجیم‌تر شوند و مدت برقراری آن‌ها نسبت به کانکشن‌های HTTP 1.x افزایش یابند. یک کلاینت باید بتواند این مشخصه‌ها را در یک کانکشن افزایش دهد، و این کانکشن می‌تواند برای مدت نسبتا زیادی برقرار باشد.

این موضوع روی چگونگی کار Load Balancerهای HTTP تأثیر می‌گذارد و ممکن است موجب شرایطی شود که یک سایت به کلاینت پیشنهاد دهد که به یک سرور دیگر متصل شود. ممکن است دلیل این کار، افزایش سرعت و کارایی باشد، یا سرور تحت تعمیر باشد و سرور دیگری دردسترس باشد و غیره.

سرور می‌تواند هدر Alt-Svc (یا فریم ALTSVC) را به کلاینت بفرستد که سرویس دیگری موجود است: یک مسیر دیگر به همان محتوا، در یک سرور دیگر، میزبان دیگر و پورت دیگر.

کلاینت باید به آن سرویس دیگر به طور ناهمگام وصل شود و فقط در صورتی از جایگزین استفاده کند که کانکشن جدید موفق باشد.

۷.۱.۱. TLS فرصت‌گرا!

هدر Alt-Svc به سرور اجازه می‌دهد که محتوا را بر پرتکل http:// ارائه دهد تا به کلاینت اطلاع دهد که همان محتوا روی یک کانکشن امن TLS هم دردسترس هستند.

این قابلیت کمی شک‌برانگیز است. این نوع کانکشن می‌تواند TLS احرازنشده انجام دهد و دیگر نتواند امن معرفی شود، هیچ علامت قفلی در UI نشان داده می‌شود و در واقع هیچ راهی وجود ندارد که به کاربر بگوییم که این همان HTTP ساده و قدیمی است، ولی همچنان این TLS فرصت‌گراست و بعضی از افراد شدیدا مخالف این مفهوم هستند.

۷.۲. بلاک‌شده

این نوع فریم وقتی ارسال می‌شود که یکی از طرفین کانکشن http2 داده‌ای را ارسال می‌کند، اما کنترل جریان اجازه‌ی ارسال را نمی‌دهد. ایده این است که اگر کلاینت یا سرور این فریم را دریافت کند، شما می‌دانید که یک جای کار مشکل دارد و یا سرعت انتقال کمتر است.

متن زیر قسمتی از پیش‌نویس ۱۲ است، قبل از این که این نوع فریم، افزونه شود.

> «فریم BLOCKED در این پیش‌نویس گنجانده شده تا آزمایش‌ها را تسهیل کند. اگر نتایج آزمایش‌ها، بازخورد مثبتی نداشته باشد، ممکن است حذف شود.»

۸. دنیایی با http2

خب، وقتی http2 تصویب شود، دنیا چگونه خواهد شد؟ اصلا تصویب خواهد شد؟

۸.۱. http2 چگونه روی انسان‌های عادی تأثیر می‌گذارد؟

http2 هنوز به‌طور گسترده نه پخش شده نه استفاده. ما نمی‌توانیم
دقیقا بگوییم که چیزها چگونه تغییر خواهند کرد. ما دیدیم که
SPDY مورداستفاده قرار گرفت و می‌توان با کمی محاسبات، حدس‌های نسبتا
دقیقی براساس آزمایش‌های قبلی و کنونی زد.

http2 تعداد رفت و برگشت‌ها در شبکه را کاهش می‌دهد، مشکل Head-of-line blocking را با طور کامل با Multiplexing و پس‌زدن سریع استریم‌های ناخواسته، حل می‌کند.

این پرتکل اجازه می‌دهد که تعداد زیادی از استریم‌های موازی استفاده شود، حتی بیش‌تر از آن‌چه که سایت‌های توزیع‌شده (Sharded) ارائه می‌دهند.

با اختصاص اولویت‌های صحیح به استریم‌ها، احتمال این که کلاینت‌ها، اول داده‌های مهم‌تر را دریافت می‌کنند بیش‌تر می‌شود. با همه‌ی این‌ها، می‌توانم بگویم که احتمال این که این پرتکل به سریع‌ترشدن بارگذاری صفحات و پاسخ‌گویی آن‌ها منجر می‌شود، بسیار بالاست. خلاصه این‌که: یک تجربه‌ی بهتر از وب.

این‌که چقدر سریع‌تر و بهتر، خواهیم دید، ولی فکر نمی‌کنم الان بتوانیم چیزی بگوییم. اول این‌که این تکنولوژی هنوز بسیار جوان است و ما هنوز حتی شروع نکرده‌ایم که ببینیم آیا کلاینت‌ها و سرورها و به طور کلی «پیاده‌سازی‌ها» از همه‌ی قدرت این پرتکل جدید استفاده خواهند کرد یا نه.

۸.۲. چگونه http2 بر توسعه‌ی وب تأثیر خواهد گذاشت؟

در طول این سال‌ها، توسعه‌دهندگان وب، جعبه ابزاری از ترفندها و ابزارها برای حل مشکلات HTTP 1.1 فراهم کرده‌اند، در اول این کتاب به بعضی از این مشکلات و راه‌حل‌ها اشاره کرده‌ام.

بسیاری از این راه‌حل‌هایی که ابزارها و توسعه‌دهندگان، این روزها به طور پیش‌فرض و بدون فکر استفاده می‌کنند، احتمالا به کارایی http2 آسیب خواهند زد یا حداقل از ابرقدرت‌های جدید http2 بهره نخواهند برد. Spriting و Inlining نباید در http2 انجام شوند. Sharding یا توزیع‌کردن هم به کارایی http2 آسیب می‌زند و احتمالا تنها از تعداد کانکشن‌های کمتر سود خواهد برد.

مشکلی که اینجاست، این است که وب‌سایت‌ها و توسعه‌دهندگان آن‌ها باید در مدت کوتاهی، محصولات خود را برای دنیایی ارائه دهند که در آن هم کاربران HTTP 1.1 وجود دارند هم http2 ارائه دهند. ارائه‌ی حداکثر سرعت و کارایی برای همه‌ی کاربران بدون ارائه‌ی دو نوع فرانت‌اند، چالش‌برانگیز خواهد بود.

به همین دلیل‌ها، احتمالا مدتی طول خواهد کشید که ببینیم از همه‌ی ظرفیت‌های http2 استفاده می‌شود.

۸.۳. پیاده‌سازی‌های http2

تلاش‌کردن برای مستندکردن پیادسازی‌های خاص در چنین نوشته‌ای، کار بیهوده‌ای است و تنها بعد از مدتی، قدیمی می‌شود. به جای این کار، شرایط به طور عمومی‌تری توضیح می‌دهم و خوانندگان را به لیست پیادسازی‌ها در وب‌سایت http2 ارجاع می‌دهم.

در ابتدا، پیادسازی‌های بسیاری وجود داشتند و تعداد آن‌ها به مرور زمان افزایش پیدا کرد. در هنگام نوشتن این کتاب، حدود ۴۰ پیادسازی لیست شده‌اند و بیشتر آن‌ها ورژن نهایی را پیاده کرده‌اند.

۸.۳.۱. مرورگرها

فایرفاکس مرورگری بوده که همواره آخرین ویژگی‌ها را پیادسازی کرده، توییتر نیز سرویس‌هایش را بر پرتکل http2 ارائه می‌دهد. گوگل از آپریل ۲۰۱۴ شروع به پشتیبانی از http2 در چند سرور آزمایشی کرد و از می ۲۰۱۴، پشتیبانی از http2 را در ورژن‌های توسعه‌دهندگان Chrome کردند. ماکروسافت نیز یک پیش‌نمایش از پشتیبانی از http2 در نسخه‌ی بعدی Internet Explorer نشان دهند. سافاری (در iOS 9 و Mac OS X El Capitan) و اپرا نیز هر دو اعلام کردند که به زودی پشتیبانی خواهند کرد.

۸.۳.۲. سرورها

پیاده‌سازی‌های زیادی از http2 در سمت سرور وجود دارد.

سرور nginx از http2 از نسخه‌ی 1.9.5 که در سپتامبر ۲۰۱۵ منتشر شد (این قابلیت، جایگزین ماجول SPDY شد تا نتوانند هر دو در یک سرور اجرا شوند.)

سرور httpd آپاچی نیز یک ماجول http2 به نام mod_http2 از نسخه‌ی ۲.۴.۱۷ دارد که در نهم اکتبر ۲۰۱۵ منتشر شد.

Caddy, nghttp2, Apache Traffic Server, H2O و LiteSpeed نیز از http2 پشتیبانی می‌کنند.

۸.۳.۳. بقیه‌ی نرم‌افزارها

curl و libcurl از http2 ناامن و همچنین امن بر مبنای TLS پشتیبانی می‌کنند.

Wireshark که بهترین ابزار برای آنالیز ترافیک http2 است نیز پشتیبانی می‌کند.

۸.۴. نقدهای رایج http2

هنگام توسعه‌ی این پرتکل، گاهی افراد شک می‌کردند که احتمالا نتیجه از سوی بعضی به عنوان یک پرتکل اشتباه خوانده خواهد شد. بعضی از انتقادات وارد به این پرتکل و توجیه درست یا غلطبودن آن‌ها را در ادامه می‌آورم.

۸.۴.۱. «این پرتکل توسط گوگل ساخته یا طراحی شده است»

این دیدگاه، به روش‌های دیگری هم بیان می‌شود که در آینده، دنیا به گوگل وابسته خواهد شد و یا کنترل می‌شود. این درست نیست. این پرتکل در IETF و به همان روشی که ۳۰ سال است پرتکل‌ها طراحی می‌شوند، طراحی شد. اما، همه‌ی ما کار تأثیرگذار گوگل با SPDY را دیدیم که ثابت کرد که نه تنها ممکن است که یک پرتکل جدید منتشر کن بلکه آمارها نشان دادند که چه چیزهایی حاصل خواهد شد.

گوگل به طور عمومی اعلام کرد که پشتیبانی از SPDY و NPN را از Chrome در ۲۰۱۶ حذف می‌کنند و سرورها را مجبور به مهاجرت به HTTP/2 می‌کنند. در فبریه‌ی ۲۰۱۶ آن‌ها اعلام کردند که SPDY و NPN بالاخره در Chrome 51 حذف شدند.

۸.۴.۲. «این پرتکل فقط در مرورگرها کاربردی است»

این دیدگاه تا حدودی درست است. یکی از انگیزه‌های اصلی توسعه‌ی http2، حل‌کردن مشکل HTTP Pipelining است. اگر برنامه‌ی شما نیازی به این تکنولوژی ندارد، پس احتمالا http2 تأثیر مثبت چندانی بر برنامه‌ی شما نخواهد گذاشت. البته، HTTP Pipelining تنها قابلیت اضافه‌شده در این پرتکل نیست.

وقتی سرویس‌ها متوجه شوند که قدرت و توانایی‌های استریم‌های Multiplexed در یک کانکشن چقدر است، احتمال می‌دهم که اپلیکیشن‌های بیشتری از http2 استفاده کنند.

REST APIهای کوچک و برنامه‌های کوچک مبتنی بر HTTP 1.x احتمالا دلیلی
برای مهاجرت به http2 نخواهند یافت. ولی همچنین، معایب‌های خیلی کمی
برای کاربران این سرویس‌ها بر بستر http2 به چشم می‌آید.

۸.۴.۳. «این پرتکل فقط برای سایت‌های بزرگ کاربردی است»

نه اصلا! قابلیت‌های Multiplexing به بهبود تجربه‌ی وب‌گردی در کانکشن‌هایی با تأخیر زیاد حتی در سایت‌های کوچکی که توزیع جغرافیایی (CDN) ندارند نیز کمک می‌کند. سایت‌های بزرگ معمولا بسیار سریع‌ترند و از سرورهای بیشتری برای کاهش زمان دریافت داده‌ها استفاده می‌کنند.

۸.۴.۴. «استفاده از TLS آن را کندتر می‌کند»

از جهاتی این مورد می‌تواند صحیح باشد. ارتباط اولیه (Handshake) در TLS می‌تواند سرعت را تا حدی کاهش دهد، ولی کارهایی در حال انجام‌شدن است تا این فرآیند سریع‌تر شود. ولی سربار حاصل از جایگزینی متن ساده با TLS خصوصا در CPU و انرژی، قابل‌چشم‌پوشی نیست، در حالی که اگر همان ترافیک و داده‌ها با متن ساده رد و بدل شوند، انرژی کمتری می‌برد. این که میزان تأثیر چه‌قدر است، مورد بحث بوده و اندازه‌گیری‌هایی نیز در این رابطه انجام شده است. برای مثال سایت istlsfastyet.com از نمونه‌ی یکی از این منابع برای آزمایش‌های انجام‌شده است.

برای مثال Telecom و دیگر اپراتورهای شبکه، در اتحاد وب باز ATIS، اعلام کرده‌اند که به ترافیک رمزنگاری‌نشده نیاز دارند تا بتوانند Caching، فشرده‌سازی و تکنیک‌های دیگر برای ارائه‌ی تجربه‌ی وب سریع‌تر را در اختیار کاربران از طریق ماهواره‌ها قرار دهند. http2 استفاده از TLS را اجبار نمی‌داند، بنابراین نباید با این بندها تداخلی داشته باشد.

بسیاری از کاربران اینترنت اعلام کرده‌اند که استفاده از TLS را به‌طور گسترده‌تر ترجیح می‌دهند، چرا که باعث می‌شود حریم خصوصی کاربران محافظت شود.

آزمایش‌ها همچنین نشان داده‌اند که با استفاده از TLS، شانس بیشتری نسبت به پیاده‌سازی پرتکل‌های متن ساده برای موفقیت وجود دارد، چرا که پرتکل‌های متن‌ساده‌ای که بر پورت ۸۰ پیاده‌سازی می‌شوند، موانع زیادی در سرراه خود دارند که ممکن است اختلال ایجاد کنند، چرا که فکر می‌کنند HTTP 1.1 است که در پورت ۸۰ رد و بدل می‌شود.

در آخر، به لطف استریم‌های Multiplexشده‌ی http2 بر روی یک کانکشن، مرورگرهای معمولی می‌توانند ارتباط‌های اولیه‌ی خیلی کمتری بر بستر TLS انجام دهند و در نتیجه سریع‌تر از HTTPS در HTTP 1.1 عمل کنند.

۸.۴.۵. «ASCII نبودن آن یک مشکل است»

بله، ما دوست داریم که اطلاعاتی که پرتکل‌ها مبادله می‌کنند را به طور واضح ببینیم، چرا که ردگیری و دیباگ‌کردن آن‌ها را آسان می‌کند. ولی پرتکل‌های برمبنای‌متن به خطا حساس هستند و مشکلات زیادی را برای پردازش متن‌های گرفته‌شده دارند.

اگر شما نمی‌توانید یک پرتکل باینری را تحمل کنید، بنابراین نباید اصلا از TLS و فشرده‌سازی در HTTP 1.x استفاده کنید، با این که هر دو مدت زیادی است که وجود دارند.

۸.۴.۶. «سرعت آن با HTTP/1.1 فرق خاصی ندارد»

این موضوعی شگفت‌انگیز و بحث‌برانگیز است که دقیقا سریع‌تر چه معنایی دارد، ولی در SPDY، قبلا آزمایش‌های زیادی انجام شده که ثابت می‌کند سرعت بارگذاری صفحات در مرورگرها بیشتر می‌شود (مثل "How Speedy is SPDY?" که توسط افرادی در دانشگاه واشنگتن تهیه‌شده یا "Evaluating the Performance of SPDY-enabled Web Servers" که توسط Hervé Servy نوشته شده) و چنین آزمایش‌هایی با http2 هم تکرارشده‌اند. من منتظر دیدن آزمایش‌های بیشتری هستم. یک تست اولیه‌ی ساده توسط httpwatch.com ساخته‌شده که نشان می‌دهد HTTP/2 به قول‌های خود وفا می‌کند.

۸.۴.۷. «این پرتکل به لایه‌بندی‌ها تعدی کرده!»

جدا این نقد شماست؟ لایه‌ها جزئی از یک دین مقدس جهانی نیستند که نتوان به آن‌ها دست زد. ما به منطقه‌های خاکستری (نسبتا خطرناک) وارد شدیم تا http2 را یک پرتکل خوب و مؤثر در چارچوب مرزها کنیم.

۸.۴.۸. «این پرتکل بعضی از نقض‌های HTTP/1.1 را رفع نمی‌کند»

این درست است. چون هدف ما، حفظ پارادایم‌های HTTP/1.1 بود، بعضی از قابلیت‌های قدیمی HTTP باید می‌ماندند، مثل هدرهای مرسوم که معمولا شامل Cookies هم می‌شدند، هدرهای احراز هویت و غیره. ولی مزیت نگه‌داشتن این پارادایم‌ها این است که ما پرتکلی داریم که انتشار آن، بدون حجم زیادی از کار برای به‌روزرسانی آن و جایگزینی زیرساخت‌های قبلی، ممکن است. http2 اساسا فقط یک لایه‌ی جدید برای Framing است.

۸.۵. آیا http2 همه‌گیر خواهد شد؟

هنوز بسیار زود است که بتوانیم با یقین حرف بزنیم، ولی می‌توانیم حدس‌هایی بزنیم که آن را در ادامه می‌آورم.

مخالفان خواهند گفت که «ببینید که IPv6 چقدر خوب عمل کرده!» تا مثالی برای پرتکل جدیدی بیاورند که دهه‌ها طول کشید تا به طور جهانی گسترش یابد. البته، پرتکل http2 اصلا شبیه IPv6 نیست. این پرتکلی بر مبنای TCP است و از همان مکانیزم‌های آپگرید HTTP، همان شماره‌ی پورت‌ها و همان TLS استفاده می‌کند و نیازی به تغیر روترها و فایروال‌ها ندارد.

گوگل با SPDY به دنیا ثابت کرد که یک پرتکل مانند این می‌تواند به طور جهانی گسترش یابد و در مرورگرها و سرویس‌ها با پیاده‌سازی‌های مختلف در مدت زمان نسبتا کوتاهی استفاده شود. در حالی که تعداد سرویس‌هایی که SPDY را ارائه می‌دهند، در حدود ۱٪ است، ولی مقدار داده‌هایی که این سرویس‌ها مبادله می‌کنند بسیار بزرگ است. بعضی از پرطرفدارترین وب‌سایت‌های امروزی، SPDY ارائه می‌دهند.

http2 بر مبنای همان پارادایم‌های SPDY است، می‌توانم بگویم که احتمال گسترش آن نسبت
به SPDY بسیار بیشتر است، چرا که استانداردی است که از
سوی IETF ارائه می‌شود. توسعه‌ی SPDY همیشه با توجیه این‌که «پرتکل ساخته‌ی گوگل است» پس زده شده.

مرورگرهای بسیاری از این به‌روزرسانی حمایت کردند. نمایندگانی از Firefox, Chrome, Safari, Internet
Explorer و Opera اعلام کرده‌اند که مرورگرهای http2 خود را ارائه کرده‌اند یا خواهند کرد.

اپراتورهای سرویس‌های بزرگ به زودی http2 را ارائه خواهند داد، مانند گوگل، توئیتر و فیس‌بوک. امیدواریم که به زودی پشتیبانی از http2 را در پیاده‌سازی‌های سمت‌سرور مانند Apache httpd و nginx ببینیم. H2O نیز یک سرور HTTP سریع است که از http2 نیز پشتیبانی می‌کند.

بعضی از بزرگ‌ترین فروشندگان پراکسی، مثل HAProxy, Squid و Varnish نیز تمایل خود را برای پشتیبانی از http2 اعلام کرده‌اند.

در سراسر سال ۲۰۱۵، ترافیک http2 در حال افزایش بوده است. در اوایل سپتامبر، سهم فایرفاکس ۴۰ از ترافیک HTTP حدود ۱۳٪ و از HTTPS حدود ۲۷٪ بوده، در حالی که گوگل حدود ۱۸٪ از درخواست‌های ارسالی را HTTP/2 می‌دانسته. باید به این نکته توجه داشت که گوگل در حال آزمایش پرتکل‌های جدیدتری است (QUIC را در قسمت ۱۲.۱ ببینید) که میزان استفاده‌ی http2 را از آن‌چه که می‌توانست باشد، کمتر می‌کند.

۹. http2 در فایرفاکس

فایرفاکس، پیش‌نویس‌های مربوط به http2 را از نزدیک دنبال می‌کرده و تست‌های مربوط به پیاده‌سازی‌ها را ماه‌ها قبل ارائه داده است. در هنگام توسعه‌ی http2، کلاینت‌ها و سرورها باید بر یک نسخه‌ی پیش‌نویس توافق می‌کردند که اجرای آزمایش‌ها را مقداری آزاردهنده می‌کرد. آگاه باشید که کلاینت و سرور بر روی کدام پیش‌نویس پرتکل پیاده‌سازی‌شده توافق می‌کنند.

۹.۱. اول، مطمئن شوید که فعال است.

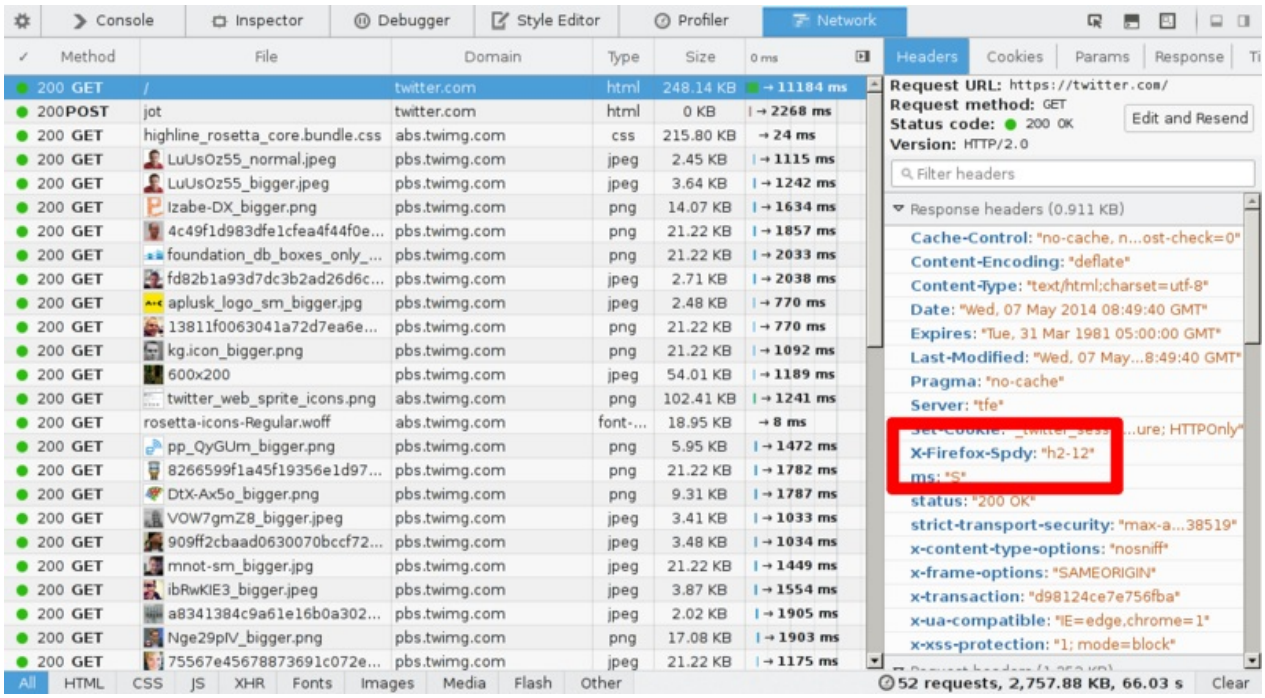
در فایرفاکس ۳۵ و بالاتر که در ۱۳ ژانویه‌ی ۲۰۱۵ منتشر شد، پشتیبانی از http2 به طور پیش‌فرض فعال است.

در آدرس‌بار مرورگر عبارت `about:config` را وارد کنید و به دنبال گزینه‌ای به نام `network.http.spdy.enabled.http2draft` بگردید. مطمئن باشید که مقدار آن *true* است. فایرفاکس ۳۶، گزینه‌ی دیگری به نام `network.http.spdy.enabled.http2` را اضافه کرده که مقدار آن به طور پیش‌فرض `true` است. گزینه‌ی دوم، نسخه‌ی «ساده»ی http2 را کنترل می‌کند، در حالی که اولی نسخه‌های پیش‌نویس را کنترل می‌کند.

۹.۲. فقط TLS

به یاد داشته باشید که فایرفاکس از http2 تنها در بستر TLS پشتیبانی می‌کند، یعنی فقط سایت‌هایی که با https:// شروع می‌شوند.

۹.۳. پشت‌صحنه!



هیچ عنصر بصری وجود ندارد که مشخص کند که دارید از پرتکل http2 استفاده می‌کنید. مشخص‌کردن آن هم توسط شما کار چندان آسانی نیست. یک راه برای تشخیص آن این است که در قسمت «Web developer->Network»، قسمت هدر پاسخ‌ها را چک کنید و ببینید که چه چیزی از سرور می‌گیرید. اگر پاسخ سرور HTTP/2.0 است که فایرفاکس هدر خودش را با نام `X-Firefox-Spdy:` اضافه می‌کند که در اسکرین‌شات بالا نشان داده‌ام.

هدرهایی که در ابزار Network می‌بینید، از فرمت باینری http2 به سبک قدیمی HTTP 1.x تبدیل شده‌اند.

۹.۴. نشان‌دادن استفاده از http2

پلاگین‌هایی برای فایرفاکس وجود دارند که نشان می‌دهند که یک سایت از http2 استفاده می‌کند یا نه. یکی از آن‌ها "HTTP/2 and SPDY Indicator" است.

۱۰. http2 در کرومیوم

تیم کرومیوم http2 را پیاده‌سازی کرده‌اند و پشتیبانی نیز برای آن ارائه می‌دهند. از کروم ۴۰ که در ۲۷ ژانویه‌ی ۲۰۱۵ منتشر شد، http2 برای تعداد مشخصی از کاربران فعال شد. این تعداد در ابتدا بسیار کم و به‌مرور زمان افزایش یافت.

پشتیبانی SPDY در کروم ۵۱ به خاطر http2 حذف شد. در یک پست بلاگ، این پروژه در فبریه‌ی ۲۰۱۶ اعلام شد:

> «بیش از ۲۵٪ منابع در کروم اکنون در بستر HTTP/2 کار می‌کنند، در مقایسه با کمتر از ۵٪ بر بستر SPDY. بر اساس چنین تأثیری، از ۱۵ می - سالروز انتشار HTTP/2 RFC - کروم دیگر از SPDY پشتیبانی نخواهد کرد.»

۱۰.۱. اول، مطمئن شوید که فعال است

عبارت `chrome://flags//#enable-spdy4` را در آدرس‌بار مرورگر خود وارد کنید و بر روی «enable» کلیک کنید، اگر قبلا فعال نشده است.

۱۰.۲. فقط TLS

به یاد داشته باشید که Chrome از http2 تنها در بستر TLS پشتیبانی می‌کند، یعنی فقط سایت‌هایی که با https:// شروع می‌شوند.

۱۰.۳. نشان‌دادن استفاده از http2

افزونه‌هایی برای Chrome وجود دارد که نشان می‌دهد که سایت از HTTP/2 استفاده می‌کند یا نه. یکی از آن‌ها "HTTP/2 and SPDY Indicator" است.

۱۰.۴. QUIC

آزمایش‌های کنونی Chrome با QUIC مقداری آمارهای HTTP/2 را تغییر می‌دهند. قسمت ۱۲.۱ را برای اطلاعات بیشتر درمورد QUIC ببینید.

۱۱. curl در http2

پروژه‌ی curl به طور آزمایشی پشتیبانی از http2 را از سپتامبر ۲۰۱۳ ارائه می‌دهد.

در روح curl، ما تلاش می‌کنیم که همه‌ی جنبه‌های http2 را پوشش دهیم. curl معمولا به عنوان ابزاری برای تست وب‌سایت‌ها به کار می‌رود و ما تلاش می‌کنیم که این روند را برای http2 نیز حفظ کنیم.

curl از لایبرری جداگانه‌ای به نام nghttp2 برای لایه‌ی فریم استفاده می‌کند. curl به nghttp2 1.0 یا بالاتر نیاز دارد.

البته، فراموش نشود که در حال حاضر، curl ارائه‌شده در لینوکس و libcurl با پشتیبانی از پرتکل HTTP/2 به طور پیش‌فرض ارائه نمی‌شوند.

۱۱.۱. شباهت به HTTP 1.x

در درون، curl هدرهای http2 را به سبک HTTP 1.x تبدیل می‌کند و آن‌ها را به کاربر ارائه می‌دهد تا مانند HTTP کنونی ظاهر شوند. این کار اجازه می‌دهد تا انتقال داده‌ها برای کاربر curl و HTTP امروزی راحت‌تر شود. در درخواست‌های رو به بیرون نیز هدرها در میانه‌ی راه از حالت HTTP 1.x به فرمت http2 تبدیل می‌شوند. این قابلیت باعث می‌شود که کاربران خیلی به این مورد اهمیت ندهند که با کدام نسخه‌ی HTTP روبه‌رو هستند.

۱۱.۲. متن‌ساده، ناامن

curl از http2 برمبنای TCP استاندارد و هدر Upgrade: پشتیبانی می‌کند. اگر شما یک درخواست HTTP انجام دهید و بخواهید از HTTP 2 استفاده کنید، curl از سرور می‌خواهد که در صورت امکان از http2 استفاده کند.

۱۱.۳. TLS با لایبرری‌های مختلف

curl از لایبرری‌های مختلف TLS می‌تواند استفاده کند. چالشی که در TLS با آن مواجه هستیم، پشتیبانی از ALPN برای Http2 است و همچنین پشتیبانی از NPN است.

curl را با ورژن‌های جدیدتر OpenSSL یا NSS بیلد (Build) کنید تا پشتیبانی ALPN و NPN را داشته باشید. از GNUTLS یا PolarSSL، پشتیبانی ALPN را می‌گیرید، ولی NPN را نه.

۱۱.۴. استفاده در خط فرمان

برای اطلاع‌دادن به curl برای استفاده از http2، چه به صورت متن‌ساده یا TLS، از آپشن `http2--` استفاده کنید. curl همچنان از HTTP/1.1 به طور پیش‌فرض استفاده می‌کند.

۱۱.۵. آپشن‌های libcurl

۱۱.۵.۱. فعال‌کردن HTTP/2

اپلیکیشن شما می‌تواند از URLهای https:// یا http:// پشتیبانی کند، ولی شما می‌توانید از آپشن `CURLOPT_HTTP_VERSION` استفاده کنید تا نسخه‌ی HTTP مورداستفاده را تغییر دهید.

۱۱.۵.۲. Multiplexing

libcurl تلاش می‌کند که رفتارهای کنونی را ادامه دهد، بنابراین باید قابلیت multiplexing HTTP/2 را با آپشن `CURLMOPT_PIPELINING` فعال کنید.

نکته‌ی ریز دیگر این‌که در خاطر داشته باشید که اگر چندین درخواست با libcurl یک‌جا بفرستید، با اینترفیس چندگانه‌ی خودش، هر تعداد انتقالی را یک‌جا شروع کند و اگر شما می‌خواهید صبر کنید تا libcurl همه‌ی آن‌ها را در یک کانکشن قرار دهد،به جای این‌که چندین کانکشن با هم باز کند، شما می‌توانید از آپشن CURLOPT_PIPEWAIT استفاده کنید.

Server push .۱۱.۵.۳

libcurl 7.44.0 و بالاتر از قابلیت Server push در HTTP/2
پشتیبانی می‌کند. شما می‌توانید از این قابلیت با اضافه‌کردن یک تابع برای فراخوانی هنگام
دریافت Push با آپشن CURLMOPT_PUSHFUNCTION ست کنید. اگر Push
توسط اپلیکیشن پذیرفته شود، یک انتقال جدید روی curl ساخته می‌شود و محتوا به همان صورت کانکشن‌های معمولی تحویل داده می‌شود.

۱۲ بعد از http2

تصمیم‌های سخت بسیاری برای http2 گرفته شده است. با عرضه‌ی http2، یک راه صحیح برای به‌روزرسانی آن به نسخه‌های بالاتر پرتکل وجود دارد که راه برای آپدیت‌های بیشتر پرتکل هموار می‌کند. همچنین، یک مفهوم و زیرساخت را برای مدیریت چندین نسخه به طور همزمان ارائه می‌دهد. شاید لازم نباشد که همه‌ی چیزهای قدیمی را برای ارائه‌ی چیزهای جدیدتر دور بیندازیم؟

http2 بسیاری از امکانات قدیمی HTTP 1 را با خود به همراه دارد تا مبادله‌ی داده‌ها
بین HTTP1 و http2 میسر باشد. بعضی از این
امکانات قدیمی، مانع گسترش و توسعه‌های جدیدتر می‌شود. شاید http3
بتواند بعضی از این امکانات را پشتیبانی نکند؟

فکر می‌کنید هنوز چه چیزی در HTTP کم داریم؟

۱۲.۱ QUIC

پروژه‌ی QUIC (Quick UDP Internet Connections) یا کانکشن سریع اینترنتی (UDP)
یک پرتکل آزمایشی جالب است که توسط گوگل طراحی شده است که با همان سبک SPDY
اجرا شده است. QUIC ترکیبی از TLS + TCP + جایگزین HTTP/2 تحت UDP است.

QUIC اجازه می‌دهد که کانکشن‌ها با تأخیر بسیار کمتری برقرار شوند،
مشکل Packet Loss
را به گونه‌ای حل می‌کند که به‌جای متوقف‌شدن همه‌ی جریان‌ها فقط یک جریان قطع شود (همانطور
که http2
هم می‌کند) و همچنین امکان برقراری ارتباط را از طریق اینترفیس‌های مختلف شبکه فراهم می‌سازد و بنابراین
مشکل MPTCP را نیز حل می‌کند.

QUIC تا به الان، تنها در گوگل کروم و نیز
سرورهای آن‌ها پیاده‌سازی شده و استفاده از کد آن‌ها چندان راحت نیست، حتی اگر
یک libquic برای این‌کار داشته باشیم. این پرتکل به
عنوان یک پیش‌نویس به کارگروه انتقال داده‌های IETF آورده شده است.

۱۳ خواندن بیشتر

اگر فکر می‌کنید این نوشته، اطلاعات کافی درمورد موضوع ارائه نمی‌دهد، می‌توانید از لینک‌های زیر برای ارضای حس کنجکاو خود استفاده کنید:

لیست ایمیلی HTTPbis و آرشیو ایمیل‌های آن: https://lists.w3.org/Archives/Public/ietf-http-wg/

مشخصات http2 در قالب HTML: https://httpwg.github.io/specs/rfc7540.html

جزئیات نحومکار فایرفاکس با http2: https://wiki.mozilla.org/Networking/http2

جزئیات پیاده‌سازی http2 توسط curl: https://curl.haxx.se/docs/http2.html

وب‌سایت http2: https://http2.github.io/ و سؤالات متداول مربوط به پرتکل: https://http2.github.io/faq/

فصل HTTP/2 در کتاب "High Performance Browser Networking" ایلیا کریگورک:
https://hpbn.co/http2/

۱۴ تقدیر و تشکر

ایده‌ی استفاده از عکس‌های لگو از Mark Nottingham گرفته شده است.

آمارهای مربوط به HTTP از https://httparchive.org/ دریافت شده است.

نمودار RTT از ارائه‌های Mike Belshe گرفته شده است.

فرزندانم، آگنس و رکس که لگوهایشان را برای نشان‌دادن قسمت «انتخاب صف درست» به من قرض دادند.

همچنین از این دوستان که مرا با دیدگاه‌ها و بازخوردهایشان کمک کردند: Kjell Ericson, Bjorn Reese, Linus Swälas و Anthony Bryan. از کمک شما بسیار سپاسگزارم، چرا که این نوشته را واقعا بهتر کردند!

در طول تغییرات، این افراد صمیمانه خطاها را گزارش کردند و نوشته را بهتر کردند:
Mikael Olsson, Remi Gacogne, Benjamin Kircher, saivlis, florin-andrei-tp, Brett Anthoine, Nick Parlante, Matthew King, Nicolas Peels, Jon Forrest, sbrickey, Marcin Olak, Gary Rowe, Ben Frain, Mats Linander, Raul Siles, Alex Lee, Richard Moore

مترجم

از علی میرجمالی بابت حمایت‌های علمی و معنوی ایشان و همچنین همه‌ی کسانی که در فرآیند ترجمه‌ی این کتاب کمک کردند و به علت کثرت نام، نمی‌توانم همگی را نام ببریم، سپاسگزارم.

امیدوارم توانسته باشم به جامعه‌ی علمی فارسی‌زبان کمکی کرده باشم :)

واژه‌نامه

واژه‌ها و اصطلاحاتی که معادل فارسی ندارند یا ساختن معادل فارسی برای آن‌ها باعث ایجاد کج‌فهمی می‌شود.

HTTP pipelining

یک راه برای فرستادن یک درخواست دیگر، در حالی که منتظر پاسخ درخواست دیگری هستیم.

Head-of-line blocking

ترجمه‌ی مفهومی آن: «مسدودسازی توسط نفر اول یک صف». متأسفانه هیچ ترجمه‌ی دیگری برای این اصطلاح پیدا نکردم.

Host Name

به نام‌هایی که به هر میزبان اختصاص می‌دهند گفته می‌شود. مثلا `wikipedia.org`، `google.com`، `localhost` و `m.wikipedia.org` هر کدام یک Host name مستقل هستند.

Cookies

Cookies به داده‌های کوچکی گفته می‌شود که در مرورگرها به درخواست سرور ذخیره می‌شوند و در هر درخواست، همراه با اطلاعات دیگر به سرور فرستاده می‌شوند.

IETF

Internet Enginnering Task Force نیروی کار مهندسی اینترنت

TLS

Transport Layer Security یکی از پروتکل‌های رمزنگاری است و برای تأمین
امنیت ارتباطات از طریق اینترنت بنا شده‌است.

NPN

Next Protocol Negotiation ارتباط اولیه‌ی پرتکول بعدی

ALPN

Application Layer Protocol Negotiation ارتباط‌اولیه‌ی لایه‌ی کاربری پرتکول