http2 explained

Daniel Stenberg

Tabela de conteúdos

http2 explicado

Este é um documento detalhado descrevendo HTTP/2 (RFC 7540), Os antecedentes, conceitos, protocolo e algo sobre implementações existentes e o que o futuro pode trazer.

Veja https://daniel.haxx.se/http2/ para a casa canônica deste projeto.

Veja https://github.com/bagder/http2-explained para o código fonte de todos os conteúdos do livro.

CONTRIBUIR

Eu incentivo e acolho a ajuda e contribuições de todos os que possam oferecer melhorias. Nós aceitamos pull requests, mas também pode apenas abrir issues ou enviar um email para daniel-http2@haxx.se com as suas sugestões!

/ Daniel Stenberg

1. Historial

Este documento descreve o http2 de uma perspectiva técnica e ao nível do protocolo. Começou como uma apresentação que Daniel fez em Estocolmo em Abril 2014 que foi subsequentemente convertido e alargado para um documento com todos os detalhes e explicações adequadas.

RFC 7540 é o nome oficial da especificação final do HTTP2 e que foi publicado em 15 de maio de 2015: https://www.rfc-editor.org/rfc/rfc7540.txt

Todos e quaisquer erros neste documento são da minha autoria e os resultados da minha falha. Por favor indique-nos para que possam ser corrigidos numa versão actualizada.

Neste documento, eu tentei usar consistentemente a palavra "HTTP2" para descrever o protocolo em termos técnicos, o nome correcto é HTTP/2 Eu fiz essa escolha por uma questão de legibilidade e de obter um melhor fluxo na Língua.

1.1 Autor

O meu nome é Daniel Stenberg e trabalho para a Mozilla. Tenho trabalhado com open source e redes há mais de vinte anos para numerosos projetos. Possivelmente sou conhecido por ser o principal desenvolvedor do curl e libcurl. Fui envolvido no grupo IETF HTTPbis durante vários anos e aí pude estar a par com o trabalho do HTTP 1.1 assim bem como no envolvimento do processo de estandardização do http2

Email: daniel@haxx.se

Twitter: @bagder

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2 Ajuda!

Se encontrar problemas, omissões, erros ou mentiras descaradas neste documento, por favor envie-me uma versão corrigida do parágrafo e eu farei versões corrigidas. Eu darei os merecidos créditos a todas as pessoas que ajudarem! Eu espero fazer este documento melhor ao longo do tempo.

Este documento está disponível em https://daniel.haxx.se/http2

1.3 Licença

1.4 História do documento

A primeira versão deste documento foi publicada em 25 de abril de 2014. Aqui segue as maiores alterações nas versões mais recentes do documento.

Versão 1.13

- Convertido a versão mestre deste documento para sintaxe Markdown
- 13: Menção de mais recursos, links atualizados e descrições
- 12: Atualizada a descrição QUIC com referência para a elaboração
- 8.5: Actualizado com números actuais
- 3.4: A média é de agora 40 conexões TCP
- 6.4: Atualizado para refletir o que a especificação diz

Versão 1.12

- 1.1: HTTP / 2 está agora em um RFC oficial
- 6.5.1: Link para o RFC HPACK
- 9.1: Menção para alterar a configuração do Firefox 36+ para http2
- 12.1: Adicionada uma seção sobre QUIC

Versão 1.11

- Muitos melhoramentos de língua, a maioria indicados pelos amigos contribuidores
- 8.3.1: Menção de ações especificas de nginx e Apache httpd

Versão 1.10

- 1: O protocolo foi "okayed"
- 4.1: Actualizada a redacção desde 2014 é o ano passado
- Frente: Adicionada imagem e chamada de “http2 explained”, link corrigido
- 1.4: Adicionado a seção Historial do documento
- Muitos erros ortográficos e gramaticais corrigidos
- 14: Adicionado obrigado aos repórteres de bugs
- 2.4: Melhores legendas para os gráficos de crescimento HTTP
- 6.3: Corrigida a ordem da carruagem na ordem to comboio multiplexado.
- 6.5.1: HPACK rascunho-12

Versão 1.9

- Actualizado para HTTP/2 rascunho-17 e HPACK rascunho-11
- Adicionada seção "10. http2 in Chromium" (== agora cumprimento de uma página)
- Muitas correcções ortográficas
- 30 implementações agora
- 8.5: Adicionado alguns números de uso atuais
- 8.3: Mencione o Internet Explorer também
- 8.3.1 Adicionado implementações "em falta"
- 8.4.3: Mencionado que TLS também aumenta a taxa de sucesso

2. HTTP hoje

HTTP 1.1 transformou-se em num protocolo usado para praticamente tudo na Internet. Grandes investimentos foram feitos em protocolos e infra-estrutura que se aproveitam dele. Isto é considerado na medida em que muitas vezes é mais fácil hoje fazer as coisas correm em cima de HTTP ao invés de construir algo próprio novo.

2.1 HTTP 1.1 é enorme

Quando se criou o HTTP e foi dado ao mundo, foi concebido como um protocolo muito mais simples e direto, mas o tempo provou o contrário. HTTP 1.0 no RFC 1945 é uma espicificação de 60 páginas publicada em 1996. O RFC 2616 que descreve o HTTP 1.1, foi publicado apenas três anos mais tarde em 1999 e cresceu significativamente para 176 páginas. Todavia quando trabalhámos dentro do IETF na actualização da especificação, foi repartida e convertida em seis documentos, com uma quantidade maior de páginas no total (resultando no RFC 7230 e a sua família). De qualquer modo, HTTP 1.1 é grande e incluí uma grande variedade de detalhes e sutilezas, sem ignorar grandes quantidades de importantes peças opcionais.

2.2 Um mundo de opções

A natureza do HTTP 1.1 de ter vários pequenos detalhes e opções disponíveis para expansão futura, permitiu um crescimento de um ecosistema de software em que quase nenhuma implementação implementa tudo - e não é realmente possivel dizer o que "tudo" é. Isto levou a uma situação onde as funcionalidades que foram inicialmente pouco usadas viram poucas implementações e aqueles que implementaram estas funcionalidades viram serem pouco utilizadas.

Depois mais tarde, causaram um problema de interoperabilidade quandos os clientes e servidores começaram a utilizar estas funcionalidades . HTTP Pipelining é um exemplo primário destas funcionalidades.

2.3 Uso inadequado do TCP

Tem sido díficil o HTTP 1.1 tirar partido completo de todo o poder e performance que o TCP oferece. Clientes HTTP e browsers têm que ser muito criativos para encontrar soluções que diminuam o tempo que uma página leva a carregar.

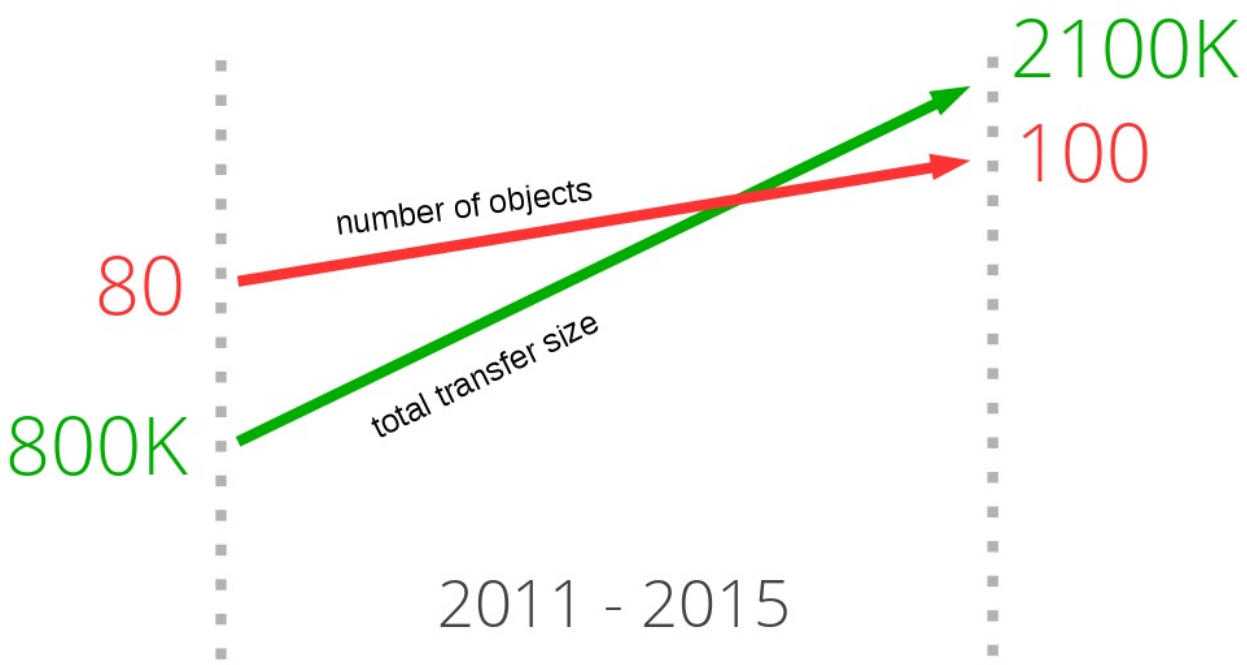
Outras tentativas que foram feitas em paralelo ao longo dos anos também vieram provar que o TCP não é assim tão facil de substituir e como tal continuamos a melhorar ambos o TCP e os protocolos no topo dele.

Posto de forma simples, TCP pode ser melhor utilizado para evitar pausas ou momentos no tempo que poderiam ter sido utilizados para enviar ou receber mais dados. A próxima seção vai realçar algumas dessas deficiências.

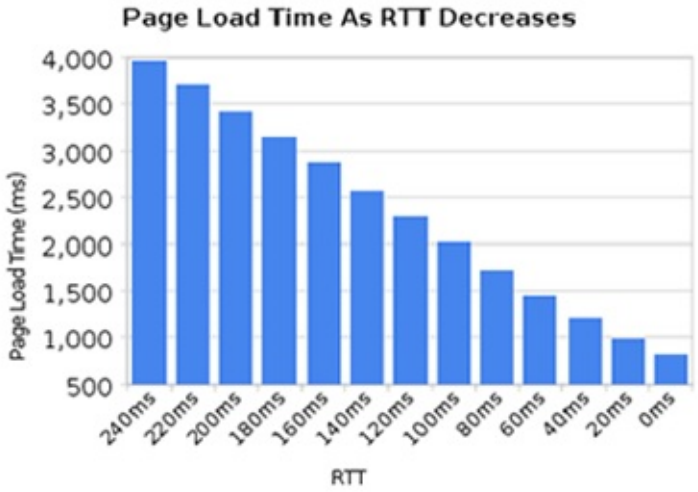
2.4 Tamanhos de transferência e número de objetos

Ao olhar para a tendência de alguns dos sites mais populares na web de hoje, e o tempo que leva a fazer download das suas páginas principais, um padrão claro emerge. Ao londo dos anos a quantidade de dados que é necessário descarregar tem vindo a aumentar acima de 1.9MB. O que é mais importante neste contexto é uma média de uma centena de recursos individuais que são necessários para exibir cada página.

Como o gráfico abaixo mostra, a tendência já se arrasta por um tempo e há pouca ou nenhuma indicação de que isso venha a mudar tão cedo. Ele mostra o crescimento do tamanho total de transferência (em verde) e o número total de solicitações usada em média (em vermelho) para servir os sites mais populares do mundo, e como eles mudaram ao longo dos últimos quatro anos.



2.5 Latência mata



HTTP 1.1 é muito sensível à latência, em parte porque HTTP Pipelining ainda é cheio de problemas o suficiente para permanecer desligada para uma grande porcentagem dos utilizadores.

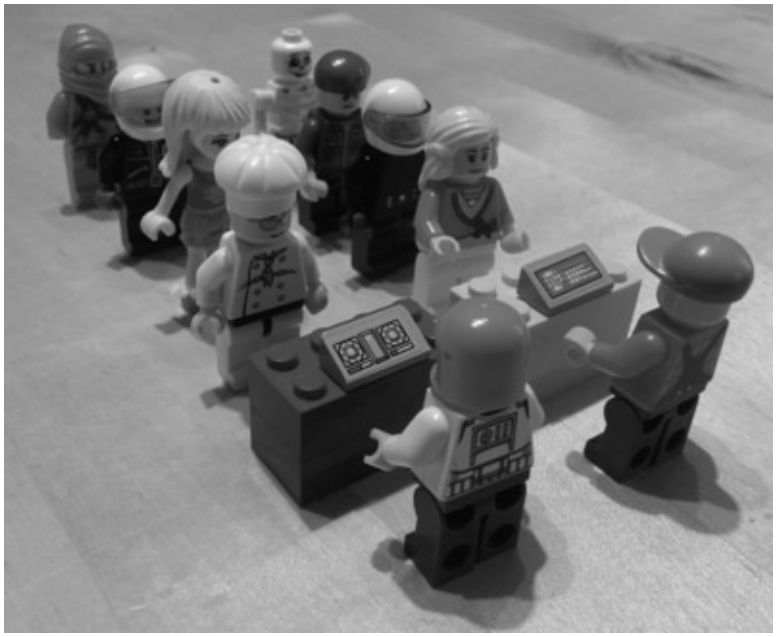
Enquanto nós vimos um grande aumento na largura de banda disponível para as pessoas ao longo dos últimos anos, não temos visto o mesmo nível de melhorias na redução da latência. Links de alta latência, como muitas das atuais tecnologias móveis, torna realmente difícil de conseguir uma boa e rápida experiência web mesmo se você tiver uma largura de banda muito alta.

Outro caso que realmente precisa de baixa latência é a de certos tipos de vídeo, como videoconferência, jogos e similares, onde não há apenas um fluxo pré-gerado para enviar para fora.

2.6. Bloqueio do primeiro da fila

HTTP Pipelining é uma forma de enviar um outro pedido enquanto aguarda a resposta a um pedido anterior. É muito semelhante a filas de um banco ou de um supermercado. Você só não sabe se a pessoa que está há sua frente é um cliente rápido ou um cliente irritante que vai demorar uma eternidade antes que ele / ela finalize: Bloqueio do primeiro da

fila.



Claro que você pode ter cuidado com a fila que escolhe, escolhendo aquela que você realmente acredita que é a correta, e às vezes você pode até mesmo iniciar uma nova fila por si mesmo, mas no final você não pode evitar de tomar uma decisão, e uma vez que é feita você não pode alternar filas.

A criação de uma nova fila está também associada com o desempenho de recursos, de modo que não é escalável para além de um menor número de filas. Simplesmente não há solução perfeita para isso.

Ainda hoje, em 2015, a maioria dos browsers de desktop fornecido com o HTTP pipelining vêem desativados por padrão.

Leitura adicional sobre este assunto podem ser encontrados por exemplo no Firefoxbugzilla entry 264354.

3. Estratégias para evitar a dor da latência

Como sempre quando se enfrentam problemas, as pessoas tentam encontrar soluções. Algumas das soluções são inteligentes e úteis, outras são apenas horríveis.

3.1 Spriting

Spriting é o termo corrente utilizado para descrever quando você põe várias imagens pequenas juntas numa só imagem grande. Depois utiliza javascript ou CSS para "cortar" pedaços dessa mesma imagem grande para mostrar imagens individuais pequenas.

Um site utilizaria este truque para velocidade. Descarregar uma imagem grande é muito mais rápido em HTTP 1.1 do que descarregar 100 imagens individuais pequenas.

Claro que isto tem as suas desvantagens para uma página de um site que apenas quer mostrar uma ou duas das imagens pequenas ou similar. Isto também faz com que as imagens sejam descartadas do cache ao mesmo tempo em vez de deixar permanecer as que são utilizadas com mais frequência.

3.2 Inlining

Inlining é outro truque para evitar enviar imagens individuais, e isto é feito utilizando data: URLs embutidos num ficheiro CSS. Isto tem benefícios similares e incovinientes como no caso de spriting.

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

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

3.3 Concatenação

Um site grande pode ter vários ficheiros diferentes de javascript. Ferramentas Front-end vão ajudar os programadores a junta-los num só ficheiro grande que o browser irá descarregar em vez de várias dezenas de ficheiros pequenos. Muitos dados serão enviados quando apenas poucos são necessários. Muitos dados terão que ser recarregados quando uma alteração é necessária.

Esta prática é claramente uma inconveniência para os programadores envolvidos.

3.4 Sharding

O último truque que irei mencionar é normalmente referido como “sharding”. Basicamente significa servir aspetos do seu serviços num numero máximo possível de servidores. De inicio isto parece estranho mas existe uma boa razão por detrás da mesma.

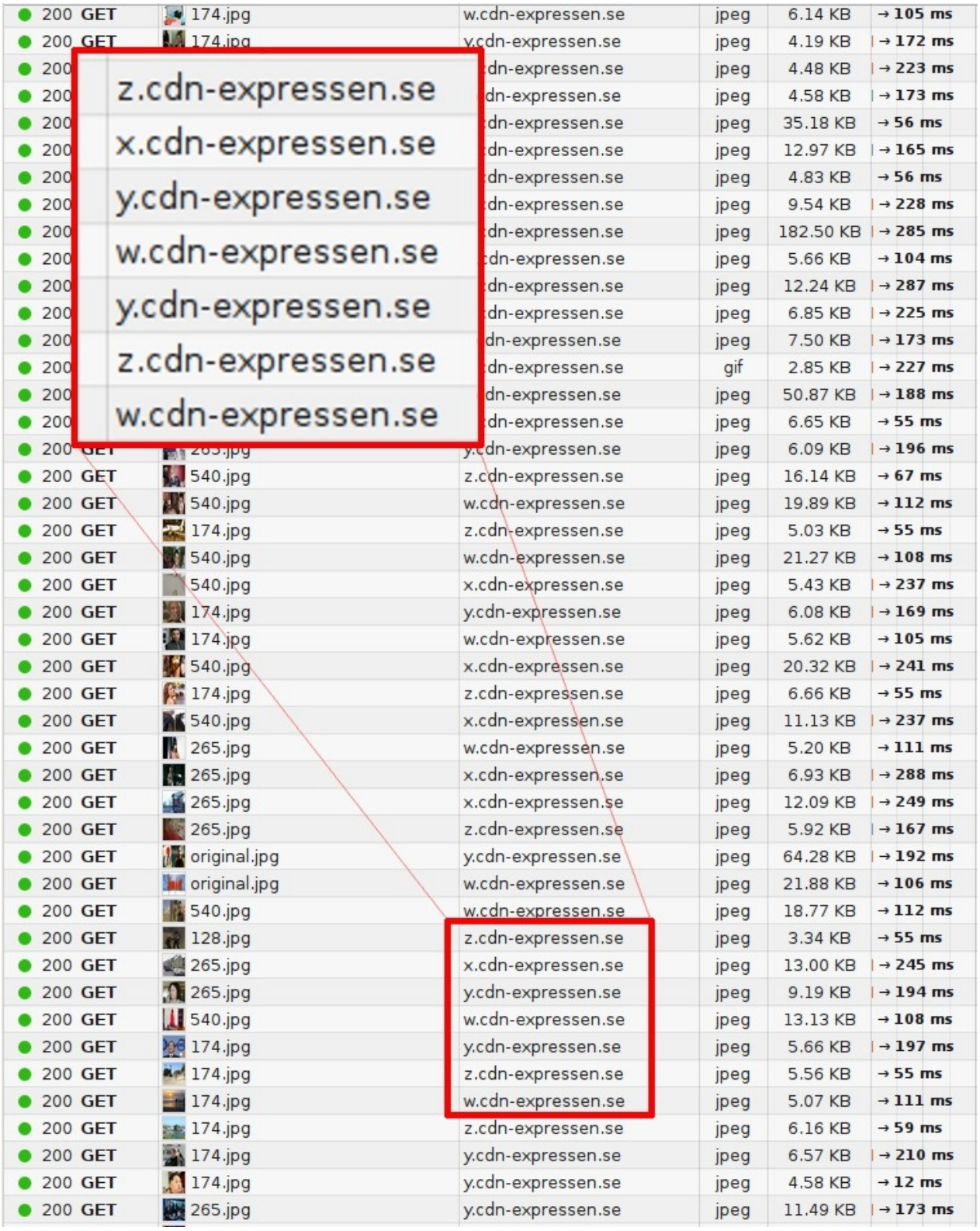
Inicialmente a especificação do HTTP 1.1 dita que o cliente poderia utilizar um máximo de duas conexões TCP para cada anfitrião. Então, como forma de não violar a especificação sites inteligentes inventaram novos host names e - voilá - pode ter mais conexões ao seu site e diminuir o tempo que leva a carregar.

Ao longo do tempo, essa limitação foi removida e hoje os clientes facilmente utilizam 6-8 conexões por anfitrião mas continuam a ter um limite e como tal os sites continuam a utilizar esta técnica para aumentar o numero de conexões. Como o número de objectos está sempre a aumentar - como eu já demonstrei - o número elevado de conexões é apenas usado para ter a certeza que o HTTP executa corretamente e o site é rápido. Isto não é raro que um site utilize 50 ou mesmo 100 ou mais conexões para apenas um site utilizando esta técnica.

Estatísticas recentes do httparchive.org mostram que os primeiros 300.000 URLs do mundo precisam em media de 40(!) conexões TCP para mostrar o site, e a expectativa é de crescer lentamente ao longo do tempo.

Outra razão é também de meter imagens ou recursos similares num anfitrião à parte que não utilize cookies, pelo que o tamanho das cookies nos dias correntes poder ser tanto ou quanto significante. Utilizando um anfitrião cookie-free pode por vezes aumentar a performance simplesmente permitindo pedidos de HTTP muito mais pequenos.

A imagem em baixo mostra como trace de packets é quando estamos a navegar um dos top sites da Suécia e como os pedidos estão distribuídos por vários anfitriões.



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

4. Actualizando HTTP

Não seria bom fazer melhorias no protocolo? Algo incluindo...

1. Fazer com que o protocolo seja menos sensível a RTT.
2. Corrigir o pipelining e o primeiro da fila.
3. Parar o desejo e necessidade e continuar a aumentar o numero de conexões para cada anfitrião
4. Manter todas as interfaces existentes, todo o conteúdo, e os formatos e esquemas de URI
5. Isso seria feito com o grupo de trabalho IETF's HTTPbis

4.1. IETF e o grupo de trabalho HTTPbis

A Internet Engineering Task Force (IETF) é uma organização que desenvolve e promove standards de internet. Essencialmente ao nível de protocolo. Eles são especialmente conhecidos pela série de documentação RFC documentando tudo desde das melhores práticas até TCP, DNS, FTP, HTTP e numerosas variantes de protocolo que nunca se desenvolveram.

Os grupos de trabalho dentro do IETF são formados com um âmbito limitado de trabalhar para um objectivo. Eles estabelecem um "carácter" com algumas guias e limitações que elas possam produzir. Qualquer pessoa e todo o mundo é permitido de se juntar às discussões e desenvolvimento. Alguém que se junte às discussões e diga algo tem as mesma possibilidade de afetar o resultado final e toda a gente conta como humanos e indivíduos, pouco conta para que empresas esses mesmo indivíduos trabalhem.

O grupo de trabalho HTTPbis (veja mais a baixo a explicação do nome) foi formada durante o verão de 2007 e foi encomendada a tarefa de actualizar a especificação do HTTP 1.1. Dentro deste grupo as discussões sobre a nova versão do HTTP só começaram realmente no final de 2012. O trabalho de actualização do HTTP 1.1 foi completado no inicio de 2014 e resultou nas séries de RFC 7320.

Na reunião operativa para o grupo de trabalho HTTPbis foi feita na cidade de Nova York no inicio de 2014. As discussões seguintes e procedimentos IETF foram realizadas para ter de fato o RFC oficial que se sucederam até ao ano seguinte.

Alguns dos agentes principais no ramo de HTTP não estiveram nos debates e reuniões do grupo de trabalho. Eu não quero mencionar nenhuma das empresas ou nome de produtos aqui, mas claramente que alguns agentes da internet hoje em dia estão confiantes que o IETF fará um bom trabalho sem que estas empresas estejam envolvidas...

4.1.1. A parte "bis" do nome

O grupo chama-se HTTPbis de onde a parte "bis" vem do advérbio de Latim dois . Bis é usado vulgarmente como sufixo ou parte do nome no IETF para uma actualização ou uma segunda revisão da especificação. Tal como no caso do HTTP 1.1.

4.2. http2 começou com o SPDY

SPDY é um protocolo encabeçado e desenvolvido pela Google. Eles certamente que o desenvolveram abertamente ao mundo e convidaram toda a gente a participar mas era obvio que iriam beneficiar se fosse controlado por ambas implementações nos browsers e uma população significante de servidores que sejam muito utilizados.

Quando o grupo HTTPbis decidiu que era tempo de começar a trabalhar no http2, o SPDY ja tinha demonstrado que era um conceito que funcionava. Mostrou que era possível implementar na Internet e havia números publicados que provavam a sua performance. O trabalho http2 começou subsequentemente depois do rascunho do SPDY/3 que era basicamente o rascunho http2 draft-00 com um pouco de mudanças.

5. Conceitos de http2

Afinal o que o http2 consegue fazer? Onde está o limite que o grupo HTTPbis se encarregou de fazer?

Na realidade foram restritos e mantiveram algumas restrições na capacidade de inovar dentro da equipa.

- Tem que manter paradigmas de HTTP. Ainda é um protocolo onde o cliente envia pedidos ao servidor através de TCP.
- http:// and https:// URLs não podem ser alterados. Não pode haver nenhum esquema novo para isto. A quantidade de conteúdo usado por URLs é demasiado grande para esperar uma mudança.
- Os servidores e clientes de HTTP1 vão existir algumas décadas, precisamos de fazer com que os proxies possam servir http2.
- Subsequentemente, proxies precisam de mapear funcionalidades a clientes HTTP 1.1 uma a uma.
- Remover ou reduzir partes opcionais do protocolo. Isto não foi realmente um requisito mas sim o mantra que veio do SPDY e da equipa da Google. Tendo a certeza que tudo é mandatório não existe uma maneira que se possa implementar tudo agora e no futuro não posa haver uma ratoeira.
- Nenhuma versão menor. Foi decidido que os clientes e servidores são compatíveis com o http2 ou não são. Se existir a necessidade de estender o protocolo ou modificar algo, então o http3 nascerá. Não existem mais versões pequenas no http2.

5.1. http2 para esquemas existentes URI

Como já foi mencionado os esquemas URI existentes não podem ser modificados, o http2 tem que ser criado usando os existentes. Uma vez que não são usados hoje em dia no HTTP 1.x, precisamos de deixar obviamente uma maneira de poder fazer upgrade para o http2 ou pedir ao servidor para usar http2 em vez de protocolos anteriores.

HTTP 1.1 define a maneira como isto se faz, especialmente o cabeçalho Upgrade:, que permite o servidor enviar de volta uma resposta utilizando um protocolo novo quando recebe esse pedido usando um protocolo antigo. Com um custo de um round-trip.

Essa penalização de roud-trip não era algo que a equipa de SPDY pudesse aceitar, e uma vez que eles apenas implementaram SPDY sobre TLS eles desenvolveram a nova extensão de TLS que é utilizada como atalho para uma negociação muito significativamente. Usando esta extensão, chamada NPN for Next Protocol Negotiation, o servidor diz ao cliente quais os protocolos que conhece e o cliente pode então escolher o protocolo que prefere.

5.2. http2 para https://

Muito do foco do http2 foi de fazer com que este tenha um comportamento correcto sobre TLS. SPDY apenas funciona sobre TLS e existia muita pressão para que no http2 fosse também mandatório mas não se chegou a um consenso e o http2 foi lançado com o TLS como opcional. No entanto, dois implementadores mais destacados disseram claramente que iram implementar apenas o http2 sobre TLS: A iniciativa Mozilla Firefox e a iniciativa Google Chrome. Dois dos principais browsers hoje em dia.

As razões para escolher apenas-TLS incluí respeito pela privacidade do utilizador e medições precoces mostraram que os novos protocolos têm maior taxa de sucesso quando utilizam TLS. Isto deve-se à suposição que qualquer tráfico que vá através da porta 80 é HTTP 1.1 faz com que qualquer dispositivo que intercepte o tráfico possa interferir ou destruir o mesmo quando se trata de um protocolo distinto de HTTP 1.1.

O assunto de o TLS ser mandatário tem causado muito movimento e muitas vozes agitadas nas listas de correio e reuniões - é bom ou mau? É um tema infectado - tenha em conta isto se perguntares alguma coisa a um membro da HTTPbis!

De maneira similar, há um debate feroz e durador sobre se o http2 deve impor a lista de cifras que devem ser mandatórias quando se usa TLS, ou se talvez se deve-se bloquear algumas ou se não deveria ser um requisito de todo pela camada do TLS mas deixe isso para o grupo de trabalho do TLS. A especificação determinou que o TLS deveria pelo menos usar a versão 1.2 e há algumas restrições de cifras.

5.3. negociação http2 sobre TLS

Next Protocol Negotiation (NPN), é um protocolo usado para negociar SPDY com servidores TLS. Como não se tratava de um Standard, foi levado à IETF e daí surgiu a ALPN: Application Layer Protocol Negotiation. ALPN é o que tem sido promovido a ser usado com http2, enquanto os clientes de SPDY ainda usam a NPN.

O facto que a NPN existe antes da ALPN levou algum tempo até a sua estandardização e levou a muitas implementações antecedentes de clientes e servidores http2. Também, a NPN é usada para muitos servidores que usam tanto SPDY como http2, suportar tanto NPN como ALPN nesses servidores faz todo o sentido.

A principal diferença entre ALPN e NPN é quem decide o protocolo a usar. com ALPN os clientes dizem ao servidor a lista de protocolos e a sua ordem de preferencia e o servidor escolhe a que quer, enquanto no NPN o cliente faz a escolha final.

5.4. http2 para http://

Como já foi mencionado anteriormente, para HTTP 1.1 em pleno texto a forma de negociar http2 é de pedir ao servidor com um cabeçalho Upgrade:. Se o servidor falar http2 irá responder com um estado "101 Switiching" e a partir daí comunica em http2 nessa conexão. Você nota que isto é uma forma de upgrade que tem o custo de um round-trip completo, mas a vantagem é que a conexão pode ser mantida e reutilizada de maneira mais generalizada do que as conexões HTTP1

Enquanto alguns representantes de browsers declararam que não iriam implementar este modo de falar http2, a equipa do Internet Explorer comunicou que irá, assim como o curl que já suporta este modo.

6. O protocolo http2

Suficiente sobre os bastidores, a história e política por trás do que nos trouxe até aqui. Vamos mergulhar nos detalhes do protocolo: os bits e os conceitos que constroem o http2.

6.1. Binário

http2 é um protocolo binário.

Apenas reflita sobre isso por um minuto. Se você tem se envolvido com protocolos da Internet antes, as chances são de que você começará a reagir instintivamente contra essa escolha, empacotando seus argumentos que dizem como protocolos baseados em texto/ascii são superiores porque humanos conseguem manusear requisições via telnet e assim por diante...

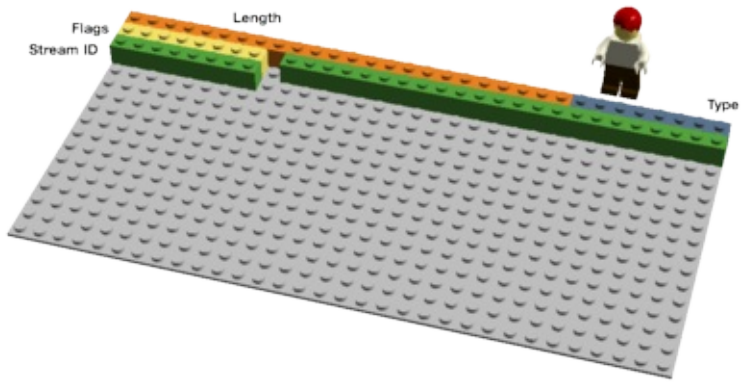
http2 é binário para tornar a construção muito mais fácil. Descobrir o início e o fim dos pacotes é uma das coisas mais complicadas no HTTP 1.1 e, na verdade, em protocolos baseados em texto em geral. Eliminando espaços em branco opcionais e diferentes formas de escrever a mesma coisa, a implementação se torna mais simples.

Além disso, se torna muito mais fácil separar as partes atuais do procolo da sua elaboração - o que é muito confuso e misturado no HTTP1.

O fato do protocolo oferecer compressão e que muitas vezes será executado sobre TLS diminui o valor do texto, pois você não verá texto trafegando de qualquer maneira. Nós simplesmente temos que nos acostumar com a ideia de utilizar uma ferramenta como o Wireshark para descobrir exatamente o que está acontecendo no nível do protocolo http2.

A depuração deste protocolo , provavelmente, será feita utilizando ferramentas como curl ou analisando o fluxo da rede com o *dissector* de http2 do Wireshark ou algo similar.

6.2. O formato binário



http2 envia quadros binários. Existem diferentes tipos de quadro que podem ser enviados e todos têm a mesma configuração: *Length* (comprimento), *Type* (tipo), *Flags* (configurações), *Stream Identifier* (identificador de fluxo) e *frame payload* (quadro de informação).

Existem 10 diferentes tipos de quadro definidos na especificação do http2 e talvez os dois tipos fundamentais que mapeiam para características do HTTP 1.1 são *DATA* e *HEADERS*. Eu vou descrever alguns dos quadros com mais detalhes adiante.

6.3. Fluxos multiplexados

O identificador de fluxo (*Stream Identifier*) mencionado na seção anterior associa cada quadro enviar via http2 com um "fluxo". Um fluxo é uma sequência de quadros independentes e bi-direcionais trocadas entre o cliente e o servidor utilizando uma conexão http2.

Um única conexão http2 pode conter vários fluxos concorrentemente abertos, com cada extremidade entrelaçando múltiplos fluxos. Fluxos podem ser estabelecidos e utilizados unilateralmente ou compartilhados pelo cliente ou servidor e eles podem ser encerrados por qualquer extremidade. A ordem em que os quadros são enviados dentro de um fluxo é importamente. Destinatários processam quadros na ordem em que eles são recebidos.

Multiplexar o fluxo significa que os pacotes de vários fluxos são misturados sobre a mesma conexão. Dois (ou mais) trens de dados são transformados em um e depois são divididos novamente do outro lado. Aqui estão os dois trens:

Os dois trens multiplexados na mesma conexão:

6.4. Prioridades e dependências

Cada fluxo tem uma prioridade (também conhecido como "peso") que é utilizada para dizer ao par (*peer*) quais são os fluxos mais importantes, no caso de existirem restrições de recursos que forçam o servidor a selecionar quais fluxos enviar primeiro.

Utilizando o quadro PRIORITY, um cliente também pode dizer ao servidor qual outro fluxo este fluxo atual depende. Ele permite que o cliente construa uma árvore de prioridades onde vários "fluxos filho" podem depender da conclusão de "fluxos pai".

A prioridade dos pesos e as dependências podem ser modificadas dinamicamente em tempo de execução, o que pode permitir que os navegadores especifiquem, numa página cheia de imagens e o usuário utiliza a barra de rolagem, quais são as imagens mais importantes para serem carregadas, ou se você alterna as abas ele pode priorizar um novo grupo de fluxos que, de repente, entram em foco.

6.5. Compressão do cabeçalho

HTTP é um protocolo sem estado (*stateless*).

HTTP is a stateless protocol. Em suma, isso significa que cada requisição necessita enviar ao servidor todos os detalhes necessários para atender essa solicitação, sem que o servidor necessite armazenar uma grande quantidade de informações e metadados de requisições anteriores. Como o http2 não muda esse paradigma, ele tem que funcionar da mesma forma.

Isso torna o HTTP repetitivo. Quando um cliente solicita muitos recursos do mesmo servidor, como imagens de uma página *web*, haverá uma série de requisições e todas parecerão idênticas. Uma séria de coisas quase idênticas implora por compressão.

Enquanto o número de objetos por página *web* tem aumentado (como mencionado anteriormente), o uso de cookies e o tamanho das requisições também tem aumentado ao longo do tempo. Cookies também precisam ser incluídos em todas as requisições, muitas vezes os mesmos em várias requisições.

As requisições HTTP 1.1 tem realmente crescido tanto de tamanho que às vezes elas acabam ficando maior que a janela TCP inicial, o que as torna muito lentas para enviar, pois elas necessitam de um *round-trip* completo para obter um ACK de volta do servidor antes que a requisição completa seja enviada. Esse é um outro argumento para compressão.

6.5.1. Compressão é um assunto complicado

A compressão de HTTPS e SPDY foi descoberta vulnerável aos ataques BREACH e CRIME. Inserindo um texto conhecido no fluxo e descobrindo como isso altera a saída, um invasor pode descobrir o que está sendo enviado em uma carga criptografada.

Realizando compressão em conteúdo dinâmico para um protocolo - sem se tornar vulnerável a um destes ataques - exige um pouco de reflexão e análise cuidadosa. Isto é o que o time HTTPbis tentou fazer.

Entra o HPACK, compressão de cabeçalho para HTTP/2, que - como o nome sugere - é um formato de compressão especialmente concebido para cabeçalhos http2, e está sendo especificado em um projeto separado. O novo formato, juntamente com outras medidas (como alguns pedem a intermediários que não comprimam cabeçalhos específicos e preenchimento opcional de quadros), deve torná-lo mais difícil de explorar a compressão.

Nas palavras de Roberto Peon (um dos criadores do HPACK):

> "HPACK" foi projetado para tornar o vazamento de informações mais difícil em uma implementação de acordo, realizar a codificação e decodificação mais rápida e barata, fornecer para o receptor controle sobre a compressão do tamanho do contexto, permitir a reindexação por parte do proxy (ou seja, estado compartilhado) entre *frontend* e *backend* dentro de um proxy), e comparações rápidas de *strings* utilizando a codificação de Huffman".

6.6. Reset - mude de ideia

Uma das desvantagens do HTTP 1.1 é que, quando uma mensagem é enviada com o cabeçalho *Content-Length* com um certo tamanho, não é possível cancelá-lo facilmente. Você pode (mas nem sempre) cancelar a conexão TCP, mas isso vem com o custo de negociar um *handshake* TCP novo.

Uma solução melhor seria apenas parar a mensagem e enviar uma nova. Isto pode ser feito com o quadro RST_STREAM do http2 que ajudará a previnir desperdícios e a necessidade de cancelar conexões.

6.7. Server push

Esta funcionalidade também é conhecida como "cache push". A ideia é que, se o cliente solicita o recurso X, o servidor pode determinar que o cliente também solicitará o recurso Z e enviará sem que o cliente o solicite. Isso ajudará o cliente colocando o recurso Z em cache de forma que ele estará lá quando for solicitado.

O envio a partir do servidor (*server push*) é algo que o cliente deve explicitamente permitir que o servidor faça. Mesmo assim, o cliente pode rapidamente encerrar o fluxo a qualquer momento com RST_STREAM não permitindo assim nenhum recurso em particular.

6.8. Controle de fluxo

Cada fluxo http2 individual tem sua própria janela de fluxo anunciada, onde a outra extremidade pode enviar informações. Se você conhece como SSH funciona, isto é muito semelhante em estilo e espírito.

Para cada fluxo (*stream*), ambas as extremidades tem que informar ao seu par que possuem espaço suficiente para lidar com os dados de entrada, e a outra ponta só é permitida enviar a quantidade informada até que a janela se estenda. Só existe controle de fluxo para quadros (*frames*) DATA.

7. Extensões

O protocolo http2 obriga que o receptor leia e ignore todos os quadros desconhecidos (*unknown frame type*). Duas partes podem negociar o uso de novos tipos de quadros em uma forma *hop-by-hop*, mas esses quadros não podem mudar o estado e eles não terão controle de fluxo.

Houve uma ampla discussão durante a fase de desenvolvimento do protocolo http2 para decidir se ele deveria ou não permitir extensões, com opiniões a favor e contra. Depois do draft-12, o pêndulo oscilou pela última vez e as extensões foram finalmente autorizadas.

Extensões não fazem parte do atual protocolo, mas serão documentadas a parte do núcleo da especificação. Já existem dois tipos de quadro (*frame*) que estão em discussão para serem incluídos no protocolo e que, provavelmente, serão os primeiros quadros a serem enviados como extensões.

7.1. Serviços alternativos

Com a adoção do http2, existem razões para suspeitar que as conexões TCP serão mais demoradas e serão mantidas vivas por muito mais tempo do que nas conexões HTTP 1.x. Um cliente deve ser capaz de fazer o que quiser com uma única conexão para cada página/*host*, e essa conexão pode, potencialmente, ficar ativa por um longo tempo.

Isso afetará o trabalho de balanceamento de carga e podem surgir situações onde uma página queira sugerir ao cliente se conectar a outro servidor. Isto poderia acontecer por questões de desempenho (performance) ou se a página está em manutenção, etc.

O servidor enviará o cabeçalho Alt-Svc: header (ou o quadro ALTSVC com http2) dizendo ao cliente sobre um serviço alternativo: outra rota para o mesmo conteúdo, utilizando outro serviço, servidor e porta.

Um cliente tentará se conectar ao serviço assincronamente e somente utilizar esta alternativa se a nova conexão for realizada com sucesso.

7.1.1. TLS oportunista

O cabeçalho Alt-Svc permite que o servidor que provê conteúdo por meio de http:// informar ao cliente que o mesmo conteúdo também está disponível através de uma conexão TLS.

Esta é uma característica um tanto discutível. Uma conexão deste tipo realizaria uma conexão TLS sem autenticar e não seria advertida como "segura" em nenhum lugar, não usaria nenhum cadeado na interface gráfica e, na verdade, não há como dizer ao usuário que não é o velho HTTP, mas é um TLS oportunista e algumas pessoas estão firmes contra este conceito.

7.2. Bloqueado

Um quadro (*frame*) deste tipo é utilizado uma única vez por um agente http2 quando ele tem dados para serem enviados, mas o controle de fluxo proíbe que seja enviada qualquer informação. A ideia é que se a implementação receber este quadro (*frame*), você sabe que algo está errado na sua implementação e/ou você está recebendo menos dados que a sua velocidade permite.

Uma citação do draft-12, antes deste quadro ser retirado para se tornar uma extensão:

> "O quadro BLOCKED está incluído nesta revisão para facilitar a experimentação. Se os resultados desta experiência não prover um resultado positivo, ele poderá ser removido"

8. Um mundo http2

O que acontecerá quando http2 for adotado? Ele será adotado?

8.1. Como o http2 afetará os humanos normais?

http2 ainda não é amplamente implantado nem utilizado. Não podemos dizer com certeza exatamente como as coisas acontecerão. Vimos como o SPDY é utilizado e podemos fazer suposições e cálculos baseados nesse e em experimentos anteriores.

http2 reduz o número de "round-trips" de rede necessários e, com a multiplexação e o descarte rápido de fluxos indesejados, evita o problema de bloqueio de primeira fila.

O protocolo permite um grande número de fluxos (*streams*) paralelos que vão além dos sites atuais que utilizam a técnica de domínio fragmentado (*sharding*).

Com a funcionalidade de prioridades utilizada adequadamente nos fluxos, as chances dos clientes receberem os dados mais importantes antes dos menos importantes são muito maiores. Tudo isso em conjunto, eu diria que temos boas chances de páginas sendo carregadas mais rapidamente e mais responsivas. Em poucas palavras: uma melhor experiência na *web*.

Creio que ainda não podemos dizer o quão mais rápido e quantas melhorias nós veremos. Em primeiro lugar, a tecnologia ainda é muito nova e ainda não temos visto clientes e servidores com implementações que realmente aproveitam todo o poder que o protocolo oferece.

8.2. Como o http2 afetará o desenvolvimento *web*?

Ao longo do anos, os desenvolvedores *web* e os ambientes de desenvolvimento *web* têm reunido uma caixa de ferramentas cheia de dicas para contornar os problemas com o HTTP 1.1, lembre-se que eu citei algumas delas no começo desse documento como justificativas para o http2.

Muitos desses contornos, que são atualmente são utilizados por padrão e sem pensar, irão provalmente prejudicar a performance do http2 ou, pelo menos, não tirar vantagem dos novos super poderes do novo protocolo. *Sprinting* e *inlining* não devem ser utilizados com http2. *Sharding* será provavelmente prejudicial ao http2, pois utilizará menos conexões.

Um problema é que, certamente, os *sites* e os desenvolvedores precisarão desenvolver para o mundo que, a curto prazo, funcionará com clientes HTTP1.1 e http2, e será um desafio ter uma melhor performance para todos os usuários sem disponibilizar duas versões diferentes de *front-end*.

Por estas razões, eu suspeito que haverá algum tempo antes de vermos todo o potencial do http2 sendo atingido.

8.3. Implementaçẽos http2

Tentar documentar implementações específicas em documentos como este é certamente fútil e condenado ao fracasso, pois ficará desatualizado em um curto espaço de tempo. Em vez disso, eu vou explicar a situação em termos mais amplos e indicar os leitores a lista de implementações no *site* do http2.

Havia uma grande quantidade de implementações já no início e essa quantidade tem aumentado ao longo do trabalho no http2. No momento dessa escrita, existem mais de 40 implementações listadas e a maioria implementa a versão final.

8.3.1 Navegadores

Firefox é o navegado que tem encabeçado as implementações *drafts*. Twitter tem ficado a altura e ofereceu seus serviços sobre http2. Google começou a oferecer suporte http2 em Abril de 2014 em alguns servidores de testes executando seus serviços e, desde Maio de 2014, eles oferecem suporte http2 nas versões de desenvolvimento do Chrome. Microsoft apresentou um suporte para http2 na próxima versão do Internet Explorer. Safari (com iOS 9 e Mac OS X El Capitan) e Opera disseram que vão suportar http2.

8.3.2 Servidores

Já existem vários servidores implementando http2.

O popular servidor Nginx oferece suporte http2 desde a versão 1.9.5 lançada em 22 de Setembro de 2015 (onde ele sobrescreve o módulo SPDY, ou seja, não é possível executar os dois módulos na mesma instância do servidor).

O servidor httpd Apache tem um módulo http2 mod_http2 desde a versão 2.4.17 que foi lançada em 9 de Outubro de 2015.

H2O, Apache Traffic Server, nghttp2, Caddy e LiteSpeed lançaram suas versões compatíveis com http2.

8.3.3 Outros

curl e libcurl suportam conexões http2 inseguras, assim como a versão baseada em TLS utilizando alguma de muitas bibliotecas TLS.

Wireshark suporta http2. A ferramenta perfeita para análise do tráfego de rede em http2.

8.4. Críticas comuns ao http2

Durante o desenvolvimento desse protocolo, houve muito debate e, certamente, há um certo número de pessoas que acreditam que esse protocolo acabou completamente errado. Eu quero mencionar alguns dos mais comuns e os argumentos contra eles:

8.4.1. “O protocolo é desenhado e feito pelo Google”

Existem variações indicando que o mundo ficaria ainda mais dependente e controlado pelo Google. Isso não é verdade. O protocolo foi desenvolvido no âmbito da IETF da mesma maneira que os protocolos são desenvolvidos por mais de 30 anos. No entanto, todos nós reconhecemos e admitimos o trabalho impressionante que o Google fez com o SPDY que não só provou ser possível implantar um novo protocolo dessa forma, mas que também forneceu números ilustrando quais ganhos poderiam ser obtidos.

Google anunciou publicamente que irão remover o suporte para SPDY e NPN no Chrome em 2016 e eles insistem que os servidores migrem para HTTP/2.

8.4.2. “O protocolo é útil somente para os navegadores”

Isto é meio que verdade. Uma das principais razões por trás do desenvolvimento do http2 é a correção o HTTP *pipelining*. Se o seu caso originalmente não tem necessidade de *pipelining*, então http2 não trará muitos ganhos para você. Certamente não é a única melhoria no protocolo, mas uma das maiores.

Assim que os serviços começarem a perceber o poder e a capacidade da multiplexação de fluxos que uma única conexão traz, eu suspeito que veremos mais uso do http2.

Pequenas APIs REST e usos simples do HTTP 1.x podem não encontrar muitos ganhos no que o http2 tem a oferecer. Mas, também, deve haver poucas desvantagens no uso do http2 para a maioria dos usuários.

8.4.3. “O protocolo é útil somente para *sites* grandes”

De forma alguma. A capacidade de multiplexação certamente ajudará a melhorar a experiência em conexões de alta latência que *sites* pequenos possuem sem distribuições geográficas.

8.4.4. “O uso de TLS o torna mais lento”

Isto pode ser verdade, de certa forma. O “handshake” TLS adiciona um pequeno tempo extra, mas existem esforços em andamento para reduzir o número de viagens (“round-trips”) necessárias. A sobrecarga para fazer TLS por meio da conexão, ao invés de texto puro, não é insignificante e claramente consome mais CPU e energia será gasta da mesma forma que um tráfego não seguro. O quanto e o que isso impactará está sujeito a opiniões e medições. Veja, por exemplo, istlsfastyet.com para mais informações.

Telecomunicações e outros operadores de rede, por exemplo o ATIS Open Web Alliance, dizem que eles precisam de tráfego não criptografado para oferecer cachê, compressão e outras técnicas necessárias para prover uma rápida experiência *web* via satélite, para aviões e similares. http2 não torna o uso de TLS obrigatório, portanto não devemos ter problemas com os termos.

Muitos usuários da Internet expressaram uma preferência por utilizar TLS de forma mais ampla e nós devemos ajudar a proteger a privacidade dos usuários.

As experiências também mostraram que, utilizando TLS, há uma chance maior de sucesso do que ao implementar novos protocolos de texto puro na porta 80, pois há muitas caixas no meio do caminho que inteferem, já que na porta 80 é comum imaginar o uso para HTTP.

Finalmente, graças a multiplexação dos fluxos http2 sobre uma única conexão, casos normais de uso do navegador podem diminuir substancialmente o número de *handshakes* TLS e melhorar a performance em comparação com o HTTPS utilizando HTTP 1.1.

8.4.5. “Não ser ASCII é um fator decisivo”

Sim, nós gostamos de poder ver os protocolos claramente, pois é mais fácil depurar e rastrear. Mas protocolos baseados em texto são mais propensos ao erro e abertos aos erros de transformação e interpretação.

Se você realmente não suporta um protocolo binário, então você também não poderia lidar com TLS e compressão no HTTP 1.x e essa funcionalidade é utilizada por muito tempo.

8.4.6. “Não é mais rápido do que HTTP/1.1”

Naturalmente é um tema sujeito a debate e discussões sobre como medir o que é mais rápido, mas, nos cenários do SPDY, muitos testes foram realizados e o tempo de carga de uma página foi mais rápido (por exemplo "How Speedy is SPDY?" pela University of Washington e "Evaluating the Performance of SPDY-enabled Web Servers" por Hervé Servy) e tais experiências também foram feitas com o http2. Estou ansioso para ver mais testes e experiências sendo publicadas. Um primeiro teste básico feito pelo httpwatch.com pode indicar que o HTTP/2 cumpre o que promete.

8.4.7. “Não respeita camadas”

Sério, esse é o seu argumento? Camadas não são pilares intocáveis e sagrados de uma religião mundial e, se nós cruzamos algumas áreas cinzentas durante a criação do http2, foi no interesse de fazer um protocolo bom e efetivo dentro das restrições inidicadas.

8.4.8. “Não corrige várias deficiências do HTTP/1.1”

É verdade. Com o objetivo específico de manutenção dos paradigmas do HTTP/1.1, houve vários recursos antigos do HTTP que tiveram que permanecer. Tais como os cabeçalhos comuns que também incluem os temidos *cookies*, cabeçalhos de autorização, entre outros. Mas como contraponto à manutenção desses paradigmas, temos um protocolo onde é possível ser implantado sem uma certa obrigação de substituir ou reescrever uma quantidade inconcebível de trabalho. http2 é basicamente uma nova camada de "framming".

8.5. http2 será amplamente utilizado?

Ainda é muito cedo para dizer com certeza, mas eu posso adivinhar e estimar e é isso que eu vou fazer aqui.

Os pessimistas dirão “veja o bem que o IPv6 fez”, como exemplo de um novo protocolo que demorou décadas para começar a ser amplamente utilizado. http2 não é um IPv6. Este é um protocolo no topo do TCP utilizando os mecanismos de atualização do HTTP, números de porta, TLS etc. Não exigirá nenhuma mudança de roteadores ou *firewalls*.

Google provou para o mundo com o trabalho no SPDY que um novo protocolo pode ser implantado e utilizado por navegadores e serviços com múltiplas implementações em um curto espaço de tempo. Enquanto o percentual de servidores na Internet que oferecem atualmente está na faixa de 1%, a quantidade de dados que estes servidores lidam é muito maior. Alguns dos mais populares *web sites* oferecem SPDY.

http2, baseado nos mesmos paradigmas do SPDY, eu diria que tem mais chances de ser mais utilizado uma vez que é um protocolo IETF. A implementação do SPDY sempre foi retido pelo estigma de “ser um protocolo Google”.

Existem vários grandes navegadores por trás do lançamento. Representantes do Firefox, Chrome, Safari, Internet Explorer e Opera disseram que irão entregar navegadores com suporte ao http2 e mostraram trabalhos em desenvolvimento.

Existem vários operadores de servidores que estão dispostos a oferecer suporte ao http2 em breve, incluindo Google, Twitter e Facebook e nós esperamos ver esse suporte sendo adicionado em implementações de servidores populares como Apache HTTP Server e nginx. H2o é um novo servidor HTTP incrivelmente rápido com suporte ao http2 e que trás muito potencial.

Alguns dos maiores fornecedores de *proxy*, incluindo HAProxy, Squid and Varnish anunciaram suas intenções para apoiar http2.

Durante todo o ano de 2015, a quantidade de tráfego utilizando http2 tem aumentado. No início de Setembro, o uso no Firefox 40 foi de 13% de tráfego HTTP e 27% de tráfego HTTPS, enquanto o Google está recebendo aproximadamente 18% de requisições HTTP/2. Deve-se notar que o Google executa outras novas experiências com novos protocolos (veja QUIC em 12.1) o que faz com que o nível de uso do http2 seja menor do que seria.

9. http2 e Firefox

Firefox vem acompanhando as alterações de perto e tem proporcionado implementações http2 de testes por muitos meses. Durante o desenvolvimento do protocolo http2, clientes e servidores têm que concordar sobre qual versão do rascunho (*draft*) do protocolo implementar, o que dificulta um pouco a execução dos testes. Fique atento para que o cliente e servidor implementem a mesma versão de rascunho do protocolo.

9.1. Em primeiro lugar, verifique se está habilitado

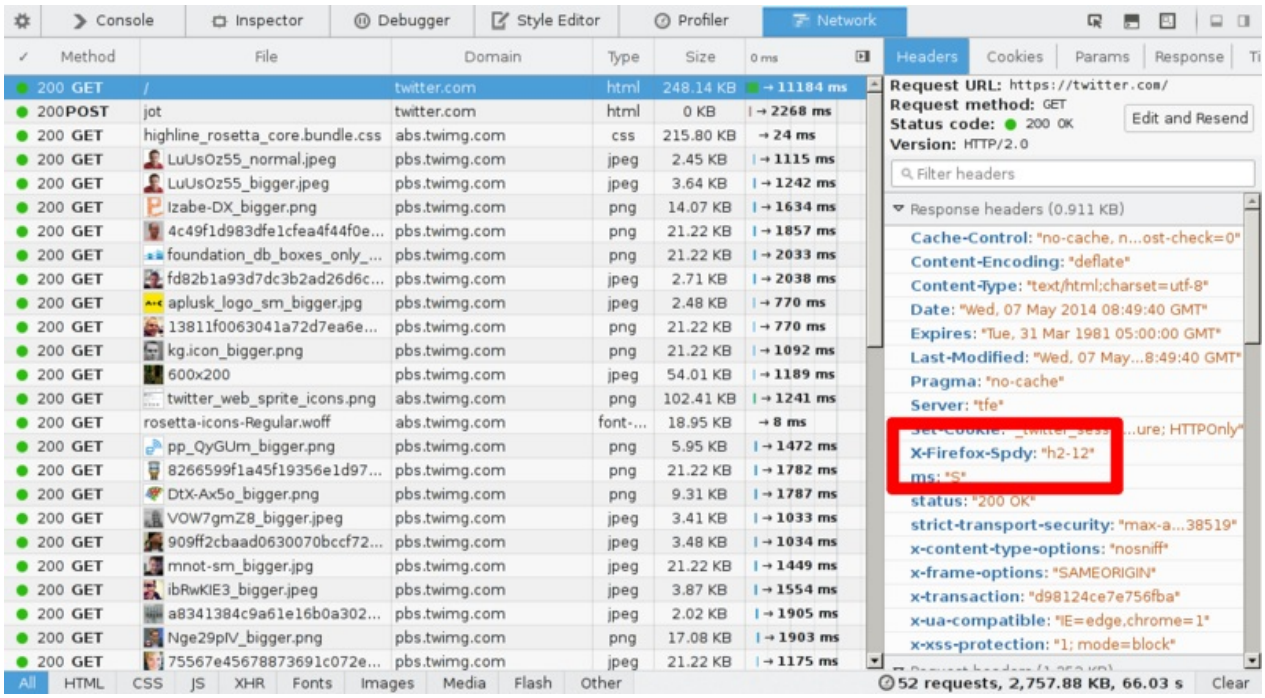
Em todas as versões do Firefox desde a versão 35, lançada em 13 de Janeiro de 2015, o suporte a http2 está habilitado por padrão.

Entre na seção 'about:config' na barra de endereços e procure pela opção “network.http.spdy.enabled.http2draft”. Tenha certeza que está definida para *true*. Firefox 36 adicionou outra configuração chamada “network.http.spdy.enabled.http2” que é definida como *true* por padrão. Esta última opção controla o http2 versão "simples" (*"plain"*), enquanto que a primeira opção habilita e desabilita a negociação de versões rascunho do http2. Ambas são definidas como *true* desde o Firefox 36.

9.2. Somente TLS

Lembre-se que o Firefox somente implementa http2 sobre TLS. Você somente verá http2 em ação com Firefox quando navegar em *sites* https:// e que ofereçam suporte http2.

9.3. Transparente!



Não existe nenhum elemento de interface que indique que você está "falando" http2. Não é possível dizer facilmente. Uma forma de descobrir é habilitando o modo "Web developer -> Network" e verificar os cabeçalhos de resposta recebidos do servidor. A resposta é “HTTP/2.0” e o Firefox adiciona seu próprio cabeçalho chamado “X-Firefox-Spdy:”, como mostra a imagem acima.

Os cabeçalhos que você vê na aba de rede ao utilizar http2 foram convertidos a partir do formato binário do http2, para o estilo clássico de cabeçalhos do HTTP 1.x.

9.4. Visualizar o uso do http2

Existem plugins do Firefox disponíveis que ajudam a visualizar se um *site* está utilizando http2. Um deles é o "HTTP/2 and SPDY Indicator".

10. http2 e Chromium

O time do Chromium implementa o http2 e provê suporte para ele nos canais *dev* e *beta* por bastante tempo. Começando pelo Chrome 40, lançando em 27 de Janeiro de 2015, http2 está habilitado por padrão para um certo número de usuários. O número começou realmente pequeno e então foi incrementado gradualmente com o passar do tempo.

O suporte SPDY será eventualmente removido. Em um artigo no blog, o projeto anunciou em Fevereiro de 2015:

> "Chrome suporta SPDY desde o Chrome 6, mas a maior parte dos benefícios estão presentes no HTTP/2, é hora de dizer tchau. Nós planejamos remover o suporte do SPDY no começo de 2016"

10.1. Em primeiro lugar, verifique se está habilitado

Digite "chrome://flags/#enable-spdy4" na barra de endereços do seu navegador Chrome e clique em "enable" se ele ainda não está habilitado.

10.2. Somente TLS

Lembre-se que o Chrome implementa http2 somente sobre TLS. Você somente verá http2 em ação com Chrome quando navegar em *sites* https:// e que ofereçam suporte http2.

10.3. Visualizar o uso do http2

Existem plugins do Chrome disponíveis que ajudam a visualizar se um *site* está utilizando HTTP/2. Um deles é o "HTTP/2 and SPDY Indicator".

10.4. QUIC

Experiências atuais do Chrome com QUIC (veja seção 12.1) diluem os números HTTP/2 de alguma maneira.

11. http2 e curl

O projeto curl provê suporte experimental para http2 desde Setembro de 2013.

No espírito do curl, nós pretendemos suportar cada detalhe do http2 que seja possível. curl é frequentemente utilizado como uma ferramenta de teste e uma maneira de “pingar” manualmente *sites* e nós pretendemos manter isto para http2 também.

curl utiliza uma biblioteca separada nghttp2 para a funcionalidade de camada de frame do http2. curl requer nghttp2 1.0 ou mais novo.

Note que, atualmente, no linux, o curl e libcurl nem sempre são instalados com o suporte ao protocolo HTTP/2 ativado.

11.1. Parecido com HTTP 1.x

Internamente, curl converterá cabeçalhos http2 de entrada para o formato de cabeçalhos HTTP 1.x e fornecê-los ao usuário, de forma semelhante ao HTTP existente. Isto permite uma transição mais fácil para quem está utilizando curl e HTTP hoje em dia. Da mesma forma, curl converterá cabeçalhos de saída no mesmo estilo: informe cabeçalhos HTTP 1.x para o curl e eles serão convertidos em tempo real quando estão conversando com servidores http2. Isto permite que os usuários não tenham que se preocupar demais com cada particularidade da versão HTTP em uso para cada conexão.

11.2. Texto puro, inseguro

curl suporta http2 sobre o padrão TCP via cabeçalho “Upgrade:”. Se realizar uma requisição HTTP e perguntar por HTTP 2, curl perguntará ao servidor se é possível atualizar a conexão para http2.

11.3. TLS e quais bibliotecas

curl suporta uma grande variedade de diferentes bibliotecas TLS para sua implementação TLS, e isso continua válido para o suporte http2. O desafio com TLS para o mundo http2 é o suporte para APLN e, de certa forma, o suporte NPN.

Compile o curl utilizando versões atuais do OpenSSL ou NSS para obter suporte para ALPN e NPN. Utilizando GnuTLS ou PolarSSL você terá suporte para ALPN, mas não para NPN.

11.4. Uso na linha de comando

Para dizer ao curl para utilizar http2, seja texto puro ou sobre TLS, deve ser utilizada a opção `--http2` (que é “traço traço http2”). O padrão no curl ainda é HTTP/1.1, portanto a opção extra é necessária para indicar o uso de http2.

11.5. Opções libcurl

11.5.1 Habilitar HTTP/2

Sua aplicação deve utilizar URLs https:// ou http:// normalmente, mas setar a opção `CURLOPT_HTTP_VERSION` do curl_easy_setopt para `CURL_HTTP_VERSION_2` para fazer uma tentativa do libcurl utilizar http2. Ele tentará conectar via http2 se for possível, mas continuará utilizando HTTP 1.1 caso ocorra algum problema.

11.5.2 Multiplexação

Como libcurl tenta manter o comportamento existente o máximo possível, será necessário habilitar multiplexação HTTP/2 para sua aplicação com a opção CURLMOPT_PIPELINING. Caso contrário, continuará utilizando uma requisição de cada vez por conexão.

Outro pequeno detalhe para ter em mente é que se várias requisições forem solicitadas de uma só vez com libcurl, utilizando sua interface múltipla, uma aplicação pode iniciar qualquer número de transferências de uma vez. Caso seja necessário que o libcurl espere um pouco para adicioná-las na mesma conexão, ao invés de abrir novas conexões para todas elas, a opção CURLOPT_PIPEWAIT pode ser utilizada para cada transferência que você prefira esperar.

11.5.3 Server push

libcurl 7.44.0 e posteriores suportam HTTP/2 server push. Para utilizar esta funcionalidade, indique um callback na opção CURLMOPT_PUSHFUNCTION. Se a aplicação aceitar o *push*, uma nova transferência será criada utilizando “CURL easy handle” e o conteúdo será entregue nele, assim como qualquer outra transferência.

12. Após o http2

Muitas decisões difíceis e compromissos foram tomados no http2. Com a implantação do http2, existe uma forma estabelecida para atualizar para outras versões de protocolo que garante a base desse funcionamento em futuras revisões do protocolo. Ele também traz a noção e infraestrutura que podem lidar com múltiplas versões diferentes em paralelo. Talvez nós não precisamos eliminar totalmente algo antigo quando lançarmos um novo?

http2 ainda possui muito “legado” que o HTTP 1 trouxe por causa do desejo de manter possível o *proxy* de ida e volta entre versões HTTP 1 e http2. Alguns desses legados dificultam o avanço no desenvolvimento e evoluções. Talvez o http3 pode eliminar alguns deles?

O que você acha que ainda está faltando no http?

12.1. QUIC

O protocolo QUIC (Quick UDP Internet Connections) feito pelo Google é um experimento interessante, realizado no estilo e espírito que eles fizeram com o SPDY. QUIC é um substituto implementado em UDP para TCP + TLS + HTTP/2.

QUIC permite a criação de conexões com muito menos latência, resolve a perda de pacotes bloqueando apenas os fluxos individuais, ao invés de todos eles como faz o HTTP/2, e permite que novas conexões sejam feitas em diferentes interfaces de rede facilmente - também cobrindo áreas que o MPTCP pretende resolver.

QUIC é, por enquanto, implementado somente pelo Google no Chrome e em seus servidores e esse código não é facilmente reutilizado em outro lugar, mesmo que haja um esforço nesse sentido, como a libquic. O protocolo está em draft para o grupo de trabalho de transporte IETF.

13. Outras leituras

Se você acha que esse documento é um pouco superficial em conteúdo e detalhes técnicos, aqui estão recursos adicionar que podem ajudá-lo a satisfazer sua curiosidade:

- A lista de e-mails HTTPbis e seus arquivos: https://lists.w3.org/Archives/Public/ietf-http-wg/
- A especificação http2 em uma versão HTML: https://httpwg.github.io/specs/rfc7540.html
- Detalhes de rede do http2 no Firefox: https://wiki.mozilla.org/Networking/http2
- Detalhes de implementação do http2 no projeto curl: https://curl.haxx.se/docs/http2.html
- O *site* http2: https://http2.github.io/ e, talvez, o FAQ: https://http2.github.io/faq/
- O capítulo sobre HTTP/2 do livro “High Performance Browser Networking” escrito por Ilya Grigorik: https://hpbn.co/http2/

14. Agradecimentos

Inspiração e imagem de pacote no formato de Lego de Mark Nottingham.

Dados de tendência HTTP vêm do https://httparchive.org/.

O gráfico RTT vem de apresentações feitas por Mike Belshe.

Meus filhos Agnes e Rex por me emprestarem suas peças de Lego para a imagem "head of line".

Obrigado aos seguintes amigos por comentários e feedback: Kjell Ericson, Bjorn Reese, Linus Swälas e Anthony Bryan. Sua ajuda é muito apreciada e realmente melhorou o documento!

Durante várias iterações, as seguintes pessoas amigáveis reportaram problemas e melhorias ao documento: 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