http2 explained

Daniel Stenberg

Table des matières

http2 expliqué

Ceci est un document détaillé décrivant HTTP/2 (RFC 7540), les prémices, concepts, protocole, les implémentations existantes et ce que le futur pourrait nous réserver.

Référez-vous à https://daniel.haxx.se/http2/ pour tout ce qui concerne ce projet.

Référez-vous à https://github.com/bagder/http2-explained pour le code source de ce livre.

CONTRIBUTIONS

J'encourage et apprécie l'aide et les contributions de chaque personne qui désire apporter des améliorations. Nous acceptons les pull requests, mais vous pouvez également simplement créer un ticket ou encore envoyer un courriel à daniel-http2@haxx.se avec vos suggestions!

/ Daniel Stenberg

1. Background

Ce document décrit http2 d'un point de vue technique et protocolaire. Il a commencé par une présentation à Stockholm réalisée par Daniel en avril 2014, présentation qui a été par la suite convertie et étoffée dans un document complet avec des explications détaillées.

La RFC 7540 est le nom officiel de la spécification http2 finale, elle a été publiée le 15 mai 2015 : https://www.rfc-editor.org/rfc/rfc7540.txt

Toute erreur dans ce document est mienne et le résultat de mes approximations. Merci de me les indiquer afin que je les corrige pour les prochaines versions.

Dans ce document, j'ai essayé d'utiliser le terme "http2" pour décrire le nouveau protocole en termes purement techniques, le nom officiel est HTTP/2. J'ai fait ce choix pour améliorer la fluidité de lecture et obtenir une meilleure lisibilité.

1.1 Auteur

Mon nom est Daniel Stenberg et je travaille chez Mozilla. J'ai travaillé dans l'open source et la réseautique, depuis plus de 20 ans, sur de nombreux projets. Je suis plus connu peut-être en tant que développeur principal de curl et libcurl. J'ai été impliqué dans le groupe de travail de l'IETF HTTPbis pendant plusieurs années où j'ai maintenu à jour les spécifications HTTP 1.1 et participé à la standardisation de http2.

Email: daniel@haxx.se

Twitter: @bagder

Web: daniel.haxx.se

Blog: daniel.haxx.se/blog

1.2 Aide!

Si vous trouvez des erreurs, oublis, fautes ou mensonges éhontés dans ce document, je vous prierais de bien vouloir m'envoyer une version corrigée que je publierai dans une édition révisée. Je mentionnerai clairement les noms des contributeurs! J'espère améliorer ce document avec le temps.

Ce document est disponible ici: https://daniel.haxx.se/http2

1.3 License

1.4 Historique du document

La première édition de ce document fut publiée le 25 avril 2014. Voici les changements majeurs des dernières éditions.

Version 1.13 :

- Conversion de la version principale de ce document au format Markdown
- 13: Mention de plus de ressources, mise à jour des liens et descriptions
- 12: Mise à jour de la description de QUIC en mentionnant le draft
- 8.5: Mise à jour des chiffres
- 3.4: La moyenne est maintenant de 40 connexions TCP
- 6.4: Mise à jour en s'alignant sur la spécification

Version 1.12 :

- 1.1: HTTP/2 est maintenant une RFC officielle
- 6.5.1: Lien vers la RFC HPACK
- 9.2: Mention du changement de config pour http2 dans Firefox 36 et +
- 12.1: Nouvelle section sur QUIC

Version 1.11 :

- Nombreuses améliorations de style linguistique (Note du traducteur: en anglais)
- 8.3.1: Mention des développements spécifiques nginx et Apache httpd

Version 1.10 :

- 1: Le protocole est "presque approuvé"
- 4.1: Rafraîchissement du style linguistique (Note traducteur: en anglais)
- Converture: ajout de l'image et légende "http2 explained", lien corrigé
- 1.4: Ajout de la section "Historique"
- Diverses corrections orthographiques
- 14: Ajout de remerciements pour les contributeurs
- 2.3: Meilleurs libellés pour le graphique de croissance HTTP
- 6.3: Correction de l'ordre des wagons dans le train multiplexé
- 6.5.1: HPACK draft-12

Version 1.9

- Mise à jour HTTP/2 draft-17 et HPACK draft-11
- Ajout de la section "10. http2 et Chromium"
- Diverses corrections orthographiques
- Désormais 30 implémentations
- 8.5: Ajout des chiffres d'utilisation
- 8.3: Mention d'Internet Explorer
- 8.3.1: Ajout des "implémentations manquantes"
- 8.4.3: Mention que TLS améliore le taux de réussite

2. HTTP aujourd'hui

Le protocole HTTP 1.1 est omniprésent sur Internet. De nombreux investissements ont été réalisés sur des protocoles et infrastructures tirant profit de celui-ci. À tel point que lors de l'implémentation d'un nouveau projet, il est souvent plus facile d'utiliser HTTP plutôt que de développer un nouveau protocole.

2.1 HTTP 1.1 est très dense

Lors de la création de HTTP, il fut probablement perçu comme un protocole plutôt simple et évident, ce qui, avec le temps, s'est révélé faux. HTTP 1.0 spécifié dans la RFC 1945 est une spécification de 60 pages datant de 1996. La RFC 2616 qui décrit HTTP 1.1 a été publiée trois ans plus tard en 1999 et comprend 176 pages. Puis, quand nous, à l'IETF, avons mis à jour cette spécification, elle a été répartie en six documents, avec davantage de pages au total (RFC 7230 et associées). Tout bien considéré, HTTP 1.1 est dense et comporte une multitude de détails, subtilités et, non dans une moindre mesure, de très nombreux points optionnels.

2.2 Une flopée d'options

De par sa nature, HTTP 1.1 possède plusieurs options et petits détails permettant l'ajout d'extensions ultérieures. Ce qui a mené à un écosystème logiciel où aucune implémentation n'a tout couvert (encore faut-il définir ce que "tout" représente). Cela a mené à un cercle vicieux où les fonctionnalités peu utilisées ont été peu implémentées ce qui en limitait l'utilité pour ceux qui en faisaient usage.

Plus tard, cela a causé des problèmes d'interopérabilité entre clients et serveurs qui utilisaient certaines de ces fonctionnalités. Le pipelining HTTP en est un exemple parlant.

2.3 Usage incorrect de TCP

HTTP 1.1 a du mal à vraiment tirer parti de la puissance et des performances offertes par TCP. Les clients et navigateurs HTTP doivent être très créatifs pour trouver des solutions qui abaissent les temps de chargement de pages web.

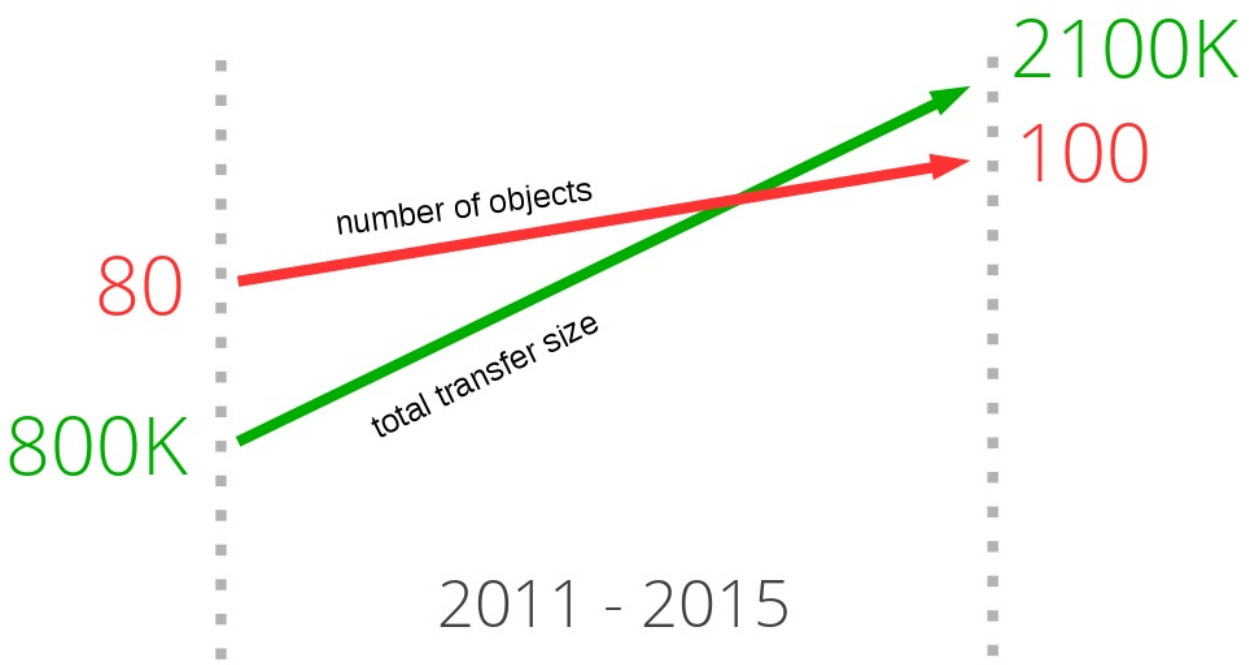
D'autres expérimentations menées en parallèle à travers les années ont confirmé qu'il est difficile de remplacer TCP et qu'il fallait donc améliorer à la fois TCP et les protocoles au-dessus.

En clair, TCP peut être utilisé à meilleur escient en évitant les pauses ou en utilisant certains moments pour envoyer et recevoir des données. Les chapitres suivants développeront ces points.

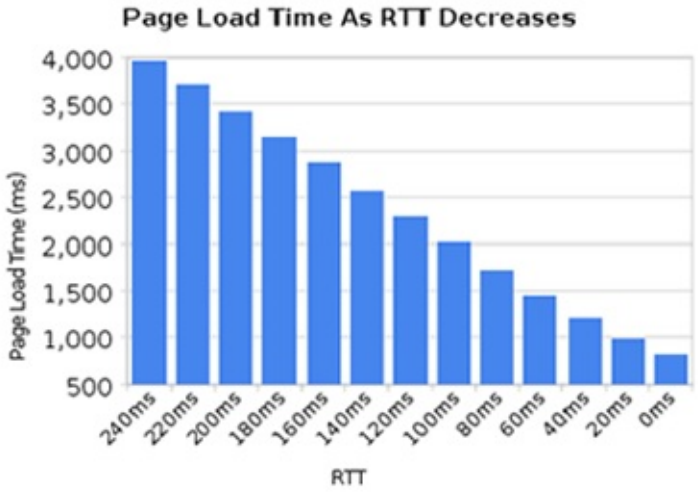
2.4 Volume de données et nombre d'objets

En regardant la tendance parmi les sites les plus importants sur le web aujourd'hui et ce que cela implique pour télécharger leurs pages d'accueil, une tendance se dégage. Au fil des années, le nombre de données à transférer a augmenté régulièrement pour atteindre et même surpasser 1.9Mo. Plus important encore, le nombre moyen de ressources distinctes (ou objets) va au-delà de la centaine pour afficher chaque page.

Le graphique ci-dessous montre que cette tendance date déjà et rien n'indique qu'elle s'inversera. On y constate l'évolution, sur les quatre dernières années, du volume de données (en vert) et du nombre moyen de requêtes (en rouge) pour le chargement des pages web les plus populaires au monde.



2.5 La latence tue



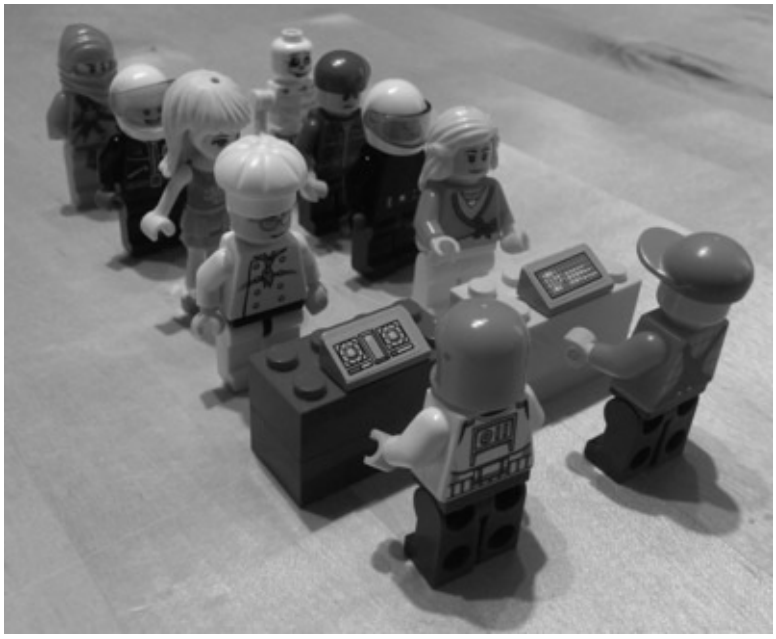
HTTP 1.1 est très sensible à la latence, en partie à cause du pipelining HTTP qui demeure problématique à un tel point qu'il est désactivé pour la plupart des utilisateurs.

Bien qu'on ait observé une augmentation de la bande passante depuis plusieurs années, la réduction de la latence n'a pas suivi la même évolution. Les liens à forte latence, comme les technologies mobiles, rendent l'expérience Web difficile même avec une connexion haut débit.

D'autres usages qui nécessitent une latence faible sont certains types de vidéos comme : la vidéo-conférence, le jeu en ligne ainsi que les flux vidéos générés en direct.

2.6. Blocage en tête de file (Head-of-line blocking)

Le pipelining HTTP est une manière d'envoyer une requête additionnelle sans attendre la réponse de la requête précédente. C'est semblable à la file d'attente d'une caisse à la banque ou au supermarché. Vous ne savez pas si le client vous précédant est rapide ou s'il s'agit d'une personne qui prendra son temps. Ce phénomène décrit parfaitement le : “head-of-line blocking”.



Bien sûr, vous pouvez faire attention et choisir la file que vous croyez être la meilleure, et parfois vous pouvez en commencer une nouvelle, mais vous devez prendre une décision que vous ne pourrez plus changer par la suite.

Créer une nouvelle file joue sur les performances et les ressources disponibles, ce qui ne permet pas de créer de nouvelles files à l'infini. Il n'y a pas de solution parfaite.

Même aujourd'hui, en 2015, le pipelining HTTP est désactivé par défaut sur la plupart des navigateurs destinés aux ordinateurs personnels.

Plus de détails sur ce sujet peuvent être consultés dans l'entrée 264354 de la base de données Bugzilla de Firefox.

3. Rustines pour s'accommoder de la latence

Comme toujours face aux problèmes, les gens trouvent des solutions de contournement. Certaines sont astucieuses et utiles, d'autres sont juste d'horribles rustines.

3.1 Spriting

Spriting est un terme anglais souvent utilisé pour décrire la consolidation de petites images en une seule grosse image. Cette image est ensuite découpée en petites images individuelles, via l'utilisation de JavaScript ou de CSS.

Cette astuce est utilisée car l'obtention d'une seule grosse image est beaucoup plus rapide en HTTP 1.1 que celle de 100 petites.

Bien sûr, cela représente une surcharge pour les pages qui n'ont besoin que d'une ou deux images de la mosaïque. Cela rend aussi le cache moins pertinent car on vide du cache toutes les images de la mosaïque en une fois au lieu de garder les images les plus utilisées dans le cache.

3.2 Inlining

L'inlining (en ligne, en français) est une autre astuce évitant l'envoi d'images individuellement. Il est possible d'imbriquer des données à l'intérieur des URLs présentes dans le CSS. Ce genre d'approche offre des avantages et inconvénients similaires au spriting.

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

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

3.3 Concaténation

Il est courant pour des sites de taille importante d'utiliser plusieurs fichiers JavaScript séparés. Les outils de conception de sites permettent aux développeurs de fusionner ces fichiers pour qu'un navigateur ne fasse qu'une seule requête vers un gros fichier JavaScript. L'inconvénient de cette méthode est qu'elle nécessite le chargement d'une quantité importante de

données là où seule une petite partie est réellement nécessaire. De la même façon, l'intégralité du fichier doit être téléchargée à nouveau si une infime partie est modifiée.

3.4 Sharding

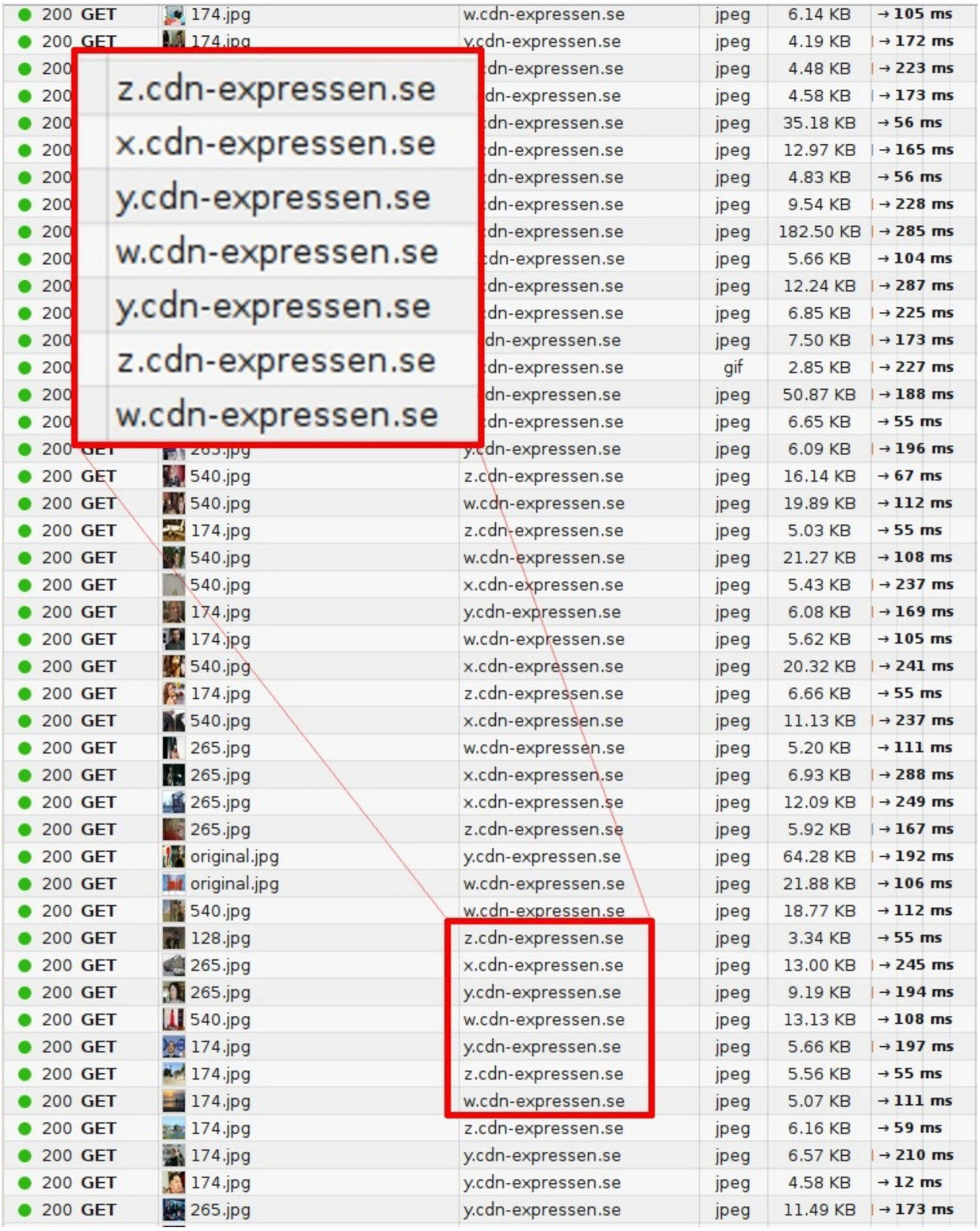
La dernière astuce que je veux mentionner est connue sous le nom de "sharding". C'est la possibilité de charger le contenu d'un site depuis autant de hosts que possible. À première vue, cela paraît étrange mais c'est finalement assez astucieux.

HTTP 1.1 limitait initialement à deux le nombre de connexions TCP simultanées d'un client à un même host. Pour ne pas contredire la spécification, des sites astucieux créaient simplement de nouveaux noms de hosts, et voilà, vous pouviez avoir davantage de connexions vers votre site et réduire le temps de chargement.

Avec le temps, cette limitation a été levée et les clients utilisent aujourd'hui typiquement 6 à 8 connexions par nom de host; cela dit, la limite perdure et certains sites continuent d'utiliser cette technique pour accroître le nombre de connexions. Comme le nombre d'objets augmente continuellement, l'utilisation d'un grand nombre de connexions permet de maximiser les performances. Il n'est pas inhabituel de voir des sites utiliser plus de 50 voire 100 connexions pour un seul site utilisant cette technique. Des statistiques récentes de httparchive.org montrent que le top 300.000 des URLs requiert en moyenne 40(!) connexions TCP pour afficher le site, et que ce nombre augmente de façon continue.

La taille des cookies devenant conséquente, il est également intéressant de placer certaines ressources comme les images sur un nom d'hôte distinct, n'utilisant pas de cookies. On augmente ainsi la performance en diminuant la taille des requêtes HTTP pour ces ressources.

L'image ci dessous montre une capture de trafic lors du chargement d'un site suédois connu et comment les requêtes sont réparties sur différents noms d'hôtes.


z.cdn-expressen.se
x.cdn-expressen.se
y.cdn-expressen.se
w.cdn-expressen.se
y.cdn-expressen.se
z.cdn-expressen.se
w.cdn-expressen.se

4. Mettre à jour HTTP

Ce serait sympa d'améliorer ce protocole, n'est-ce pas ? Notamment:

1. Un protocole moins sensible au RTT (Round Trip Time, Aller-Retours)
2. En corrigeant le pipelining et le head of line blocking
3. En évitant le besoin de multiplier le nombre de connexions vers un même hôte
4. En conservant toutes les interfaces existantes, tout le contenu et le format d'URI
5. Tout cela à travers le groupe de travail de l'IETF HTTPbis

4.1. IETF et le groupe de travail HTTPbis

L'IETF (Internet Engineering Task Force) est une organisation qui développe et promeut les standards de l'Internet, et ce essentiellement au niveau protocolaire. Très connue pour la série de RFC documentant tout depuis TCP, DNS, FTP, HTTP aux bonnes pratiques en passant par des variantes de protocoles qui n'ont jamais vu le jour.

Au sein de l'IETF existent des groupes de travail se penchant sur un sujet bien établi avec des objectifs précis. Ils établissent une charte qui définit les règles et limitations liées à l'objectif. Toute personne intéressée peut se joindre aux discussions et au développement, et chaque participant bénéficie du même droit de parole et du même poids que les autres. Peu d'importance est apportée aux sociétés pour lesquelles les participants travaillent (le cas échéant).

Le groupe de travail HTTPbis (voir plus tard pour l'origine du nom) a été créé à l'été 2007 et chargé de mettre à jour la spécification HTTP 1.1. Des discussions sur une nouvelle version de HTTP ont réellement commencé fin 2012. La mise à jour HTTP 1.1 s'est terminée début 2014 avec la série des RFC 7320.

La réunion finale d'interopérabilité pour le groupe de travail HTTPbis s'est tenue à New York en juin 2014. Toutefois les discussions en suspens et les procédures IETF nécessaires pour obtenir la RFC officielle continueront jusque l'année suivante.

Des acteurs importants de HTTP ont manqué à l'appel du groupe de travail pendant les discussions et réunions. Je ne souhaite pas mentionner de nom de société ou produit, mais il est clair que ces acteurs de l'Internet font confiance à l'IETF pour finir le travail sans leur participation...

4.1.1. Le "bis" dans HTTPbis

Le groupe est appelé HTTPbis, le "bis" faisant référence au latin "bis". Bis est souvent utilisé comme suffixe ou partie d'un nom à l'IETF lors d'une mise à jour ou addendum d'une spécification. Comme ici avec HTTP 1.1, donc.

4.2. http2 provient de SPDY

SPDY est un protocole développé par Google. Ils l'ont clairement créé dans un esprit d'ouverture et ont invité tout le monde à participer, mais il était évident qu'ils bénéficiaient d'une position de contrôle avec à la fois un navigateur populaire et un nombre significatif de serveurs très utilisés pour leurs services.

Quand le groupe HTTPbis décida de travailler sur http2, SPDY avait déjà prouvé que le concept fonctionnait. Il avait montré qu'il était possible de le déployer sur Internet et des mesures publiées montraient sa meilleure performance. Le travail sur http2 a donc démarré depuis le draft SPDY/3, rapidement transcrit dans un draft-00 de http2 grâce à quelques remplacements.

5. Concepts http2

Que réalise http2 ? Quelles sont les limites imposées par le groupe HTTPbis ?

Elles sont plutôt strictes avec des contraintes.

- il faut maintenir le principe de HTTP. C'est toujours un protocole où le client envoie une requête vers un serveur en TCP.
- les URLs http:// et https:// ne doivent pas changer. Il ne faut pas de nouvelle méthode. Le contenu utilisant ces URLs est trop important pour s'attendre à ce que tout le contenu change de méthode.
- clients et serveurs HTTP1 seront encore là pour des décennies, on doit pouvoir les faire basculer (“proxifier”) vers des serveurs http2.
- en conséquence, les proxys doivent mapper les fonctionnalités http2 à 1:1 vis-à-vis des clients HTTP 1.1.
- retirer ou réduire les composants optionnels du protocole. Ce n'est pas vraiment un pré-requis mais un principe venant de SPDY et des équipes Google. En s'assurant que tout est obligatoire, il n'y a pas de risques d'omettre d'implémenter quelque chose, puis de se retrouver piégé plus tard.
- pas de versions mineures. Il est décidé que clients et serveurs sont compatibles http2 ou pas du tout. S'il faut modifier ou étendre le protocole, alors http3 sera défini. Il n'y aura pas de versions mineures en http2.

5.1. http2 avec les méthodes URI existantes

Comme mentionné, les formats d'URL existants ne peuvent pas être modifiées, http2 doit donc gérer les URI actuelles. Comme elles sont déjà utilisées en HTTP 1.x, on doit pouvoir mettre à niveau (“upgrader”) le protocole vers http2 ou demander au serveur d'utiliser http2 plutôt que les anciens protocoles.

HTTP 1.1 a défini un moyen de le faire, l'en-tête Upgrade:, qui permet au serveur de renvoyer une réponse avec le nouveau protocole quand la requête était envoyée avec un protocole plus ancien. Au prix d'un aller-retour (round-trip).

Le coût de cet aller-retour n'est pas quelque chose que l'équipe SPDY allait tolérer. Comme ils n'implémentaient SPDY qu'au dessus de TLS, ils ont développé une extension TLS permettant de raccourcir la négociation de façon drastique. Cette extension, appelée NPN pour Next Protocol Negotiation, permet au serveur d'indiquer au client quels protocoles il connaît, celui-ci choisissant alors d'utiliser celui qu'il préfère.

5.2. http2 pour https://

Une attention particulière a été portée au bon fonctionnement du protocole http2 en conjonction avec TLS. SPDY lui-même ne fonctionnait qu'au dessus de TLS, et nombreux sont ceux qui souhaitaient que http2 ne soit utilisé qu'avec TLS. Ces derniers n'ont pas réussi à obtenir un consensus, et l'utilisation de TLS n'est qu'optionnelle. Néanmoins, les équipes de deux des plus importants navigateurs actuels, Mozilla Firefox et Google Chrome, ont clairement fait savoir qu'ils n'avaient pas l'intention d'implémenter http2 sans TLS.

Les motivations pour imposer TLS impliquent le respect de la vie privée et les mesures montrant que le nouveau protocole avait plus de chances de succès avec TLS. En effet, il est couramment répandu que tout ce qui passe sur le port TCP 80 est du HTTP 1.1 et pas mal d'intermédiaires réseau interfèrent ou cassent le trafic quand d'autres protocoles sont utilisés.

Le sujet TLS obligatoire a causé pas mal d'agitation dans les meetings et mailing-list, est-ce bien ou mal ? C'est une question typiquement empoisonnée, attention quand vous abordez ce sujet avec un membre du groupe!

De même, il y eut un long débat pour savoir si http2 devrait forcer une liste obligatoire d'algorithmes de chiffrement (ciphers en anglais) avec TLS, ou en bloquer certains ou encore laisser cette tâche au groupe de travail TLS. La spécification indique finalement la version minimale TLS à utiliser, 1.2, et une restriction sur les algorithmes à utiliser.

5.3. Négociation http2 en TLS

Next Protocol Negotiation (NPN) est le protocole utilisé pour négocier l'usage de SPDY avec les serveurs TLS. Ce n'était pas un protocole standardisé, après passage à l'IETF, il devint ALPN: Application Layer Protocol Negotiation. ALPN est promu pour son utilisation en http2, tandis que les clients et serveurs SPDY continuent d'utiliser NPN.

Le fait que NPN arriva en premier et que la standardisation d'ALPN prit du temps, eut la conséquence d'avoir des clients et serveurs http2 implémentant les deux extensions pour négocier http2. Puisque NPN est utilisé dans SPDY et que de nombreux serveurs offrant SPDY et http2, supporter à la fois NPN et ALPN fait sens.

ALPN diffère de NPN sur qui décide du protocole à utiliser. Avec ALPN, le client indique au serveur la liste des protocoles et ses préférences, le serveur obtient le dernier mot. Au contraire avec NPN, c'est le client qui impose son choix.

5.4. http2 pour http://

Comme indiqué précédemment, en HTTP 1.1 et en clair, la façon de négocier du http2 est de le demander au serveur avec l'en-tête Upgrade:. Si le serveur parle http2, il répond avec le code "101 Switching" et passe en http2 pour cette connexion. Bien que cette procédure d'upgrade coûte un aller-retour réseau, elle peut être conservée et réutilisée de manière bien plus efficace qu'une connexion HTTP1, amortissant ainsi le coût initial.

Même si certains porte-paroles de navigateurs ont indiqué qu'ils n'implémenteraient pas cette méthode, l'équipe d'Internet Explorer le fera, et curl la supporte déjà.

6. Le protocole http2

Assez sur la gestation et la politique qui nous ont mené ici. Plongeons dans les spécificités du protocole: les détails et concepts qui font http2.

6.1. Binaire

http2 est un protocole binaire.

Posons-nous un moment. Si vous avez déjà travaillé sur des choix de protocoles IP par le passé, il est probable que votre première réaction soit négative, et que vous argumentiez que les protocoles textuels sont bien supérieurs car les êtres humains peuvent forger des requêtes à la main via telnet, etc.

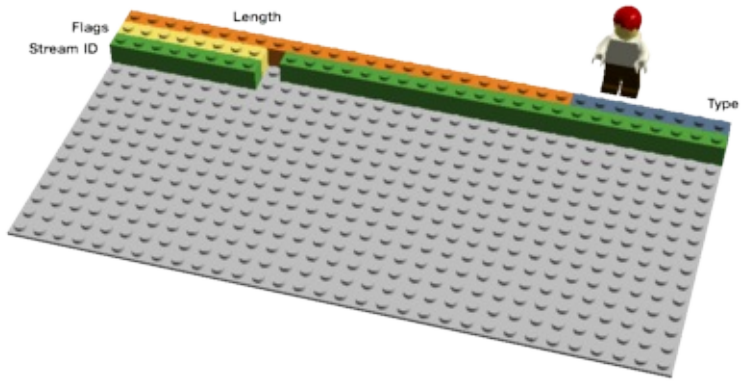
http2 est binaire pour rendre le découpage (framing en anglais) plus simple. Trouver le début et la fin d'une trame en HTTP 1.1 et dans les protocoles textuels en général est toujours très compliqué. En évitant les blancs optionnels et les diverses façons d'écrire la même chose, les implémentations sont plus simples.

De plus, cela permet de séparer plus nettement les parties du protocole et le découpage. Ce manque de séparation nette a toujours été perturbant avec HTTP1.

Le fait que le protocole utilise la compression et TLS diminue la valeur ajoutée de garder du texte pur puisqu'on ne verra pas de texte en clair passer de tout façon. Nous devons simplement nous habituer à utiliser un analyseur de trafic type Wireshark pour comprendre exactement ce qu'il passe au niveau protocolaire http2.

Débugger ce protocole se fera du coup probablement via des outils type curl ou le dissecteur http2 de Wireshark.

6.2. Le format binaire



http2 envoit des trames binaires. Il y a différentes types de trames envoyées et elles ont toutes le même format:

Longueur, Type, Flags, Identifiant de Flux (Stream ID) et contenu (payload).

Il y a dix trames différentes définies dans la spec http2 et deux sont fondamentales car répliquent le fonctionnement HTTP 1.1 avec DATA (données ou contenu) et HEADERS (en-tête). Je décris ces trames plus en détail par la suite.

6.3. Multiplexage de flux

L'identifiant de flux (Stream Identifier) mentionné dans le chapitre précédent permet d'associer à chaque trame http2 un "flux" (stream). Un flux est une association logique: une séquence indépendante et bidirectionnelle de trames échangées entre un client et un serveur via une connexion http2.

Une seule connexion http2 peut contenir plusieurs flux simultanés, avec chaque extrémité de la connexion pouvant entrelacer les flux. Les flux sont établis et utilisés unilatéralement ou de concert par le client ou le serveur. Ils peuvent être terminés par l'un ou l'autre côté de la connexion. L'ordre d'envoi des trames dans un flux est important. Le traitement des trames reçues se fait dans l'ordre de réception.

Multiplexer des flux implique que des groupes de trames sont mélangés (ou “mixés”) sur une même connexion. Deux (ou plusieurs) trains de données sont fusionnés en un seul puis scindés à nouveau à la réception. Voici deux trains:

Ils sont mélangés l'un dans l'autre sur la même connexion via multiplexage:

En http2 nous allons voir des dizaines voire des centaines de flux simultanés. Le coût de création d'un nouveau flux est très faible.

6.4. Priorités et dépendances

Chaque flux a une priorité (ou “poids”), utilisée pour dire à l'autre extrémité de connexion quels flux sont importants, en cas de congestion (manque de ressources) forçant le serveur à faire un choix sur les trames à envoyer en premier.

En utilisant la trame PRIORITY, un client peut indiquer au serveur quel autre flux dépend du flux actuel. Cela permet au client de construire un “arbre” de priorités dans lequel des “flux enfants” dépendront de la réussite des “flux parents”.

Les priorités peuvent changer dynamiquement, ce qui permettra aux navigateurs d'indiquer quelles images sont importantes quand on fait défiler une page remplie d'images, ou encore de prioriser les flux affichés à l'écran quand on passe d'un onglet à un autre.

6.5. Compression de l'en-tête.

HTTP est un protocole sans état (state-less). Cela veut dire que chaque requête doit apporter tous les détails requis au serveur pour traiter la requête, sans que le serveur ait besoin de conserver les informations ou meta-données des requêtes précédentes. Comme http2 ne change pas ce principe, il doit en faire de même.

Cela rend HTTP répétitif. Quand un client demande plusieurs ressources d'un même site, comme les images d'un site web, il y a aura une série de requêtes quasi identiques. Une série de points identiques appelle un besoin de compression.

Le nombre d'objets par page web croît comme vu précédemment, l'utilisation de cookies et la taille des requêtes croît également. Les cookies sont inclus dans toutes les requêtes, souvent identiques sur plusieurs requêtes.

Les tailles de requêtes HTTP 1.1 sont devenues tellement importantes qu'elles dépassent parfois la taille de la fenêtre TCP initiale, ce qui rend le tout très lent puisqu'un aller-retour est nécessaire pour qu'un ACK soit envoyé par le serveur. Encore une bonne raison de compresser.

6.5.1. La compression est un sujet délicat

Les compressions HTTPS et SPDY ont été vulnérables aux attaques BREACH et CRIME. En insérant un texte connu dans le flux et en analysant les changements résultant, l'attaquant peut déduire ce qui a été envoyé.

Compresser du contenu dynamique sans devenir vulnérable à une de ces attaques requiert de la réflexion. L'équipe HTTPbis s'y attelle.

Voici HPACK, Header Compression for HTTP/2 (compression d'en-tête pour HTTP/2), qui, comme son nom l'indique, est un format de compression spécialement créé pour les entêtes http2 et spécifié dans un draft IETF distinct. Le nouveau format, avec d'autres contre-mesures comme un bit qui interdit aux intermédiaires de compresser un en-tête spécifique ou du remplissage de trames (padding), devrait rendre plus compliqué l'exploitation de cette compression.

Voici les les mots de Roberto Peon (l'un des créateurs de HPACK):

> “HPACK a été conçu pour rendre difficile la fuite d'information avec une implémentation s'y conformant, rendre le codage et décodage très rapide et peu coûteux, fournir au destinataire un contrôle sur la taille du contexte de compression, permettre une réindexation par un proxy, et permettre une comparaison rapide avec des chaînes codées avec un algorithme Huffman.”.

6.6. Reset - changez de perspective

Un des inconvénients de HTTP 1.1: quand le message HTTP est envoyé avec un en-tête Content-Length d'une taille particulière, on ne peut pas l'interrompre facilement. Bien sûr vous pouvez toujours terminer la session TCP mais cela a un coût: renégocier le handshake TCP.

Une meilleure solution serait simplement d'interrompre le message et en démarrer un nouveau. Cela peut se faire en http2 avec la trame RST_STREAM qui permet d'éviter de gâcher de la bande passante et de terminer des connexion TCP.

6.7. Server push

Cette fonctionnalité est aussi connue comme "cache push". Cas typique: un client demande une ressource X au serveur, le serveur sait qu'en général ce client aura besoin de la ressource Z par la suite et la lui envoie sans que le client l'ait demandée. Cela aide le client à mettre Z dans son cache pour qu'elle soit disponible quand il en aura besoin.

Le Push Serveur (Server Push) doit être explicitement autorisée par le client et, quand bien même le client l'autorise, ce dernier se réserve le droit de terminer un flux "poussé" avec un RST_STREAM.

6.8. Contrôle de flux

Chaque flux individuel en http2 a sa propre fenêtre de flux annoncée que l'autre extrémité du flux peut utiliser pour envoyer des données. Si vous savez comment SSH fonctionne, le principe est très similaire.

Pour chaque flux, les deux extrémités doivent indiquer qu'ils ont davantage de place pour accepter des données supplémentaires, l'autre extrémité est autorisée à envoyer ce volume de données uniquement jusqu'à l'extension de la fenêtre. Seules les trames DATA (données) bénéficient du contrôle de flux.

7. Extensions

Le protocole oblige le destinataire à lire et ignorer toutes les trames inconnues utilisant un type de trame inconnu. Les deux parties peuvent ainsi négocier l'utilisation d'un nouveau type de trame de manière unitaire; ces trames ne seront pas autorisées à changer d'état et ne bénéficieront pas de contrôle de flux.

Le fait d'autoriser ou non les extensions dans http2 a été débattu pendant le développement du protocole avec des opinions pour et contre. Depuis le draft-12, la balance a penché en faveur du pour: les extensions sont autorisées.

Les extensions ne font pas partie du protocole actuel mais seront documentées en dehors de la spécification principale. En l'état, il existe deux types de trames qui pouvaient faire partie du protocole et qui seront probablement les premières trames définies comme extensions. Je les décris ici car elles sont populaires et étaient considérées auparavant comme des trames "natives":

7.1. Services Alternatifs

Avec http2 adopté, on peut suspecter que les connexions TCP seront plus longues (en temps) et maintenues actives plus longtemps que les connexions HTTP 1.x. Un client doit donc pouvoir se débrouiller avec une seule connexion par hôte/site et cette connexion pourrait rester ouverte pendant un certain temps.

Cela affectera comment les load balancers HTTP réagissent quand un site voudra que les utilisateurs se connectent sur un autre host, pour des raisons de performance ou pour réaliser une maintenance.

Le serveur enverra alors l'en-tête Alt-Svc: (ou la trame http2 ALTSVC) pour indiquer au client un service alternatif. Une autre route pour le même contenu, utilisant un autre service, host et numéro de port.

Un client est alors susceptible d'essayer de se connecter à ce service de manière asynchrone et n'utiliser que celui-ci s'il fonctionne.

7.1.1. TLS opportuniste

L'en-tête Alt-Svc permet à un serveur servant du contenu en http:// d'informer le client que le même contenu est aussi disponible en TLS.

Cette fonctionnalité est débattue. Cette connexion serait du TLS non authentifié et ne serait pas affichée comme "sécurisée", n'utiliserait pas de cadenas dans l'interface graphique du navigateur même si ce n'est pas du simple HTTP. Plusieurs personnes sont contre.

7.2. Bloqué

Ce type de trame doit être envoyée une seule fois par un client http2 quand des données doivent être envoyées alors que le contrôle de flux l'interdit. L'idée est que si votre implémentation reçoit une telle trame, cela vous indique que quelque chose cloche dans votre implémentation et que vous avez une performance suboptimale.

Une citation du draft-12, avant que cette trame ne soit retirée pour devenir une extension:

> “La trame BLOCKED est incluse dans ce draft pour en faciliter son expérimentation. Si les resultats de cette expérimentation ne donnent pas de feedback positif, elle pourra être retirée”

8. Le monde http2

Que donnera le monde une fois http2 adopté ? http2 sera-t-il adopté ?

8.1. Comment http2 affecte les humains ?

http2 n'est pas encore largement déployé ou utilisé. Nous ne pouvons savoir ce qu'il adviendra. Nous avons vu comment SPDY a été utilisé et pouvons estimer et calculer l'adoption de http2 à partir de ces expérimentations passées et actuelles.

http2 réduit le nombre d'aller-retours nécessaires et évite le head of line blocking en multiplexant et en se débarrassant des flux non voulus.

Il permet un nombre important de flux parallèles, plus important que ce qui est requis par les sites utilisant massivement le sharding aujourd'hui.

Si les priorités de flux sont utilisées correctement, on a une chance que les clients obtiennent les données les plus importantes avant les moins importantes. Tout compris, je dirais qu'il y de très bonnes chances pour que cela mène à un meilleur temps de chargement des pages et à des sites web plus réactifs. En résumé: une meilleure expérience web.

Dans quelle mesure cela sera plus rapide, je ne pense pas qu'on puisse y répondre pour l'instant. La technologie est encore très jeune et nous n'avons pas encore vu d'implémentations client et serveur tirer parti de toutes les possibilités que le protocole offre.

8.2. Comment http2 affectera le développement web ?

Avec le temps, les développeurs Web et les environnement de développement ont accumulé plein d'astuces et outils pour contourner les limitations de HTTP 1.1, souvenez-vous de tout ce qui a été indiqué en début de document pour justifier http2.

Beaucoup de contournements utilisés par défaut sans y penser, pénaliseront probablement les performances de http2 ou ne permettront pas l'utilisation des super pouvoirs de http2. Spriting et inlining ne devraient probablement pas être utilisés avec http2. Le sharding pénalisera http2 car il bénéficiera de moins de connexions.

Un problème ici est bien sûr que les sites et développeurs web devront avoir, au moins dans un premier temps, des versions s'accommodant de clients HTTP1.1 et http2. Et avoir les meilleures performances pour tous ces utilisateurs sera compliqué sans devoir afficher deux front-ends différents.

Pour ces seules raisons, je pense qu'il faudra du temps avant de tirer le plus grand bénéfice de http2.

8.3. Implémentations http2

Essayer de documenter des implémentations particulières dans un document comme celui-ci est futile et voué à l'échec car il sera obsolète en peu de temps. À la place, je vous explique la situation en termes plus généraux et vous renvoie à la liste exhaustive des implémentations http2 sur le site web.

Il y avait déjà un certain nombre d'implémentations très tôt, et ce nombre a augmenté pendant le travail sur http2. À ce jour, il existe 30 implémentations connues, et la plupart se basent sur la version finale de http2.

Firefox a été le navigateur le plus en avance, Twitter a suivi et propose ses services en http2. Google commença en avril 2014 à offrir ses services en http2 sur quelques serveurs et depuis mai 2014 via les versions de développement de Chrome. Microsoft a montré une préversion d'Internet Explorer supportant http2.

curl et libcurl supportent http2 non-TLS et TLS à partir d'une des librairies TLS disponibles.

8.3.1. Implémentations manquantes

Les deux serveurs les plus populaires, Apache HTTPD et Nginx, supportent tous deux SPDY mais à la date du 22 Septembre 2015, seul Nginx a publié une version supportant officiellement http2. Nginx a publié "nginx-1.9.5" et le module HTTP/2 pour Apache, nommé mod_h2, est en bonne voie pour être embarqué "bientôt" dans une nouvelle version.

8.4. Critiques courantes de http2

Pendant le développement de ce protocole, les débats n'ont jamais cessé et plusieurs personnes pensent que ce protocole a mal fini. Je voudrais ici lister quelques problèmes et les discuter :

8.4.1. "Le protocole a été écrit par Google"

Cela implique aussi que le monde devient davantage dépendant de Google. Ce n'est pas vrai. Le protocole a été développé à l'IETF de la même manière que tous les autres protocoles depuis 30 ans. Cela dit, nous reconnaissons tous le travail impressionnant réalisé par Google sur SPDY qui a permis de montrer qu'on pouvait déployer un nouveau protocole avec, de plus, des données chiffrées sur les gains obtenus.

Google a publiquement annoncé qu'ils retireraient le support de SPDY et NPN de Chrome en 2016 et qu'ils poussaient les serveurs à migrer vers HTTP/2 à la place.

8.4.2. "Ce protocole est seulement utile aux navigateurs"

C'est plutôt vrai. Une des principales raisons derrière le développement de http2 est de corriger le pipelining HTTP. Si votre cas d'usage n'avait pas besoin de pipelining alors il y a des chances que http2 n'améliore pas beaucoup votre situation. Ce n'est pas la seule amélioration du protocole mais une amélioration majeure.

Quand les différents services réaliseront la puissance des flux multiplexés sur une seule connexion, je pense que nous verrons davantage d'applications tirant parti de http2.

Les APIs REST simples utilisant HTTP 1.x ne verront pas d'avantages à passer à http2. Cela dit, il y aura peu d'inconvénients à passer à http2 pour la plupart des gens.

8.4.3. "Ce protocole n'est utile que pour les très gros sites"

Pas du tout. Les capacités de multiplexage vont énormément améliorer l'expérience des connexions à latence importante pour les petits sites n'ayant pas de présence mondiale.

Les gros sites ont déjà une présence mondiale et du coup des aller-retours moins longs vers la plupart des utilisateurs.

8.4.4. “Its use of TLS makes it slower”

Cela peut se révéler vrai. La négociation TLS ajoute un peu de latence, mais il existe des projets pour réduire encore les aller-retours en TLS. La surcharge du TLS par rapport à du texte en clair n'est pas neutre et a un impact CPU. L'impact en lui-même est sujet à discussions et mesures. Voir par exemple istlsfastyet.com pour avoir une source d'information.

Les opérateurs télécom et réseaux, par exemple l'ATIS Open Web Alliance, indiquent qu'ils nécessitent du trafic non chiffré pour permettre au cache et à la compression de fonctionner, notamment pour une expérience web rapide par satellite. http2 n'oblige pas à utiliser TLS, on ne devrait donc pas mélanger les deux.

De nombreux utilisateurs ont indiqué leur préférence à utiliser TLS et nous devrions respecter ce droit à la vie privée.

Des expérimentations ont montré que l'utilisation de TLS a une meilleure probabilité de succès qu'avec un protocole en clair sur le port 80. Il y a trop de boîtes intermédiaires sur le réseau qui vont perturber le trafic sur le port 80 en imaginant qu'il n'y a que du trafic HTTP 1.1 sur ce port.

Enfin, grâce aux flux multiplexés que permet http2 sur une seule connexion, les navigateurs initieront moins de négociations TLS et iront finalement plus vite qu'avec HTTPS en HTTP 1.1.

8.4.5. "Pas de l'ASCII = inutilisable"

Oui, nous préférons voir des protocoles en clair car le debugging en est facilité. Mais un protocole texte est aussi davantage sujet à des problèmes de parsing.

Si vous ne pouvez pas vous faire à un protocole binaire, alors vous ne pouviez pas vous faire non plus à TLS ni à la compression, utilisés depuis longtemps en HTTP 1.x.

8.4.6. "Ce n'est pas plus rapide que HTTP/1.1"

Cela est bien sûr sujet à débat sur comment qualifier "plus rapide"; les tests menés lors des expérimentations SPDY montraient des temps de chargement de pages web plus rapides (voir “How Speedy is SPDY?” par University of Washington et “Evaluating the Performance of SPDY-enabled Web Servers” par Hervé Servy). Idem en http2 avec d'autres tests. Je souhaite voir davantage de tests publiés. Un premier essai réalisé par httpwatch.com a tendance à montrer que HTTP/2 répond aux promesses.

http2 est clairement plus rapide dans certains scénarios, en particulier avec les scénarios où une connexion à latence importante est utilisée avec un site comportant beaucoup de ressources à charger, ce nombre ayant tendance à augmenter.

8.4.7. "Il y a violation de couches"

Sérieusement ? Les couches OSI ne sont pas intouchables, nous avons franchi des zones grises avec http2 dans l'intérêt de le rendre efficace dans les limites imposées.

8.4.8. "Cela ne corrige pas certaines limitations HTTP/1.1"

C'est vrai. L'objectif de conserver certains paradigmes HTTP/1.1 font que certaines vieilles fonctionnalités restent. Par exemple les en-têtes conservent les cookies, l'authentification et davantage. Mais en maintenant ces principes nous avons un protocole qu'il est possible de déployer sans réécrire ou remplacer des parties protocolaires considérables. http2 est juste une nouvelle couche de framing.

8.5. http2 sera-t-il largement déployé ?

Il est encore trop tôt pour le dire, mais je peux le deviner et l'estimer, voici comment.

Les esprits négatifs montreront "regardez comme IPv6 a marché" comme un exemple qui a pris des décennies pour juste commencer à être largement déployé. http2 n'est pas IPv6. C'est un protocole au-dessus de TCP utilisant les mécanismes d'upgrade HTTP, un port standard, TLS, etc. Il ne requiert pas un changement de la plupart des routeurs et firewalls.

Avec SPDY, Google a prouvé au monde que l'on pouvait déployer et utiliser un nouveau protocole en peu de temps. Même si la somme des serveurs SPDY avoisine les 1% aujourd'hui, le volume de données est plus important. Certains des plus gros sites utilisent SPDY.

http2, basé sur les mêmes principes que SPDY et ratifié par l'IETF, devrait être encore plus largement déployé. Les déploiements SPDY ont toujours été limités par le syndrome "inventé par Google"

Il y a plusieurs navigateurs importants derrière http2. Des représentants de Firefox, Chrome, Safari, Internet Explorer et Opera ont tous indiqué leur intention de livrer des navigateurs avec http2 et ont montré des implémentations fonctionnelles.

Plusieurs gros sites proposent http2, avec Google, Twitter et Facebook et nous espérons que http2 sera ajouté aux serveurs Apache HTTP et nginx. H2o est un nouveau serveur très rapide supportant http2 avec beaucoup de potentiel.

Les plus gros proxy, HAProxy, Squid et Varnish ont mentionné leur intention de supporter http2.

Je pense qu'il y aura davantage d'implémentations quand la RFC sera ratifiée.

Durant tout 2015, la quantité de trafic en http2 n'a cessé d'augmenter. Au début Septembre, sur Firefox 40, il génère 13% de tout le trafic HTTP, et 27% du trafic HTTPS, tandis que Google voit 18% de HTTP/2. Il faut noter que Google déroule d'autres expérimentations en parallèle (voir QUIC en 12.1), ce qui rend la quantité de trafic http2 plus basse que ce qu'elle aurait pu être.

9. http2 et Firefox

Firefox a suivi les drafts de près et fourni des protocoles de test de http2 depuis des mois. Pendant le développement du protocole http2, les clients et serveurs devaient s'accorder sur le draft du protocole à utiliser, ce qui compliquait les tests. Soyez juste conscient de vous mettre d'accord sur le draft à utiliser entre client et serveur.

9.1. Assurez-vous de l'activer

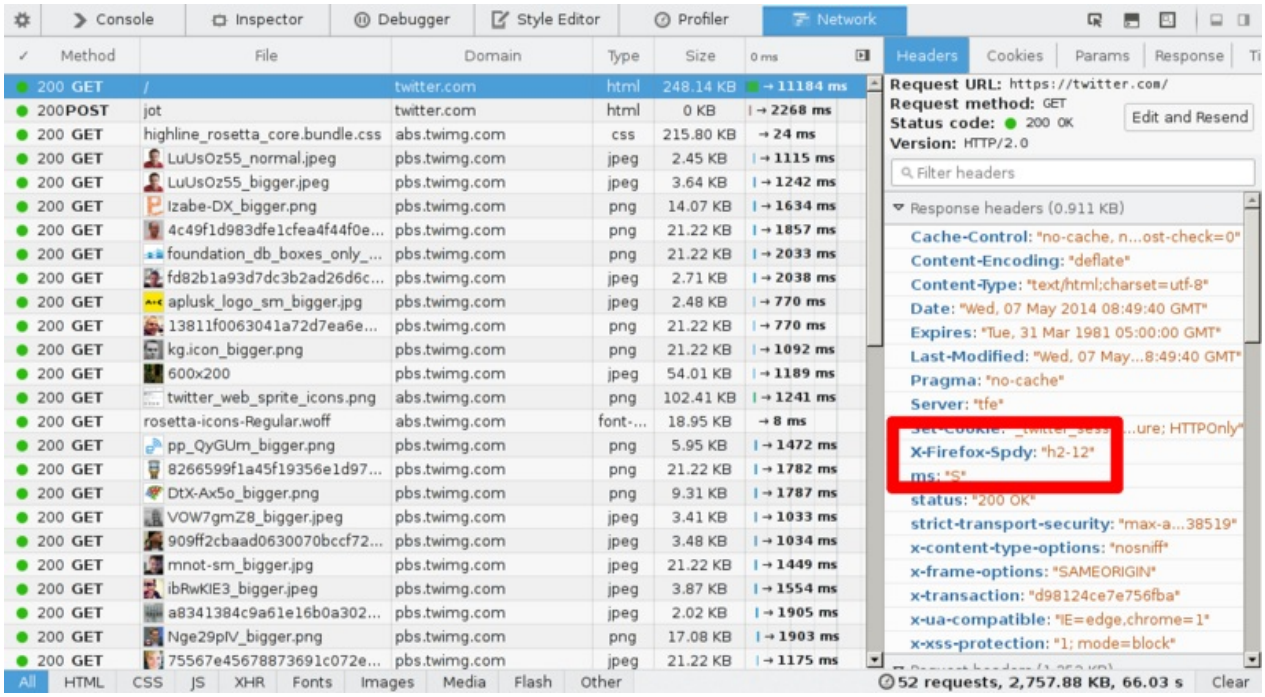
Toute version de Firefox 35 et ultérieure, depuis le 13 janvier 2015, a le support http2 activé par défaut.

Allez dans "about:config" depuis la barre d'URL et cherchez l'option appelée "network.http.spdy.enabled.http2draft". Assurez-vous qu'elle est à "true". Firefox 36 ajouta une option activée par défaut "network.http.spdy.enabled.http2". Cette dernière contrôle la version finale de http2 tandis que la première active ou non la négociation des versions drafts de http2. Les deux sont activées depuis Firefox 36.

9.2. TLS uniquement

Souvenez-vous que Firefox n'implémente http2 que sur TLS. Vous ne verrez du http2 que si vous allez sur les sites https:// qui offrent http2.

9.3. Transparent!



Il n'y a pas d'indication graphique que vous utilisez http2. Vous ne pouvez pas le voir facilement. Une manière de le trouver est d'activer "Web developer->Network" et de vérifier l'en-tête de réponse. Elle doit comporter "HTTP/2.0", Firefox insère son propre entête "X-Firefox-Spdy:" comme montré sur la copie d'écran précédente.

Les en-têtes que vous voyez dans l'outil ont été convertis du format binaire http2 vers un format qui ressemble aux en-têtes HTTP 1.x.

9.4. Voir l'utilisation http2

Il existe des plug-ins Firefox permettant de visualiser si un site utilise http2. En voici un “SPDY Indicator”.

10. http2 et Chromium

L'équipe Chromium a implémenté http2 depuis un certain temps via les canaux beta et dev. Depuis Chrome 40, sorti le 27 janvier 2015, http2 est activé par défaut pour un certain nombre d'utilisateurs. Ce nombre a commencé petit pour devenir important au fil du temps.

Le support SPDY sera retiré. Dans un blog, il est annoncé en février 2015:

> “Chrome supporte SPDY depuis Chrome 6, mais comme les bénéfices sont présents dans HTTP/2, il est temps de lui dire au revoir. Le support SPDY sera retiré début 2016.”

10.1. Assurez-vous de l'activer

Allez sur "chrome://flags/#enable-spdy4" dans la barre d'URL et cliquez sur "enable" si ce n'est déjà fait.

10.2. TLS uniquement

Souvenez-vous que Chrome n'implémente http2 que sur TLS. Vous ne verrez du http2 que si vous allez sur les sites https:// qui offrent http2.

10.3. Voir l'utilisation http2

Il existe des plug-ins Chrome permettant de visualiser si un site utilise http2. En voici un: “SPDY Indicator”.

10.4. QUIC

Les tests QUIC en cours masquent un peu les résultats HTTP/2. (voir chapitre 12.1)

11. http2 et curl

Le projet curl a fourni un support http2 expérimental depuis septembre 2013.

Dans l'esprit curl, nous voulons supporter toutes les fonctionnalités http2 possibles. curl est souvent utilisé comme outil de test et nous voulons que ce soit le cas pour http2 également.

curl utilise une librairie distincte nghttp2 pour la couche http2. curl requiert nghttp2 1.0 ou plus.

Notez qu'actuellement, sous linux, curl et libcurl ne sont pas toujours délivrés avec le support du protocole HTTP/2 activé.

11.1. Ressemblance HTTP 1.x

En interne, curl convertit les en-têtes entrant http2 en en-têtes du style HTTP 1.x, pour qu'ils apparaissent similaires à l'utilisateur. Cela permet une transition plus simple pour toute personne utilisant curl en HTTP aujourd'hui. Idem pour les en-têtes sortants. Donnez à curl des en-têtes au format HTTP 1.x et il les convertira en http2 à la volée vers les serveurs http2. Cela permet aux utilisateurs de ne pas forcément se soucier de la version de HTTP utilisée.

11.2. En clair

curl supporte http2 sur TCP via l'en-tête Upgrade:. Si vous initiez une requête HTTP en demandant HTTP 2, curl demandera au serveur de mettre à niveau (Upgrader) sa connexion en http2.

11.3. Quelles librairies TLS ?

curl supporte différentes librairies TLS et c'est toujours valide pour http2. La difficulté avec TLS et http2 est le support de ALPN et potentiellement NPN.

Compilez curl avec des versions récentes d'OpenSSL ou NSS pour avoir ALPN et NPN. Avec GnuTLS et PolarSSL vous n'aurez que ALPN et pas NPN.

11.4. Ligne de commande à utiliser

Pour indiquer à curl d'utiliser http2, en clair ou TLS, utilisez l'option `--http2` ("tiret tiret http2"). curl utilise HTTP/1.1 par défaut, d'où cette option nécessaire pour http2.

11.5. Options libcurl

11.5.1 Activez HTTP/2

Votre application utilisera http:// ou https:// comme d'habitude, mais vous pouvez régler la variable curl_easy_setopt de `CURLOPT_HTTP_VERSION` vers `CURL_HTTP_VERSION_2` pour que libcurl tente d'utiliser http2. Il le fera en best effort sinon utilisera HTTP 1.1.

11.5.2 Multiplexage

Etant donné que libcurl essaye de maintenir ses anciens comportements dans la mesure du possible, vous devez activer le multiplexage HTTP/2 pour votre application via l'option CURLMOPT_PIPELINING. Sinon elle continuera à utiliser une requête à la fois par connexion.

Un autre détail à garder à l'esprit et que si vous demandez plusieurs transferts en simultanés à libcurl, en utilisant son interface multi, une application peut très bien commencer autant de transfert que voulu, et que si vous préférez que libcurl attende un peu pour les placer tous sur la même connexion, plutôt que d'ouvrir une connexion pour chacun, utilisez l'option CURLOPT_PIPEWAIT pour chaque transfert que vous préférez attendre.

11.5.3 Server push

libcurl 7.44.0 et plus supporte le server push de HTTP/2. Vous pouvez tirez des avantages de cette fonctionnalité en configurant un callback de push avec l'option CURLMOPT_PUSHFUNCTION. Si le push est accepté par l'application, il créera un nouveau transfert en tant que CURL easy handle et l'utilisera pour délivrer le contenu, comme tout autre transfert.

12. Après http2

Plusieurs décisions et compromis sont intervenus pour concevoir http2. Avec http2 en cours de déploiement, on a un moyen éprouvé de mise à niveau de version de protocole, qui permettra le développement d'autres révisions de protocoles. Cela montre aussi qu'une même infrastructure peut supporter plusieurs versions de protocole en parallèle. Peut-être pouvons-nous garder l'ancien protocole une fois le nouveau mis en place ?

http2 a pas mal de principes hérités de HTTP 1 pour qu'il soit possible de proxifier du trafic entre HTTP 1 et http2. Certains principes freinent le développement et l'innovation. Peut-être http3 pourra-t-il passer outre ces principes ?

Que pensez-vous qu'il manque à http ?

12.1. QUIC

Le protocole QUIC (Quick UDP Internet Connections) de Google est une expérimentation très intéressante, menée dans le même esprit que SPDY. QUIC est un substitut à TCP + TLS + HTTP/2 implémenté avec UDP.

QUIC permet la création de connexions avec moins de latence, il résout la perte de paquet en ne bloquant qu'un flux particulier au lieu de tous les flux en HTTP/2 et il permet d'établir des connexions à travers différentes interfaces réseau, et du coup, couvre des problématiques que MPTCP résout.

QUIC est pour l'instant uniquement disponible à travers Chrome et les serveurs Google et le code n'est pas facilement réutilisable, même s'il existe une libquic pour ce faire justement. Le protocole a été soumis en tant que draft à l'IETF transport working group.

13. Lecture complémentaire

Si vous pensez que ce document était léger en contenu ou détails techniques, voici quelques ressources pour satisfaire votre curiosité:

- La mailing-list HTTPbis et ses archives : https://lists.w3.org/Archives/Public/ietf-http-wg/
- La spécification http2 au format HTML : https://httpwg.github.io/specs/rfc7540.html
- Les aspects réseaux http2 de Firefox : https://wiki.mozilla.org/Networking/http2
- Détail d'implémentation http2 dans curl : https://curl.haxx.se/docs/http2.html
- Les sites web http2: https://http2.github.io/ et en particulier la FAQ : https://http2.github.io/faq/
- Le chapitre de Ilya Grigorik sur HTTP/2 dans son livre “High Performance Browser Networking”: https://hpbn.co/http2/

14. Remerciements

L'inspiration et l'assemblage Lego : Mark Nottingham

Les tendances HTTP : https://httparchive.org/

Le graphique RTT (Aller-Retour) vient des présentations de Mike Belshe.

Mes enfants Agnès et Rex pour m'avoir prêté leurs Lego.

Merci à mes amis pour les relectures et commentaires : Kjell Ericson, Bjorn Reese, Linus Swalas et Anthony Bryan. Votre aide est appréciée et a réellement amélioré ce document!

Pendant les diverses itérations du document, les personnes suivantes ont fourni ou suggéré des améliorations: Mikael Olsson, Remi Gacogne, Benjamin Kircher, saivlis, florinandrei-tp, Brett Anthoine, Nick Parlante, Matthew King, Nicolas Peels, Jon Forrest, sbrickey, Marcin Olak, Gary Rowe, Ben Frain, Mats Linander, Raul Sile, Alex Lee et Richard Moore.

Le traducteur de cette version française, Olivier Cahagne, souhaite remercier Luc Trudeau, Mehdi Achour, Pascal Borreli et Remi Gacogne pour leurs relectures, nombreuses corrections et suggestions.