Introduction aux protocoles Web

Temps de lecture : 10 minutes

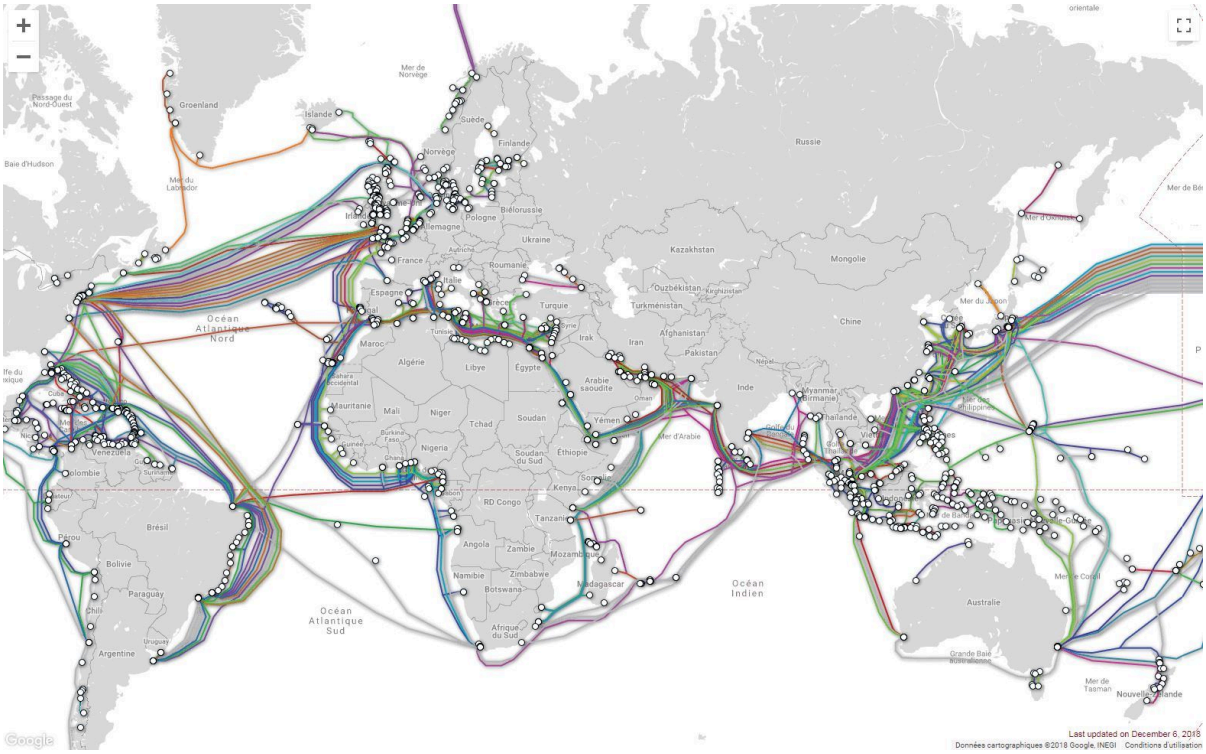
Nous allons parler d'un autre sujet passionnant : les **protocoles réseaux**, et comment Internet fonctionne.

Même chose que pour le chapitre précédent, vous pouvez passer rapidement, mais nous vous conseillons de lire pour comprendre comment fonctionne Internet, et les serveurs Web !

L'infrastructure d'Internet

Par où voyagent vos SMS, vos appels, vos emails, Internet ? Par des câbles pour 95% des communications et 99,9% lorsque qu'elles sont transcontinentales.

Voici une petite carte des câbles sous-marins :



Sans surprise les GAFAM (Google, Amazon, Facebook, Apple et Microsoft) investissent dans plus de 50% de ces câbles : sans eux pas d'Internet ! Il en existe aujourd'hui environ 500.

Par exemple, le débit du câble déployé par Google en ce moment entre les Etat-Unis et la France est de 30 TeraBits par secondes. Oui, ça fait une bonne bande passante.

Super, mais comment un ordinateur et un serveur se retrouvent là dedans ?

Ils utilisent de nombreux protocoles pour se retrouver et communiquer, dont une adresse unique, temporaire ou permanente que vous connaissez : **l'adresse IP** (pour Internet Protocol, nous allons l'étudier).

Nous avons une adresse et des câbles pour communiquer, mais cela ne suffit pas il va falloir transformer nos données en signaux physiques (signaux électriques dans des câbles) puis les envoyer au bon endroit, puis les interpréter correctement (mais aussi gérer les erreurs et plein d'autres choses).

Pour réaliser toutes ces tâches très complexes, vous l'imaginez bien, il existe ce qu'on appelle la **protocol stack** ou pile de protocoles et cela ressemble à ça :

TCP/IP	OSI Model	Protocols
Application Layer	Application Layer	DNS, DHCP, FTP, HTTPS, IMAP, LDAP, NTP, POP3, RTP, RTSP, SSH, SIP, SMTP, SNMP, Telnet, TFTP
	Presentation Layer	JPEG, MIDI, MPEG, PICT, TIFF
	Session Layer	NetBIOS, NFS, PAP, SCP, SQL, ZIP
Transport Layer	Transport Layer	TCP, UDP
Internet Layer	Network Layer	ICMP, IGMP, IPsec, IPv4, IPv6, IPX, RIP
Link Layer	Data Link Layer	ARP, ATM, CDP, FDDI, Frame Relay, HDLC, MPLS, PPP, STP, Token Ring
	Physical Layer	Bluetooth, Ethernet, DSL, ISDN, 802.11 Wi-Fi

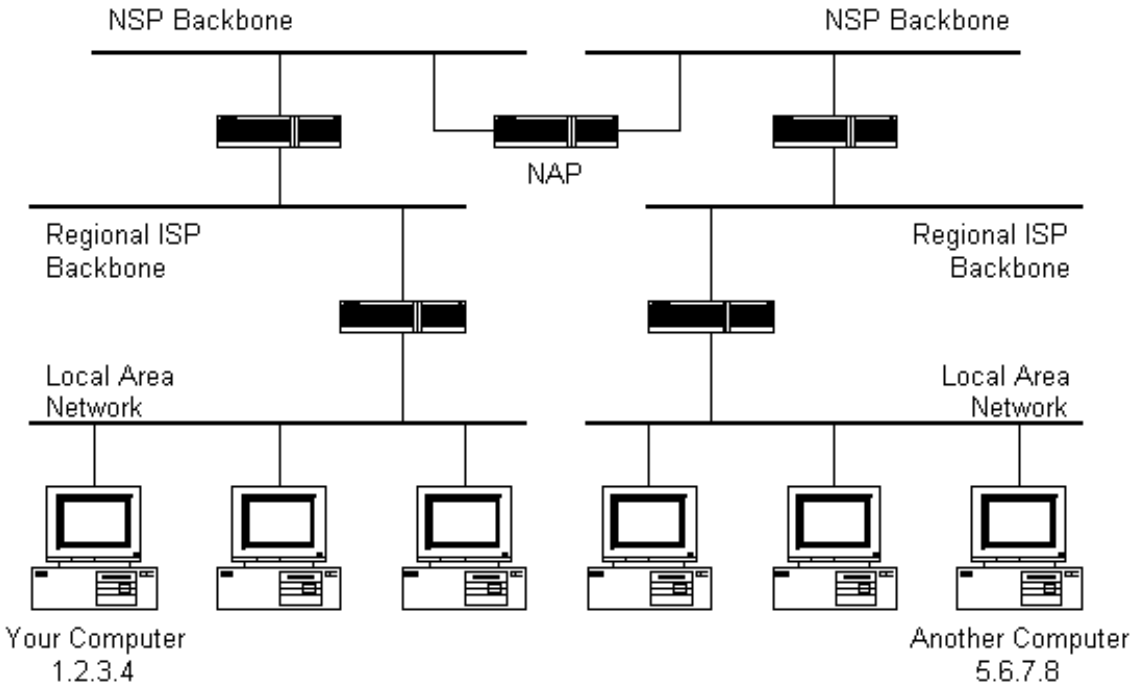
Avant de les expliquer, nous allons continuer sur **l'infrastructure réseau**.

Internet est constitué de quelques immenses réseaux (les Network Service Providers ou **NSP**). Ces larges réseaux contrôlent les infrastructures les plus chères et les plus rapides (les autoroutes d'Internet en quelque sorte), c'est eux qui opèrent les câbles sous marins.

Ils se connectent entre eux dans des points d'accès géants appelés **NAP** (pour Network Access Point) et **MAE** (pour Metropolitan Area Exchange).

Ces gros opérateurs vendent de la bande passante aux fournisseurs d'accès Internet qui ont en charge les réseaux locaux plus petits (Free, Bouygues, SFR par exemple) et eux vendent leur réseaux aux entreprises et aux particuliers.

Voici un schéma pour résumer :



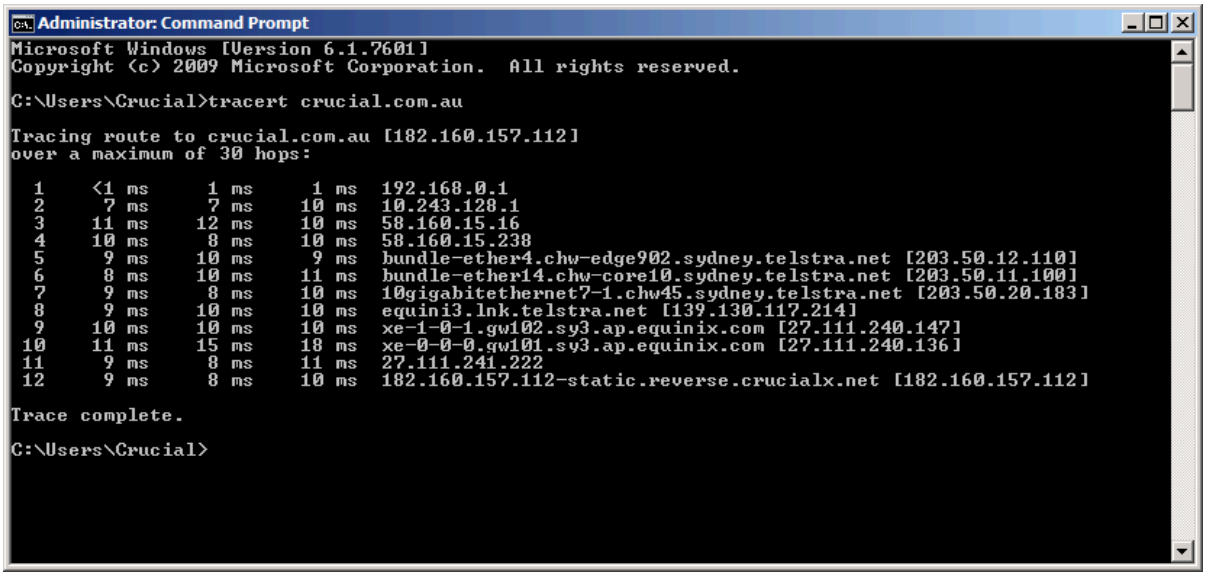
Le routing des informations

Après l'infrastructure il faut passer au routing : comment les informations trouvent leur chemin sur le réseau ?

Lorsque vous envoyez des informations par Internet ils suivent un chemin précis, pour le connaître tapez :

Sur Windows `tracert google.com` ou sur Linux/Mac `traceroute google.com`.

Vous obtiendrez quelque chose comme ça :



Impressionné ? C'est le chemin que chaque bribe d'informations que vous avez envoyée à Google a fait.

Mais pourquoi y a t-il besoin de 12 étapes ? Tout simplement parce que votre ordinateur n'a aucune idée de l'adresse des serveurs de Google.

C'est le rôle des **routers** qui se trouvent à chaque interconnexion du réseau. Ce sont les boites en noir sur le schéma précédent.

Chaque router connaît toutes les adresses IP en dessous de lui mais pas au dessus.

Lorsque l'information arrive à un router, il va examiner si il connait l'adresse IP de destination.

S'il la connaît il va l'envoyer sur le bon réseau, et le router du dessous l'enverra au bon sous-réseau et ainsi de suite jusqu'à l'ordinateur de destination.

Si il ne la connaît pas il va l'envoyer au router du niveau au dessus, et ainsi de suite jusqu'aux routers du NSP tout en haut.

Ces routers savent sur quel grand réseau l'envoyer, ensuite cela va descendre de sous réseaux en sous réseaux grâce aux routers de chaque niveau.

Les noms de domaine et la résolution de l'adresse IP

D'accord, mais nous avons envoyé nos informations à `google.com` et non à une adresse IP. Comment les routers peuvent savoir où envoyer les informations avec une chaîne de caractères ?

Pour cela, il faut interroger un gigantesque système distribué appelé **DNS** (Domain Name System).

Cela fonctionne avec un système hiérarchique comme le routing en utilisant l'adresse IP.

En fait votre système d'exploitation contient un résolveur DNS qui est paramétré par défaut en fonction de votre FAI.

Votre système connaît donc une adresse IP pour un des 13 ensembles de serveurs racines, 12 organisations contrôlent ces serveurs racines (notamment la NASA, l'ICANN ou le département de la défense américaine).

Ces serveurs connaissent toutes les adresses IP des serveurs s'occupant des extensions (.fr, .com, .org etc), ou **TLD** (pour Top Level Domain).

Les serveurs de niveau **TLD** vous donnent ensuite le serveur qui a la liste des adresses IP et des noms de domaine pour l'extension.

Ce dernier serveur vous renverra l'adresse IP résolue à partir du nom de domaine de départ, `google.com`.

Cette résolution DNS ne se fait pas pour toutes les requêtes, elle est mise en cache par votre navigateur notamment en fonction de la **TTL** (Time To Live) spécifié par l'hébergeur.

Retour à nos protocoles.

Qu'est-ce qu'un protocole ?

Un protocole de communication est un système de règles qui permettent à deux ou plus entités d'un système de communication de transmettre des informations en utilisant tout système de variation d'une quantité physique.

Pensez par exemple aux modulations de fréquence (FM) ou d'amplitude (AM) pour les ondes radios.

La couche applicative (`Application Layer`)

Occupons nous maintenant de la couche applicative.

TCP/IP	OSI Model	Protocols
Application Layer	Application Layer	DNS, DHCP, FTP, HTTPS, IMAP, LDAP, NTP, POP3, RTP, RTSP, SSH, SIP, SMTP, SNMP, Telnet, TFTP
	Presentation Layer	JPEG, MIDI, MPEG, PICT, TIFF
	Session Layer	NetBIOS, NFS, PAP, SCP, SQL, ZIP
Transport Layer	Transport Layer	TCP, UDP
Internet Layer	Network Layer	ICMP, IGMP, IPsec, IPv4, IPv6, IPX, RIP
Link Layer	Data Link Layer	ARP, ATM, CDP, FDDI, Frame Relay, HDLC, MPLS, PPP, STP, Token Ring
	Physical Layer	Bluetooth, Ethernet, DSL, ISDN, 802.11 Wi-Fi

Les protocoles de cette couche inclus le `DNS` que nous avons vu, `FTP` pour le transfert de fichier, `SMTP` pour les emails, `SSH` pour se connecter à distance de manière sécurisée et `HTTP` pour se connecter au Web.

Nous avons parlé d'Internet qui est l'immense réseau permettant d'échanger des informations.

Le protocole `HTTP`

Intéressons nous maintenant au Web, (www ou World Wide Web) qui est le moyen de consulter, avec un navigateur, des pages accessibles sur des sites.

Pour communiquer un navigateur et un serveur Web utilisent le protocole `HTTP` (Hypertext Transfer Protocol).

Il s'agit d'un protocole client-serveur : les requêtes sont envoyées par une entité appelée **l'agent utilisateur**. Cela peut être un navigateur, mais aussi un robot de crawl de Google ou même un terminal.

Ce protocole est **sans état** : il n'y a pas de lien entre deux requêtes qui sont effectuées successivement au serveur.

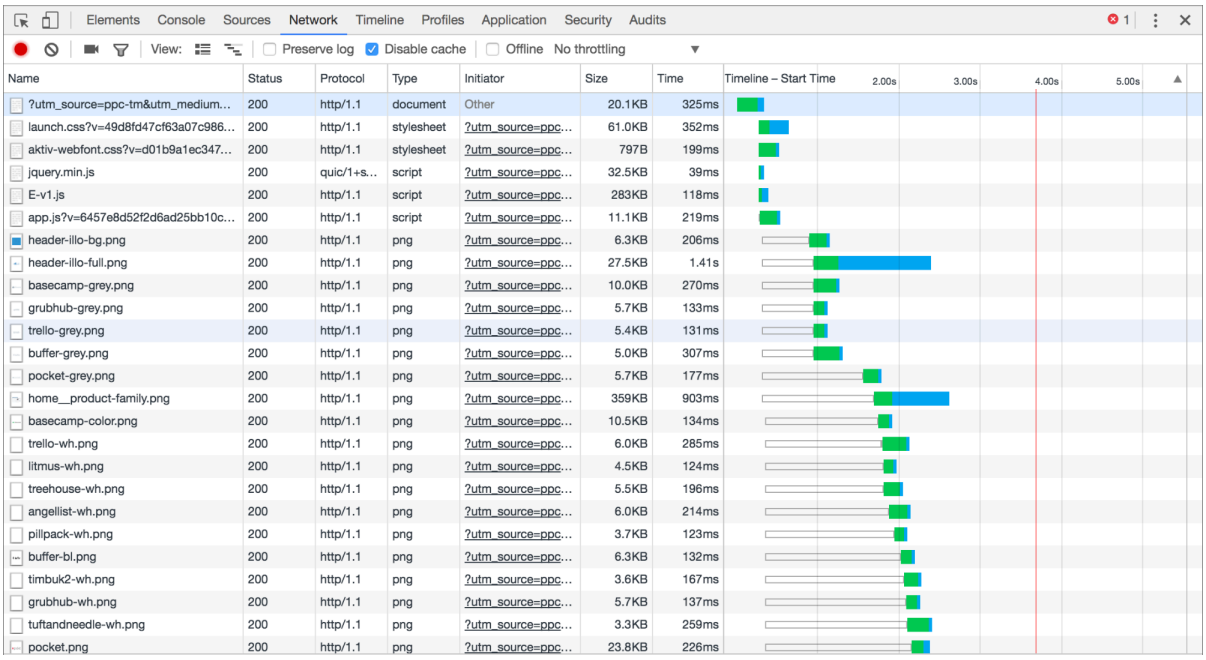
Depuis la version 1.1 du protocole, la connexion `TCP` (nous verrons cela juste après) avec le serveur est maintenue par défaut. Ce qui permet le `pipelining` : c'est-à-dire l'envoi de plusieurs requêtes sans attendre les réponses, et donc un gain substantiel en performance.

Dans la version 1.0 d'HTTP une connexion `TCP` est ouverte avant chaque requête puis fermée à la fin de chaque requête ce qui est plus coûteux en performances.

En effet dans les versions 0.9 et 1, le modèle de chargement était en cascade : le navigateur demande la page HTML, il découvre que la page HTML a besoin de CSS il fait une requête pour le CSS, il découvre qu'il a besoin aussi du JavaScript et fait une autre requête. A chaque fois il attend la réponse du serveur et le chargement complet de la ressource.

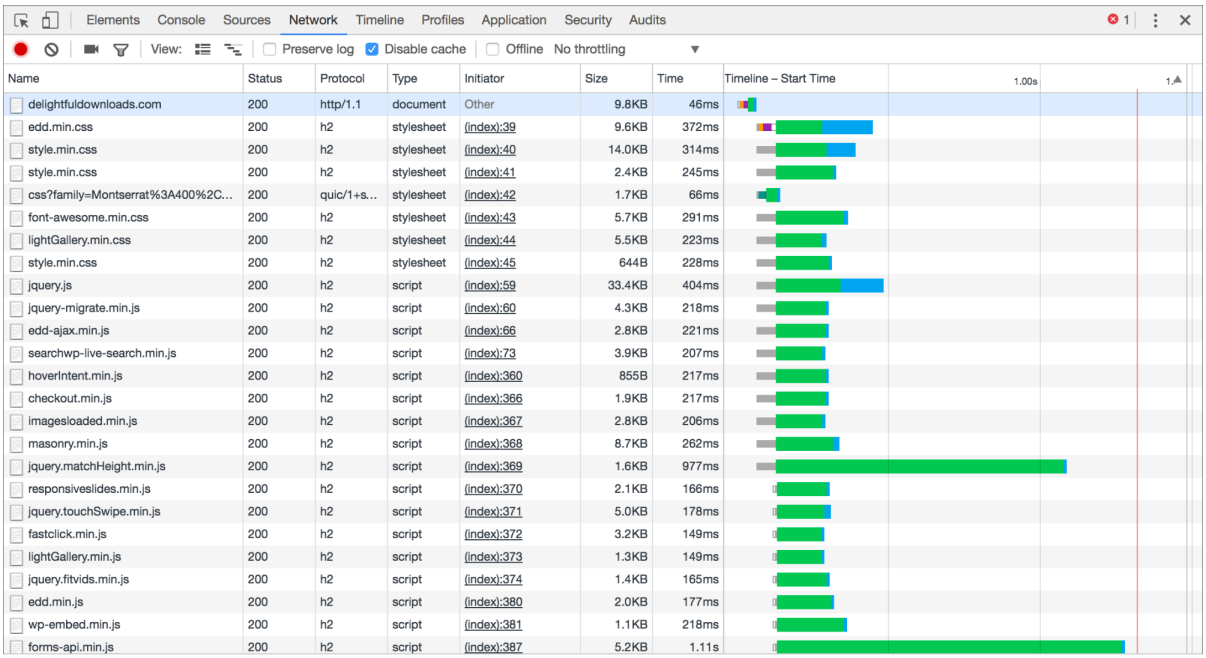
Dans la version HTTP 1.1, pour augmenter les performances, les navigateurs ouvrent entre 2 et 8 connexions `TCP` (8 est le maximum) avec le serveur et font donc au maximum 8 requêtes en parallèle.

Les requêtes sont donc envoyées avec un système de file d'attente, donnant un chargement des ressources en cascade :



La version 2 du protocole permet le `multiplexing` c'est-à-dire d'envoyer plusieurs requêtes en même temps sur la même connexion `TCP`, ce qui augmente grandement les performances. Il permet également le `Server Push` : c'est-à-dire l'envoi de ressources HTML, CSS et JavaScript avant que le client ne les demande. Ces deux particularités permettent des gains de performance énormes (3 à 4 fois plus rapides).

La version 2 permet l'envoi parallèle de ressources sur une seule connexion `TCP` et économise donc 5 connexions `TCP` !



Faites le testici.

A noter dans ce test HTTP/1.1 vs HTTP/2 qu'en fait il s'agit de HTTP/2 + HTTPS. En effet, le protocole HTTP/2 n'est implémenté par les navigateurs qu'avec HTTPS.

Pourquoi ? Car tous les navigateurs veulent obliger les sites à passer en HTTPS car c'est un protocole sécurisé comme nous allons le voir.

Les navigateurs mettent déjà des cadenas rouges et des avertissements si le site n'utilise pas HTTPS et il ne serait pas étonnant que le HTTP soit interdit par les navigateurs dans les prochaines années.

A noter également que le test utilise le `server push` et est parfaitement optimisé. L'implémentation "simple" d'HTTP/2 ne vous donnera pas de telles performances sans configuration.

HTTP/2 est aujourd'hui utilisé par 34% des 10 millions plus gros sites du monde.

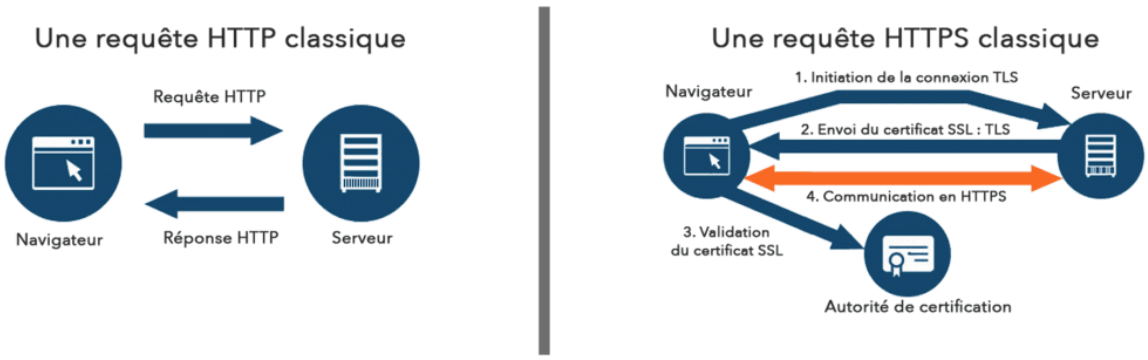
Nous verrons dans les prochaines leçons que le protocole `HTTP` comprend un format spécifique pour les requêtes et les réponses.

Le protocole `HTTPS`

Le protocole HTTPS (pour Hypertext Transfer Protocol Secure) est la combinaison du protocole HTTP et d'un protocole de chiffrement SSL (Secure Sockets Layer) ou TLS (Transport Layer Security).

Cela peut donc être toute combinaison entre HTTP/1, HTTP/1.1, HTTP/2 d'un côté et SSL ou TLS de l'autre.

HTTPS permet au navigateur de vérifier l'identité du site auquel il accède, grâce à un certificat d'authentification émis par une autorité tierce, réputée fiable.



Etape 1 : Le client demande une connexion sécurisée au serveur en faisant une requête HTTPS sur le port 443 (nous étudierons les ports plus loin), ou le serveur passe en HTTPS de son propre chef si la requête est faite en HTTP.

Etape 2 : Le serveur confirme qu'il gère le HTTPS et envoi un **certificat** (émis par une autorité de certification tierce, la plupart du temps ce certificat est payant). Ce certificat permet de

garantir l'identité du serveur par rapport au domaine. Le certificat contient également une **clé publique** utilisée pour le **chiffrement asymétrique** (nous expliquerons en détails cela dans le chapitre où nous vous apprendrons à mettre en place un serveur HTTPS).

Etape 3 : le navigateur vérifie auprès de l'autorité de certification que le certificat est bien valide (authentique, pas périmé etc).

Etape 4 : Le navigateur va générer une clé de session et va l'encrypter en utilisant la clé publique du serveur. Il envoi cette clé de session au serveur que seul lui peut décrypter.

Etape 5 : Le serveur décrypte la clé de session avec sa clé privée. Ensuite, grâce à la clé de session qui est maintenant partagée par le client et le serveur ils peuvent passer au **chiffrement symétrique** qui est beaucoup plus rapide pour le reste des échanges.

Tout le trafic client serveur peut maintenant être totalement crypté et indéchiffrable sauf par le client ou le serveur.

Grâce à ce protocole le serveur et client se sont identifiés de manière sécurisée et se sont échangé **de manière chiffrée** un code pour dialoguer de manière secrète.

Le protocole `SMTP`

Prenons un autre protocole en exemple, le protocole `SMTP` (Simple Mail Transfer Protocol) utilisé pour l'échange de courriels.

Nous n'allons pas le détailler car vous n'en aurez très probablement pas besoin.

Pour dialoguer un client commence par envoyer au serveur SMTP l'expéditeur du message, puis le ou les destinataires du message, Le serveur SMTP vérifie à fois fois l'existence de l'expéditeur et du destinataire.

Ensuite le client envoi les données de l'email. Le serveur confirme la réception et coupe la session.

Il enverra ensuite les messages à un autre serveur SMTP proche du destinataire en utilisant le même protocole.

Enfin le serveur SMTP de destination enverra l'email au destinataire final en utilisant les protocoles `POP` ou `IMAP`.

La couche transport (`Transport Layer`)

Intéressons nous maintenant au transport de l'information sur Internet avec la deuxième couche : la `transport layer`.

Les informations sur Internet sont découpées en `paquets` qui sont envoyées sur le réseau. La couche transport permet de gérer la fiabilité des échanges (que faire si un paquet s'est perdu sur le réseau ?), l'ordre d'arrivée des paquets et leur intégrité.

Le protocole `TCP`

Le protocole `TCP` (pour `Transmission Control Protocol`) est un protocole permettant le transport fiable de données découpées en paquets TCP.

C'est le protocole qui transporte 95% du trafic Internet.

Il permet de gérer **deux flux d'**`octets` : un dans chaque direction car il s'agit d'une connexion bidirectionnelle. Le serveur peut envoyer des données au client et le serveur au client (nous supposons un modèle client/serveur mais une connexion TCP peut se faire entre deux serveurs par exemple).

Ces deux flux se produisent en même temps dans les deux directions **sauf lors de l'établissement de la connexion et de la déconnexion**.

Il s'assure que les paquets sont envoyés dans l'ordre et sans perte. Si un paquet est perdu il le retransmet.

Il permet de **s'adapter au réseau** et il va ajuster ses opérations suivant la connexion pour optimiser le transfert : il ne faut pas trop envoyer sinon le réseau est surchargé ou trop peu envoyer sinon c'est plus lent.

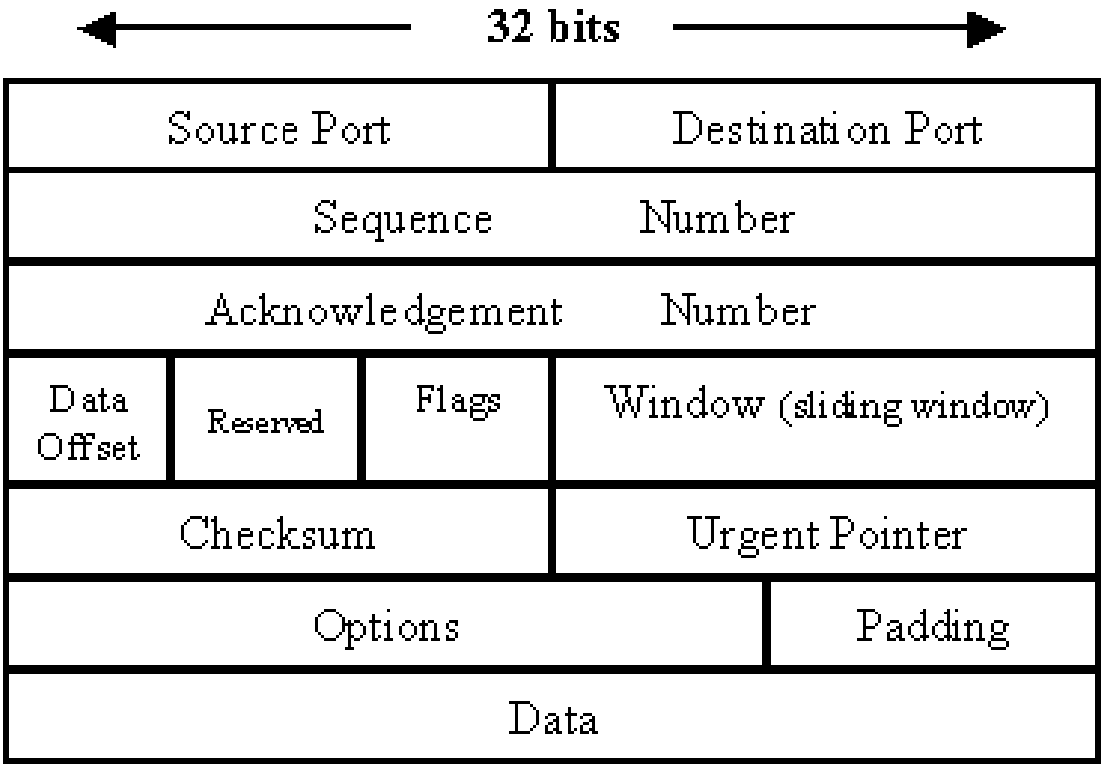
Il permet également de contrôler le flux d'`octets` (`flow control`) en gérant des `buffer` de données en fonction de la vitesse de connexion.

Le connexion TCP se fait en trois temps, elle est appelée `three-way handshake`. Elle permet d'établir les deux numéros de séquences initiaux (`ISN` pour Initial Sequence Number) pour le transfert de données bidirectionnelles (un pour chaque sens).

Les **ISN** sont générés par des algorithmes pour des questions de sécurité. Un `ISN` et tous les numéros de séquence font 32 `bits`.

Ensuite tous les `octets` envoyés par TCP sont numérotés à partir de ces deux `ISN`.

La **structure d'un paquet TCP** est la suivante :



Les **ports** permettent sur une machine donnée, de distinguer différents programmes appelés services. Il s'agit d'un numéro qui est codé sur **16** `bits` donc peut aller de 0 à 2^16 = 65 535.

Lorsque deux machines communiquent ils ont besoin de savoir à quels ports s'adresser.

Par exemple, si vous envoyez une requête HTTP vous ne voulez pas que ce soit l'application mail qui vous réponde, il faut que ce soit l'application serveur Web sur le serveur physique.

Pour cela vous avez besoin de connaître quel port est écouté par l'application serveur Web. Si vous ne le connaissez pas un port par défaut suivant le protocole sera utilisé.

Votre système d'exploitation contient une liste de ports par défaut utilisé dans un fichier `services`.

Par exemple le port par défaut pour le protocole `HTTP` est le `80`.

Celui par défaut pour le protocole `HTTPS` est le `443`.

Pour `SSH` c'est le `22`, pour `SMTP` c'est le `25`, pour le `DNS` c'est le `53` etc

C'est pour cela que vous ne tapez pas `www.google.fr:80` et seulement `www.google.fr`, la requête HTTP / HTTPS sera envoyée par défaut sur le port `80` et les applications Web écoutent toujours ce port pour y répondre.

Port source / port destination : les 16 `bits` identifiants le port de départ et ceux identifiants le port d'arrivé.

Numéro de séquence : il s'agit du numéro de séquence du premier `octet` des `octets` envoyés dans le paquet. Le numéro de séquence du premier paquet commence à `ISN + 1`.

Numéro d'acquittement : numéro de séquence du prochain octet attendu par l'envoyeur du paquet.

Somme de contrôle : somme de contrôle calculée sur l'ensemble de l'en-tête TCP et des données permettant de ‘assurer de l'intégrité du paquet.

Vous n'avez pas besoin de connaître le reste.

Grâce aux **numéros de séquence et d'acquittement**, les destinataires des flux peuvent remettre les paquets reçus dans l'ordre. Les émetteurs peuvent s'assurer que leurs paquets ont bien été reçus et les renvoyer dans le cas contraire.

Vous comprenez maintenant pourquoi le protocole `TCP` est parfaitement adapté pour le Web. Il permet de s'assurer qu'aucun `octet` n'est perdu. Ce serait embêtant si il manquait des lettres dans nos textes, non ?

Le protocole `UDP`

Ok, le protocole `TCP` est génial mais il est gourmand en performances avec la taille de ses `headers`, les processus de connexion et déconnexion (les `handshakes`) etc.

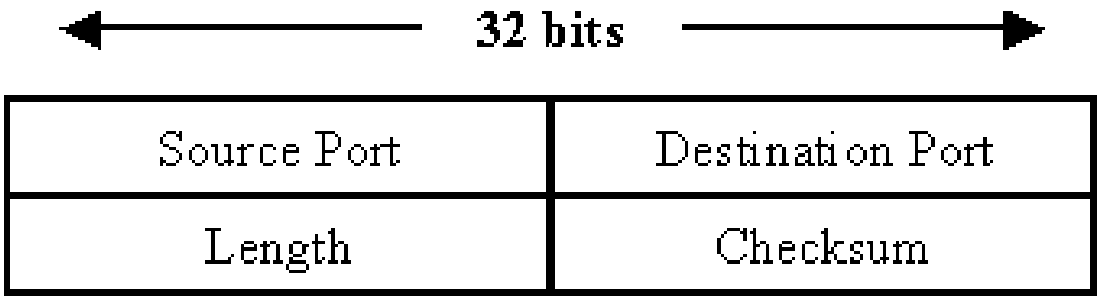
Le protocole `UDP` pour User Datagram Protocol est beaucoup plus simple.

Le rôle de ce protocole est de permettre la transmission de données de manière très simple entre deux entités.

Cependant seule l'intégrité des messages est garantie. Il n'existe pas de garantie sur la livraison ni sur l'ordre d'arrivée.

C'est un protocole sans connexion : on envoi directement un flux unidirectionnel sur une `IP` et un port.

La structure d'un message, appelé datagram UDP est beaucoup plus simple :



Ce protocole est idéale pour les applications réseaux dont la latence est critique : comme les jeux, les communications vidéos et audios.

elles peuvent souffrir des pertes de paquets car elles ont des programmes permettant de gérer ces pertes le mieux possible. En revanche, elles ne peuvent pas attendre qu'un paquet soit renvoyé si il est perdu cela prendrait trop de temps.

Le meilleur des deux mondes : le protocole `QUIC`

Nos amis de Google ont mis au point un nouveau protocole de transport appelé `QUIC` (Quick UDP Internet Connection).

Ce protocole est basé sur le protocole UDP mais ajoute de la sécurité avec des pertes en performance minimes.

Il s'inspire de UDP, TCP et TLS pour créer un nouveau protocole le plus sécurisé et performant possible.

Il reste plusieurs années de développement, mais l' `Internet Engineering Task Force` (IETF, les personnes qui développent les normes Internet) l'a déjà adopté et la prochaine version de `HTTP`, `HTTP/3` utilisera `QUIC` et non plus `TCP` comme `transport layer`.

La couche Internet (`Internet / Network Layer`)

Nous descendons encore d'un niveau pour arriver à la couche Internet avec le protocole `IP` (Internet Protocol).

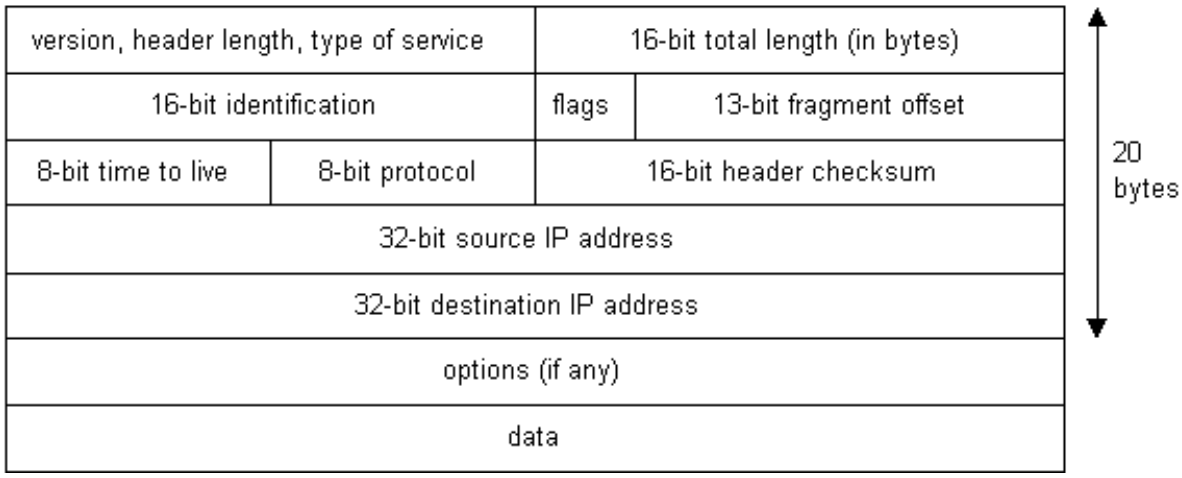
Pour l'instant nous avons parlé des adresses `IP`, il est temps de voir le protocole plus en profondeur.

Le protocole `IP` permet **uniquement de router des paquets entre deux machines sur Internet**.

Il ne s'intéresse pas aux ports, ni au fait qu'un paquet soit perdu, corrompu, ni à l'ordre de réception des paquets.

Il est dit "non fiable" pour ces raisons. Il s'assure juste que les paquets soient envoyés au bon endroit le plus rapidement possible.

Voici à quoi ressemble un `header IPv4` :



Les versions utilisées aujourd'hui sont `IPv4` et `IPv6`.

La différence est que l'adresse IP dans la première version est codée sur 32 `bits` et dans la seconde sur 128 `bits` : tout simplement parce que nous allons manquer d'adresses IP !

En effet, nous avons dépassé 2^32 adresses IP et avec `IPv6` nous pouvons avoir 2^128 adresses donc ça laisse plus de marge !

Version vaut tout simplement `0100` pour IPv4 et `0110` pour IPv6 (tout simplement 4 et 6 en binaire).

Nous retrouvons bien sûr l'adresse IP de l'émetteur et l'adresse IP du receveur.

Nous retrouvons le `protocol` de transport utilisé c'est un `id` codé sur 8 `bits`. Par exemple 6 pour `TCP`.

Nous retrouvons la taille du paquet en `octets` incluant les données et le `header` codé sur 16 `bits` donc un paquet peut avoir un maximum de 2^16 = 65536 `octets`.

Nous avons un `TTL` (Time To Live) qui permet aux paquets perdus sur le réseau d'être supprimés.

Le reste n'est pas intéressant pour vous.

Un paquet `TCP/IP`

Pour modéliser un paquet `TCP/IP` voilà à quoi cela ressemble :