Protocole de signalisation HTTP/2 pour le 5GC

EFORT
http://www.efort.fr

1. Introduction

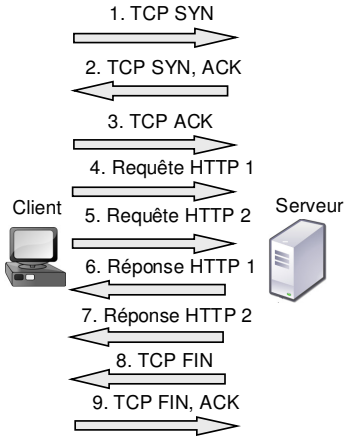
L'introduction du réseau 5G fait apparaître un nouveau protocole de signalisation utilisé par les fonctions du plan contrôle du réseau coeur 5G, à savoir HTTP2 (HyperText Transfer Protocol version 2). HTTP/2 hérite de la sémantique du protocole HTTP/1.1 (méthodes, codes de statut et headers) mais améliore différents aspects du protocole HTTP/1.1 tels que le passage d 'un protocole texte à un protocole binaire, le multiplexage sur une même connexion TCP d 'un grand nombre de requêtes/réponses HTTP avec la priorisation des flux HTTP, la compression des en-têtes HTTP avec la méthode HPACK, etc. L 'échange des données dans les payloads des requêtes et réponses HTTP s 'effectue au format JSON (JavaScript Object Notation).
Le but de ce tutoriel est de décrire brièvement le protocole HTTP/1.1 et présenter le protocole HTTP/2.

2. Protocole HTTP/1.1

HTTP a été inventé par Tim Berners. Cette première version HTTP est connue sous le nom de HTTP/0.9.
En Mai 1996, HTTP/1.0 voit le jour et est décrit dans le RFC 1945.
En Janvier 1997, HTTP/1.1 devient finalement standard de l'IETF. Il est décrit dans le RFC 2068 de l'IETF, puis dans le RFC 2616 en Juin 1999.
En Mars 2012, les travaux à propos de HTTP/2 démarrent et le protocole est spécifié en Mai 2015 dans le RFC 7540.
En Février 2014, la spécification de HTTP 1.1 a été republiée. Elle a été éclatée en plusieurs RFC et corrigée pour toutes ses imprécisions, RFC 7230 à RFC 7237.

HTTP signifie HyperText Transfer Protocol. HTTP est un protocole de communication entre client et serveur pour accéder à un contenu sur le Web. Ce contenu peut être un fichier HTML, des images, des vidéos, etc. Le client identifie la ressource à laquelle il veut accéder via son URL et le serveur lui retourne la représentation de cette ressource.
HTTP est un protocole de la couche application qui utilise TCP pour transporter ses messages (requêtes et réponses). Il utilise les fonctions de ce protocole de transport pour offrir un transfert fiable, ce qui permet à HTTP de se concentrer sur des aspects d'échange et de négociation des types de données échangées. L'une des forces de HTTP est sa simplicité. HTTP est un protocole de requête/réponse sans état via l 'échange de requêtes et de réponses associées. Son ensemble de requêtes et de réponses est réduit, mais accompagné de divers en-têtes qui permettent aux clients et aux serveurs d'échanger des informations supplémentaires sur les données.
Le navigateur est le client HTTP. Il émet des requêtes à un serveur HTTP qui lui retourne les réponses associées. Chaque requête contient une commande spécifique qui peut être suivie de l'adresse du document suivie d'en-têtes optionnels. La réponse du serveur inclut également des en-têtes suivis des données.
Un "client" HTTP est un programme qui établit une connexion à un serveur dans le but de l 'envoi d'une ou plusieurs requêtes HTTP. Un "serveur" HTTP est un programme qui accepte les connexions afin de servir les requêtes HTTP en retournant des réponses HTTP. Le terme "serveur origine" désigne le programme pouvant générer des réponses faisant autorité pour une ressource cible donnée.

Le port TCP par défaut côté serveur est 80 quand HTPP est directement utilisé sur TCP et 448 quand http est utilisé sur TLS sur TCP. La figure montre le modèle d'interaction utilisé par HTTP.
Les pages WEB complexes se composent de nombreux objets (graphique, vidéo, audio) distincts, et le client ne se voit retourner que la page WEB ascii contenant les URLs des objets à obtenir.
Le client doit alors émettre une demande HTTP distincte pour obtenir chacun de ces objets. Une page WEB contient des dizaines d'objets graphiques.
La version 1.1 du protocole HTTP utilisée actuellement sur le Web offre la persistance de la connexion TCP. La persistance permet à un client de continuer à utiliser une connexion TCP existante après que sa demande http initiale ait été traitée par le serveur. Le client émet



simplement une nouvelle demande sur la même connexion. La figure 1 décrit ce fonctionnement.

Figure 1 : Persistance et traitement en pipeline

La persistance autorise une autre fonctionnalité HTTP qui améliore les performances : le traitement en pipeline. Avec le traitement en pipeline, un client n 'a pas à attendre une réponse à une demande avant de délivrer une nouvelle demande sur la connexion. Il peut faire suivre la première demande immédiatement par une seconde demande, ce qui est montré sur la figure 2.

2.1. Requêtes et réponses HTTP

Il existe deux types de messages HTTP : les requêtes et les réponses. Les deux sont de format similaire. Ils sont constitués d'une suite de lignes contenant des caractères en ASCII.

Une requête HTTP est formée d'une suite de lignes composées comme suit :
- une ligne de requêtes contenant la commande de la requête appelée Méthode, l'URL de la ressource demandée et la version de HTTP utilisée;
- zéro, une ou plusieurs lignes d'en-tête;
- une ligne vide;
- le corps de la requête (body de la requête).

Chaque ligne doit se terminer par le caractère « retour chariot » (CR ou Carriage Return) suivi de « Retour à la ligne » (LF ou line feed). Les champs dans chaque ligne sont séparés par un ou plusieurs espacements.

METHODE URL VERSION<*crlf*>
EN-TETE : Valeur<*crlf*>
.
.

.
EN-TETE : Valeur*<crlf>*
Ligne vide<crlf>
CORPS DE LA REQUETE

Le format d'une ligne de requêtes est le suivant :
<Méthode> <URL> <Version de HTTP>
L'URL désigne la ressource. La version correspond au numéro de version du protocole HTTP utilisée. Actuellement, la plupart des serveurs et des navigateurs comprennent la version 1.1. On spécifie la version sous la forme HTTP/1.1. La méthode indique le type d'opération que l'on veut effectuer sur la ressource.

À chaque requête HTTP, le serveur répond par un message. Ces réponses sont formées d'une suite de lignes composées comme suit :

- une ligne de code d'état contenant la version de HTTP utilisée, un nombre qui indique le type de la réponse et un texte descriptif de ce code;
- zéro, une ou plusieurs lignes d'en-tête;
- une ligne vide;
- le corps de la réponse (body de la réponse).

VERSION-HTTP CODE EXPLICATION*<crlf>*
EN-TETE : Valeur*<crlf>*
.
.
.
EN-TETE : Valeur*<crlf>*
Ligne vide<crlf>
CORPS DE LA REPONSE

Il existe huit méthodes (ou requêtes) HTTP :

- GET. C'est la méthode la plus couramment employée. Le client demande au serveur de lui retourner la représentation d'une ressource identifiée par son URL.
- HEAD. Semblable à GET dans sa syntaxe, mais ici le client ne demande que les en-têtes de la réponse du serveur (informations sur la représentation de la ressource qui aurait été retournée, mais pas la représentation de la ressource elle-même). Cette méthode peut être utilisée pour vérifier qu'une ressource est disponible, pour connaître la taille de la représentation de la ressource, la date de dernière modification de la ressource, etc. sans pour autant retourner la représentation de la ressource.
- POST. La méthode POST permet de créer une ressource et de lui associer une représentation présente dans le body. La méthode POST permet aussi comme la méthode GET d'obtenir la représentation d'une ressource. La différence réside dans le fait que la requête POST ajoute des données supplémentaires (e.g., paramètres de requête tels que des critères de recherche) dans le corps de la requête POST, alors que la requête GET inclue toutes les données requises dans l'URL.
- PUT. La méthode PUT remplace toute la représentation de la ressource par la nouvelle représentation contenue dans le body de la requête.
- PATCH. La méthode PATCH est utilisée pour appliquer des modifications partielles à une ressource (modification d'une partie de la représentation de la ressource par celle présente dans le body de la requête).
- DELETE. Le client demande la suppression d'une ressource.
- TRACE. Demande au serveur de renvoyer la requête qu'il reçoit. Cette méthode permet au client de déterminer les modifications éventuelles appliquées à sa requête par les serveurs intermédiaires par lesquels elle est passée. C'est l'équivalent d'un traceroute au niveau HTTP.
- OPTIONS. Le client demande au serveur de lui retourner la liste des méthodes supportées par le serveur ou par une ressource donnée.

- CONNECT. Demande la conversion de la connexion HTTP en un tunnel TCP/IP. Cette méthode est principalement utilisée pour établir des communications chiffrées à l'aide du protocole TLS (HTTPS) lorsqu'un proxy transparent est utilisé.

Les codes de réponse ou de statut HTTP sont divisés en cinq classes différentes toutes déterminées par les premiers chiffres de leurs codes. Le statut 200 correspond à la classe 2xx tandis que le statut 404 correspond à la classe 4xx. Cette répartition repose sur la signification et la fonction des codes de statut. Les classes définies sont les suivantes :

- Classe 1xx, codes d'information : si le code commence par 1, le serveur indique au client que la requête actuelle est en cours de traitement. Un client doit être en mesure de recevoir plusieurs réponses provisoires 1xx avant de recevoir une réponse finale appartenant aux autres classes.
- Classe 2xx, codes de succès : un code qui commence par 2 indique que la requête a abouti. Elle a été reçue par le serveur, comprise et acceptée. En conséquence, les codes 2xx du serveur sont retournés en même temps que les informations des pages Web désirées.
- Classe 3xx, codes de redirection : un code 3xx décrit des cas de déplacement de la ressource demandée. La réponse contient la nouvelle URL à considérer pour accéder à la ressource.
- Classe 4xx, erreurs du client : le code 4xx renvoie à une erreur commise par le client. Le serveur a reçu la requête mais ne peut pas l'exécuter. En général, il s'agit d'erreur de syntaxe..
- Classe 5xx, erreurs du serveur : Le code 5xx décrit des situations de problèmes liés au fonctionnement du serveur.

2.2. Headers HTTP

Parmi les en-têtes associés aux requêtes figurent les suivants :

- Accept : Peut être utilisé pour spécifier les types de médias acceptables par le navigateur. Il indique que la requête est limitée à un ensemble restreint de types de médias, par exemple les images JPEG et GIF, que le navigateur peut lui-même interpréter sans recourir à des applications externes.
- Accept-Charset : Indique l'ensemble des caractères acceptables par le navigateur. Il spécifie également le codage utilisé pour ces caractères, tel que le codage ISO-Latin (nommé ISO-8859-1).
- Accept-Encoding : Spécifie les types de codages acceptables par le navigateur. Il permet de spécifier, par exemple, que le navigateur accepte des données compressées (par exemple utilisant un codage tel que .zip).
- Accept-Language : Spécifie les langues (français, etc.) que le navigateur accepte (il s'agit d'une préférence).
- Autorization : Permet au client de soumettre des paramètres d'authentification pour une autorisation d'accès au serveur.
- Connection : Permet au client d 'indiquer aux proxys tout en-tête dans le message qui ne doit pas être relayé. Permet aussi au client de demander la persistance ou non de la connexion TCP.
- Content-Encoding : Spécifie le type de codage du body ou corps de la requête (exemple : le body de la requête est compressé au format .zip). Les requêtes POST, PUT et PATCH disposent toujours d'un body.
- Content-Length : Indique la longueur en octets des données contenues dans le body de la réquête.
- Content-Type : Indique le type de contenu de la requête. Il s'agit du type MIME utilisé, e.g., text/html.
- Host : Contient l'adresse du serveur destination.
- User-Agent : Donne des informations (nom, version, etc.) sur le logiciel navigateur utilisé.

- Via : Ajoutés par des proxies, et peuvent apparaître parmi les en-têtes de requête et les en-têtes de réponse.

Parmi les en-têtes associés aux réponses figurent les suivants :

- Age : Transporte l 'âge estimé de la réponse lorsque celle-ci est obtenue d 'une mémoire cache.
- Allow : Identifie les méthodes http qu 'une ressource supporte. Ce header est présent dans la réponse à une requête OPTIONS.
- Cache-Control : Spécifie des directives pour les mécanismes de mise en cache dans les requêtes et les réponses.
- Content-Encoding : Spécifie le type de codage du contenu de la réponse (exemple, : le body de la réponse est compressé au format .zip).
- Content-Language : Décrit la langue (français, anglais, etc.) utilisée dans le document inclus dans la réponse.
- Content-Length : Indique la longueur en octets des données contenues dans la réponse.
- Content-Type : Indique le type de contenu de la réponse, e.g., text/html
- Last-Modified : Le champ d'en-tête Last-Modified indique la date et l'heure à laquelle le serveur estime que la ressource a été modifiée pour la dernière fois.
- Location : Permet au serveur de demander au client de rediriger sa requête ailleurs.
- Retry-After : Le champ d'en-tête de réponse Retry-After peut être utilisé avec une réponse 503 (Service unavailable) pour indiquer pendant combien de temps le service va être indisponible pour le demandeur. Ce champ peut aussi être utilisé avec toute réponse 3xx (Redirection) pour indiquer la durée minimum pendant laquelle le demandeur va devoir attendre avant de rediriger sa demande. La valeur de ce champ peut être une date ou un nombre entier de secondes.
- Server : Donne des informations (nom, version, etc.) sur le logiciel serveur (par exemple, Apache, IIS) utilisé.
- WWW-Authenticate : Indique le schéma et les paramètres d 'authentification applicables au serveur pour la ressource cible

Exemple de requête :

```
GET /images/lapin.jpg HTTP/1.1
Accept: image/jpeg, text/html,*/*
Accept-Language: fr
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; fr) AppleWebKit/418.9.1 (KHTML, like Gecko)
      Safari/419.3
Connection: keep-alive
Host: www.ft.com
```

Exemple de réponse :

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length : 110

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML>
<HEAD>
<TITLE>Lapin</TITLE>
</HEAD>
<BODY>
Ce matin un lapin a tué un chasseur...
</BODY>
</HTML>
```

3. Protocole HTTP/2

HTTP/2 apporte son lot d'améliorations, parmi lesquelles (RFC 7540):

- Connexion unique : Une connexion persistante est ouverte entre le client et le serveur. Le nombre de connexions TCP à établir est ainsi considérablement réduit. A la place d'une connexion par ressource, il s 'agit ici d 'une connexion par hôte.
- Multiplexage : Permet de gérer au sein d'une même connexion des flux concurrents qui ne se bloquent pas entre-eux.
- Priorisation : Maintenant que les requêtes ne sont plus envoyées les unes à la suite des autres, il faut définir l'ordre dans lequel elles seront traitées. Les ressources peuvent désormais être accompagnées d'un poids allant de 1 à 256 qui aident le serveur à savoir ce qu'il doit traiter en premier.
- Server Push : Le server push consiste à utiliser le fait que la connexion entre le client et le serveur est persistante pour envoyer les ressources avant même qu'elles aient été demandées. Cela complémente le schéma classique dans lequel le client initie l'échange et le serveur lui répond.
- Compression de l'en-tête : Les en-têtes HTTP 1.1 sont verbeux, souvent redondants d'une requête à l'autre et ne peuvent pas être compressés. HTTP/2 utilise HPACK pour compresser les en-têtes ce qui réduit leur taille et améliore donc les temps de chargement
- Encodage : HTTP/2 est un protocole binaire versus protocole texte pour HTTP 1.1.

HTTP/2 retient la sémantique de HTTP/1.1 incluant :

- les méthodes HTTP,
- les codes de statut (status codes),
- URIs,
- les champs d 'en-tête (headers fields)

3.1. Couche de tramage binaire HTTP/2

La nouvelle couche de tramage binaire, qui détermine comment les messages HTTP sont encapsulés et transférés entre le client et le serveur, est au cœur de toutes les améliorations de HTTP 2.0 en termes de performances.
La «couche» fait référence à un choix de conception pour introduire un nouveau mécanisme entre l 'interface de socket et l'API HTTP supérieure exposée aux applications: la sémantique HTTP, tels que les verbes, les méthodes et les en-têtes ne sont pas affectés, mais la façon dont ils sont codés lorsqu'ils transitent est ce qui est différent. À la différence du protocole HTTP 1.1 qui est un protocole en texte brut, toutes les communications HTTP/2 sont décomposées en messages et trames plus petits, chacun d'eux est codé en format binaire.

Par conséquent, le client et le serveur doivent utiliser le nouveau mécanisme de codage binaire pour se comprendre : un client HTTP 1.x ne comprend pas un serveur qui ne supporte que le protocole HTTP/2 et vice versa.

Figure 2 : Trames HTTP/2

Le protocole HTTP/2 est binaire pour rendre le découpage (framing) 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 HTTP/1.

Alors que HTTP/1 utilise la ligne de requête pour transmettre le nom de la méthode (e.g., POST) et l'URI cible, et la ligne de code de statut pour la réponse, HTTP/2 utilise des champs Pseudo-Header spéciaux commençant par ':' à cet effet.
Les champs pseudo-Header ne sont pas des champs d'en-tête HTTP. Les endpoints ne doivent pas générer des champs Pseudo-Headers autres que ceux définis par le standard. Il existe des Pseudo-Headers qui ne peuvent être présents que dans les requêtes et d 'autres Pseudo-Headers pour les réponses uniquement.
Tous les Pseudo-Headers doivent apparaître dans les blocs d 'en-tête avec les en-tête traditionnels et transportés dans les trames HEADERS.
Les champs Pseudo-Header dans les requêtes sont :
":method" : Il s 'agit de la méthode HTTP
":scheme" : Inclut la portion schéma de l 'URI cible, e.g., "http", "https", etc.
":authority" : Inclut la portion authority de l 'URI cible. Il remplace le header Host HTTP/1.1.
":path" : Inclut le chemin et paramètres de requête.
Les champs Pseudo-Header dans les réponses sont :
":status" : Inclut le code de statut.

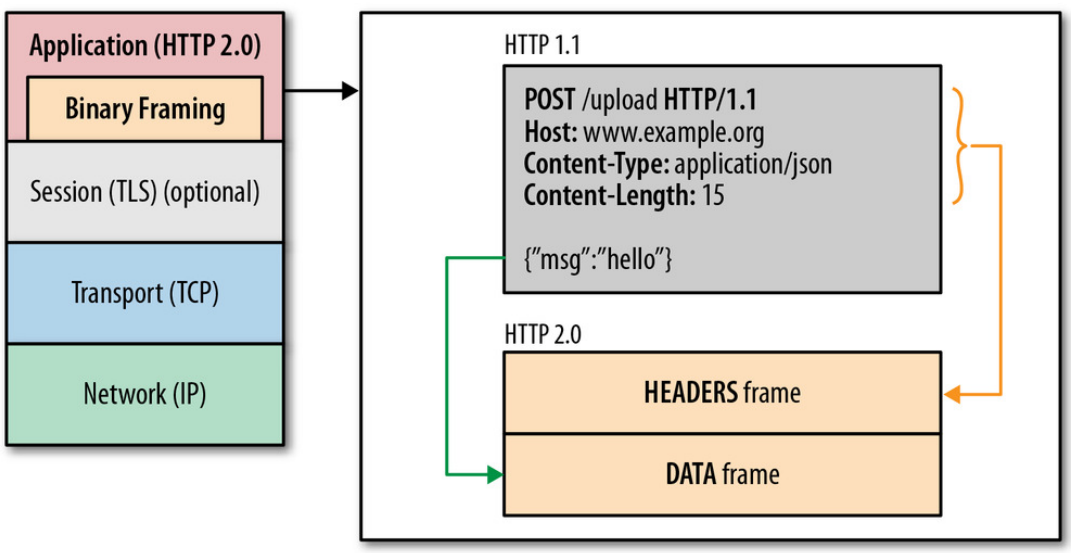


Figure 2 : Principe de tramage HTTP/2

Toutes les requêtes HTTP/2 doivent inclure exactement une valeur valide pour les champs Pseudo-Header ":method", ":scheme", et ":path" à moins qu'il s 'agisse d 'une requête CONNECT
Toutes les réponses HTTP doivent inclure exactement une valeur valide pour le champ Pseudo-Header ":status".
HTTP/2 ne définit pas de manière de transporter la version qui est incluse dans la ligne de
Une requête HTTP voit sa ligne de requête et ses headers transportés dans une trame HEADERS et son éventuel body dans une ou plusieurs trame DATA selon la taille maximum autorisée pour la trame DATA (Figure 2).
Une réponse HTTP voit sa ligne de statut et ses headers transportés dans une trame HEADERS et son éventuel body dans une ou plusieurs trame DATA selon la taille maximum autorisée pour la trame DATA.

3.2. Stream, Message et Trame

L'introduction du nouveau mécanisme de trame binaire modifie la manière dont les données sont échangées entre le client et le serveur. Pour décrire ce processus, il est nécessaire d 'introduire une nouvelle terminologie HTTP 2.0:
Stream : Flux d'octets bidirectionnel au sein d'une connexion établie.
Message : Une séquence complète de trames qui se traduisent en un message logique.
Trame : La plus petite unité de communication dans HTTP 2.0, contenant chacune un en-tête de trame, qui au minimum identifie le stream auquel appartient la trame.
Toutes les communications HTTP 2.0 sont effectuées au sein d'une connexion pouvant transporter n 'importe quel nombre de flux bidirectionnels. À leur tour, chaque flux communique en messages, qui consistent en une ou plusieurs trames, chacune pouvant être entrelacées, puis réassemblées via l'identifiant de stream incorporé dans l'en-tête de chaque trame.

Toutes les trames HTTP 2.0 utilisent un encodage binaire, et les données d 'en-tête sont compressées. La figure ci-dessus illustre la relation entre streams, messages et trames, et non pas leur encodage sur le lien de transmission.
La terminologie Streams, Messages et Trames est une connaissance essentielle pour comprendre HTTP 2.0 :
• Toutes les communications sont réalisées sur une seule connexion TCP.
• Le stream est un canal virtuel dans une connexion TCP qui transporte des messages de manière bidirectionnelle. Chaque stream a un identifiant unique sous forme d 'entier (1, 2, …, N).
• Le message est un message HTTP logique, tel qu'une requête, une réponse, qui consiste en une ou plusieurs trames (au moins une trame HEADERS et 0, 1 ou plusieurs trames DATA).
• La trame est la plus petite unité de communication qui transporte un type spécifique de données, e.g., en-tête HTTP, données utiles (payload), etc.
En bref, HTTP 2.0 décompose le protocole de communication HTTP en des petites trames individuelles, qui constituent des messages dans un stream logique. Plusieurs streams peuvent échanger des messages en parallèle sur une même connexion TCP.

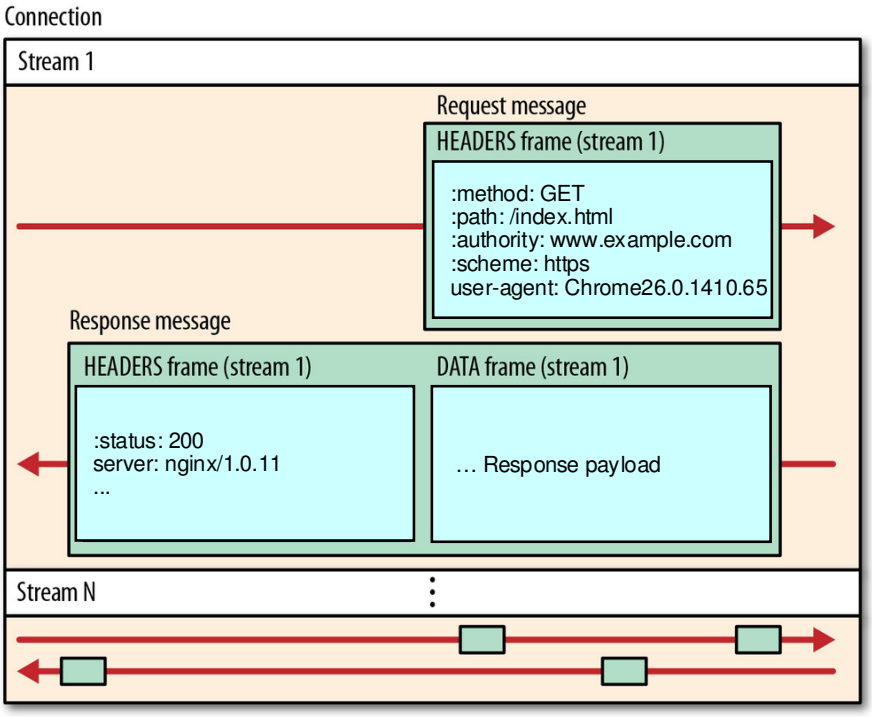


Figure 3: Multiplexage de requête et réponse HTTP/2 sur une connexion TCP

3.2.1. Stream

Les streams ont plusieurs caractéristiques importantes :

- Une seule connexion HTTP / 2 peut contenir plusieurs streams concurrents et ouverts simultanément. Comme le numéro est positionné sur 31 bits, 2^{31} streams, soit 2,14 milliards de streams peuvent être utilisés.
- Les streams peuvent être initiés par le client ou le server. Un stream est bidirectionnel et est utilisé pour émettre une requête HTTP/2 et retourner la réponse HTTP/2 associée.
- Les streams peuvent être fermés par l'un ou l'autre des endpoints.
- Une fois un stream consommé, il ne peut plus être réutilisé. Quand tous les streams possibles sur une connexion TCP ont été utilisés, il est nécessaire de fermer la connexion TCP et d'ouvrir une nouvelle connexion TCP pour continuer à échanger des requêtes et des réponses HTTP/2.
- L'ordre dans lequel les trames sont envoyées sur un stream est significatif. Les destinataires traitent les trames dans l'ordre dans lequel elles ont été reçues. En particulier, l'ordre des trames HEADERS et DATA est sémantiquement important.
- Les streams sont identifiés par un entier. Les identifiants de stream sont attribués aux streams par l 'endpoint à l'origine du stream. Le client choisit toujours des numéros de stream impairs. Le serveur sélectionne des numéros de stream pairs lorsque le mécanisme 'server push' est autorisé.
- Un Stream a une priorité avec 256 valeurs possibles.
- Le stream ID 0 (0x0) est utilisé pour les messages de contrôle de connexion. Il ne peut pas être utilisé pour établir un nouveau stream.

3.2.2. Trame SETTINGS

Une fois que la connexion TCP est établie entre peers HTTP/2, une trame SETTINGS doit être émise dans chaque direction par ces peers sur le stream 0. La trame SETTINGS contient les paramètres de configuration qui affectent la manière avec laquelle les endpoints communiquent, telles que les préférences et les contraintes sur le comportement du peer. Les paramètres présents dans la trame SETTINGS ne sont pas négociés. Ils décrivent les caractéristiques qui sont utilisées par le peer qui les reçoit. Des valeurs par défaut existent pour chaque paramètre négocié.
Des valeurs différentes pour un même paramètre peuvent être négociées par chaque peer. Une trame SETTINGS doit être émise par les deux endpoints au début de la connexion TCP et peut être renvoyée à n 'importe quel autre moment par n 'importe lequel des endpoints pendant la durée de vie de la connexion pour modifier les valeurs des paramètres de configuration choisies auparavant.
Six paramètres peuvent être négociés :
SETTINGS_HEADER_TABLE_SIZE (0x1) : Permet à l 'émetteur d 'indiquer à l 'endpoint distant la taille maximum de la table dynamique HPACK pour le decodage des blocs de header, en octets. La valeur initiale est 4096 octets. L 'encodeur peut choisir n 'importe quelle taille égale ou inférieure à cette valeur.
SETTINGS_ENABLE_PUSH (0x2) : Ce paramètre peut être utilisé pour désactiver le Push de serveur. Un endpoint ne doit pas envoyer de trame PUSH_PROMISE s'il reçoit ce paramètre positionné à 0. La valeur initiale de ce paramètre est 1, ce qui indique que le push de serveur est permis.
SETTINGS_MAX_CONCURRENT_STREAMS (0x3) : Indique le nombre maximum de Streams parallèles que l 'émetteur de la trame permet. Cette limite est directionnelle : elle s'applique au nombre de Streams que l 'émetteur de la trame SETTINGS permet au récepteur d 'initier. Initialement, il n'y a pas de limite à cette valeur. Il est recommandé que cette valeur ne soit pas inférieure à 100, afin de ne pas limiter inutilement le parallélisme.
SETTINGS_INITIAL_WINDOW_SIZE (0x4) : Indique la taille de la fenêtre initiale de l 'émetteur pour le contrôle de flux au niveau du Stream. La valeur initiale est 65 535 octets. Ce paramètre affecte la taille de la fenêtre pour chaque Stream.

SETTINGS_MAX_FRAME_SIZE (0x5) : Indique la taille maximum de payload de trame que l ’émetteur souhaite recevoir, en octets. La valeur initiale est 16384. La valeur soumise par un endpoint doit être entre la valeur initiale soit 16384 et la taille maximum autorisée égale à 16777215 octets. Les valeurs situées en dehors de cette plage doivent être traitées comme une erreur de connexion : PROTOCOL_ERROR.
SETTINGS_MAX_HEADER_LIST_SIZE (0x6): Ce paramètre informe le peer de la taille maximale de la liste d'en-tête que l ’émetteur est prêt à accepter, en octets. La valeur est basée sur la taille non compressée des champs d'en-tête, y compris la longueur du nom et valeur en octets plus une surcharge de 32 octets pour chaque champ d'en-tête.

3.2.3. Multiplexage de requête et réponse HTTP/2 sur une connexion partagée

Avec HTTP 1.x, si le client veut émettre plusieurs demandes parallèles pour améliorer ses performances, alors plusieurs connexions TCP doivent être utilisées. Ce comportement est une conséquence directe du modèle de transfert HTTP 1.x, qui garantit qu'une seule réponse peut être délivrée à un instant donné par connexion (file d'attente de réponse). Cela se traduit également par un blocage en tête de ligne (head-of-line blocking) et une utilisation inefficace de la connexion TCP sous-jacente.
La nouvelle couche de tramage binaire dans HTTP 2.0 supprime ces limites et permet le multiplexage des requêtes et des réponses, en permettant au client et au serveur de décomposer un message HTTP en trames indépendantes, les entrelacer, puis les réassembler à l'autre extrémité.
La figure 4 décrit plusieurs streams dans le même connexion : le client transmet une trame DATA (Stream 5) au serveur, tandis que le serveur transmet une séquence de trames entrelacées au client pour les Streams 1 et 3. Il y a donc trois échanges requête-réponse en parallèle.
Lorsque le serveur a reçu la requête GET sur le stream 1, il commence à la traiter et retourne une réponse 200 OK sous forme d ’une trame HEADERS et trames DATA (le body de la réponse est volumineux et nécessite plusieurs trames DATA). Après avoir envoyé la première trame DATA, il reçoit une 2ème requête GET sur le stream 3. Comme ce stream a une priorité plus élevée que le stream 1, le serveur retourne la réponse de la 2ème requête GET sous forme de trame HEADERS et trame DATA (la réponse peut être encapsulée dans une seule trame DATA). Puis le serveur continue à délivrer les trames DATA supplémentaires sur le stream 1. Enfin le serveur a reçu sur le stream 5 une trame HEADERS et va recevoir la trame DATA relative à une requête POST sur ce même stream 5. De nouveau, si le stream 5 est plus prioritaire que le stream 1, le serveur retournera la réponse de la requête POST avant de continue à retourner les trames DATA supplémentaires sur le stream 1.
La possibilité de décomposer un message HTTP en trames indépendantes, de les entrelacer, puis les réassembler à l'autre extrémité est l'amélioration la plus notable de HTTP 2.0. Cela conduit à de nombreux bénéfices en terme de performance, qui permet de:
• Entrelacer plusieurs requêtes en parallèle sans bloquer aucune
• Entrelacer plusieurs réponses en parallèle sans bloquer aucune
• Utiliser une seule connexion pour transmettre plusieurs requêtes et réponses en parallèle
• Réduire les temps de chargement des pages en éliminant les temps de latence inutiles, etc.
La nouvelle couche de tramage binaire du protocole HTTP 2.0 résout le problème de blocage en tête de ligne présent avec HTTP 1.1 et élimine le besoin de connexions multiples pour permettre le traitement et livraison parallèles des requêtes et des réponses. En conséquence, cela rend l ’application plus rapide, plus simple et moins coûteuse à déployer.

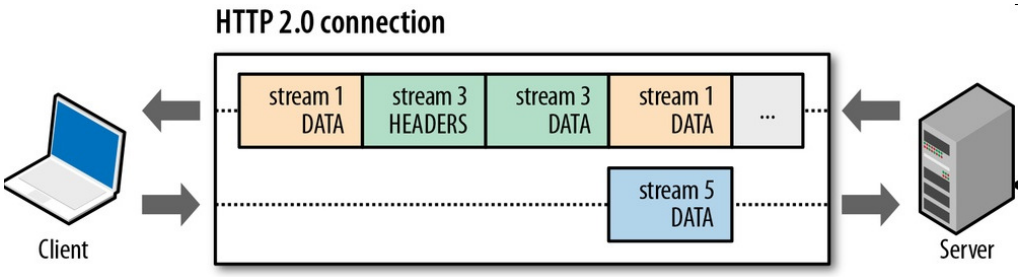


Figure 4 : Principe de multiplexage HTTP/2 sur une même connexion TCP

3.3. Server push

Une nouvelle fonctionnalité puissante de HTTP 2.0 est la capacité du serveur à envoyer plusieurs réponses pour une seule requête du client. C'est-à-dire qu'en plus de la réponse à la demande initiale, le serveur peut pousser (server push) des ressources additionnelles au client, sans que le client ait à demander explicitement chacune.
Pourquoi aurions-nous besoin d'un tel mécanisme ? Une application Web typique consiste en des dizaines de ressources, qui sont toutes découvertes par le client en examinant le document fourni par le serveur. Par conséquent, pourquoi ne pas éliminer la latence supplémentaire et laisser le serveur transmettre les ressources associées, au client à l'avance? Le serveur sait déjà de quelles ressources le client aura besoin; c'est cela même qui caractérise le serveur push. En insérant manuellement la ressource dans le document, on la pousse vers le client, sans attendre que le client la demande. La seule différence avec HTTP 2.0 est qu'il est possible maintenant de déplacer ce traitement hors de l 'application et le transférer dans le protocole HTTP lui-même, qui offre des avantages importants :
• Les ressources envoyées peuvent être mises en cache par le client.
• Les ressources envoyées peuvent être refusées par le client.
• Les ressources envoyées peuvent être réutilisées sur différentes pages.
• Les ressources envoyées peuvent être priorisées par le serveur.

Dans l'exemple à la figure 5, le client envoie une requête GET sur le stream 5. Le serveur retourne une réponse 200 OK contenant un body sous forme HTML qui inclut les URIs de trois objets graphiques que normalement le client doit obtenir via l'envoi de trois requêtes GET supplémentaires au service.
Comme le serveur veut pousser deux des trois objets graphiques, il émet une trame PUSH_ PROMISE par objet graphique à pousser. Cette trame PUSH_PROMISE contient l'URI de l'objet qui sera poussé ainsi que le numéro de stream pair sur lequel l'objet sera poussé (Promised stream). Les trames PUSH-PROMISE doivent être envoyées sur le même stream que la réponse 200 OK et avant de renvoyer la réponse afin d'éliminer les conditions de concurrence avec le client, par exemple le client demandant la même ressource que le serveur est sur le point de pousser. Sur la figure, le serveur retourne donc sur le stream 5 deux trames PUSH_PROMISE et la réponse 200 OK représentée par une trame HEADERS et une trame DATA. Puis le serveur délivre sur les streams 8 et 10 les deux objets promis. Chaque objet promis est envoyé sous forme d'une trame HEADERS et de une ou plusieurs trame DATA selon la taille maximum autorisée pour la trame DATA et la taille de l'objet graphique à envoyer. Le client en analysant la réponse 200 OK retournée sur le stream 5 identifiera dans son body les URIs à obtenir. Il analyse si tous les URIs présents seront poussé par le serveur en identifiant les URIs présents dans les trames PUSH_PROMISE. Si des URIs ne seront pas poussés, c'est au client de les obtenir en utilisant des requêtes GET émises sur des streams impairs.

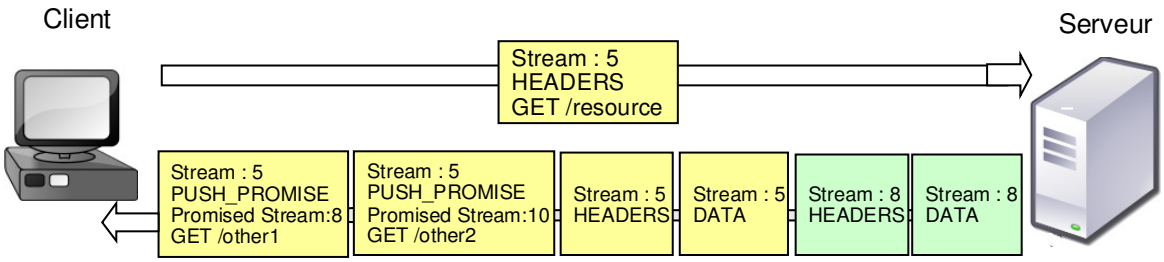


Figure 5 : Mécanisme de « server push »

3.4. Trames HTTP/2

En plus des trames **HEADERS**, **DATA**, **SETTINGS** et **PUSH_PROMISE** décrites plus haut, le protocole HTTP/2 consiste en les trames suivantes :

- La Trame **PRIORITY** permet de modifier à tout moment la priorité d'une stream. Elle est émise sur le stream en question.
- La trame **RST_STREAM** permet de mettre immédiatement fin à un Stream. RST_STREAM est émis sur le stream en question pour demander la fin du Stream. Cette trame indique un code d'erreur.
- La trame **PING** qui doit être acquittée par une trame PING ayant un flag ACK positionné à 1 est un mécanisme permettant de mesurer le temps aller-retour à partir de l 'émetteur de la trame, ainsi que pour déterminer si une connexion http/2 à l 'état de repos est toujours fonctionnelle. Les trames PING peuvent être émises à partir de n 'importe quel endpoint sur le stream 0.
- La trame **GOAWAY** sert à déclencher la fin de la connexion TCP suite) une erreur globale. La trame GOAWAY permet à un endpoint de cesser proprement d'accepter de nouveaux Streams sur une connexion TCP tout en terminant le traitement des Streams précédemment établis. Par exemple, lorsque serveur a consommé tous ses streams pour le mécanisme server push, il émet une trame GOAWAY sur le stream 0. Cela force le client à établir une nouvelle connexion TCP pour continuer les échanges HTTP/2. Par contre le serveur continue à traiter les requêtes déjà reçues sur des streams impairs et retourne les réponse associées.
- La trame **WINDOW_UPDATE** est utilisée pour implanter un contrôle de flux au niveau applicatif indépendamment de celui mis en œuvre au niveau TCP. Ce contrôle de flux s 'applique à deux niveaux : sur chaque Stream individuellement et sur toute la connexion. Si le contrôle de flux concerne un stream particulier, la trame WINDOW_UPDATE, elle est émise sur ce stream. S'il concerne l'ensemble de la connexion, la trame WINDOW_UPDATE est émise sur le stream 0.
- Si une taille maximum est spécifiée pour la trame HEADERS, il est possible que tous les headers d'une requête ou d'une réponse ne puissent pas être encapsulés dans une seule trame HEADERS. Dans ce cas, les premiers headers sont présents dans la trame HEADERS (notamment les pseudo headers) et les autres sont transportés dans une ou plusieurs trames **CONTINUATION**.

Une trame HTTP/2 est la plus petite unité de communication au sein d'une connexion HTTP/2 et comporte un en-tête et une séquence d'octets de longueur variable dont la structure correspond au type de trame.
Les champs de l'en-tête de trame sont définis comme suit (Figure 6):

- Length : La longueur de la charge utile (Payload). Entier de 24 bits. La taille maximum notamment pour une trame DATA peut être limitée par le paramètre SETTINGS_MAX_FRAME_SIZE. Les 9 octets de l'en-tête de trame ne sont pas inclus dans cette valeur.
- Type : Le type de trame codé sur 8 bits. Le type de trame détermine le format et la sémantique de la trame. Les mises en œuvre doivent ignorer et éliminer toute trame dont

le type est inconnu. DATA : 0x0; HEADERS : 0x1; PRIORITY : 0x2; RST_STREAM : 0x3; SETTINGS : 0x4; PUSH_PROMISE : 0x5; PING : 0x6; GOAWAY : 0x7; WINDOW_UPDATE : 0x8; CONTINUATION : 0x9;

- Flags : Un champ de 8 bits est réservé aux flags (drapeaux) booléens spécifiques au type de trame. Les flags ont une sémantique qui dépend du type de trame. Les flags sans sémantique définie pour un type de trame particulier doivent être laissés non-définis (0x0) lors de l'envoi.
- R: Un champ sur 1 bit réservé. La sémantique de ce bit n'est pas définie, le bit ne doit pas être positionné (0x0) lors de l'envoi et doit être ignoré lors de la réception.
- Stream identifier : identificateur de flux exprimé en entier de 31 bits. La valeur 0x0 est réservée aux trames associées à la connexion dans son ensemble à la différence d ’un stream individuel.
- La structure et le contenu de la charge utile (payload) de la trame dépendent entièrement du type de trame.

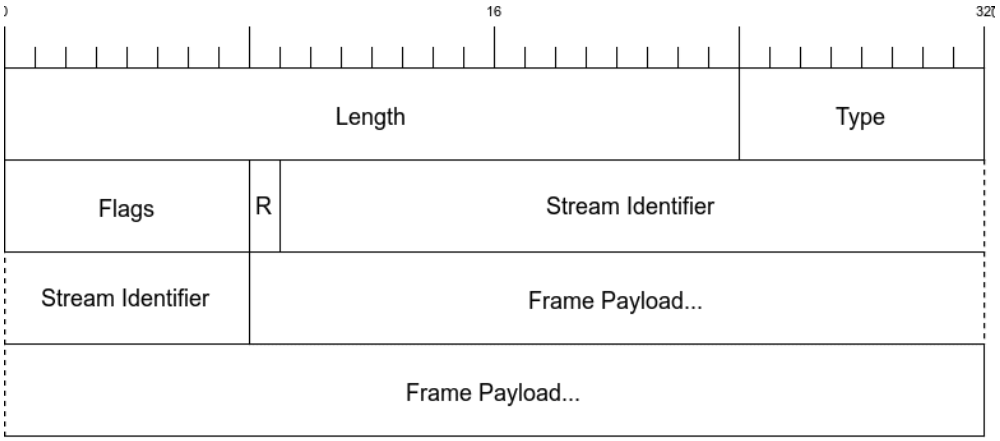


Figure 6 : Format d’une trame HTTP/2

3.5. Compression d’en-tête

Chaque transfert HTTP comporte un ensemble d’en-têtes décrivant la ressource transférée et ses propriétés. Avec HTTP 1.x, ces métadonnées sont toujours envoyées sous forme de texte brut et rajoutent de 500 à 800 octets d ’overhead par demande et des Koctets supplémentaires si les cookies HTTP sont utilisés. Pour réduire cet overhead et ainsi améliorer les performances, HTTP/2 compresse les métadonnées d’en-tête (mécanisme HPACK, RFC 7541):
HPACK utilise trois méthodes de compression :

- Dictionnaire statique: Un dictionnaire prédéfini de 61 en-tête communs, certains avec des valeurs prédéfinies.
- Dictionnaire dynamique: Une liste d ’en-têtes identifiés pendant la connexion. Ce dictionnaire a une taille limitée (par defaut 4096 octets, la 1ère entrée commençant par l ’index 62) et lorsque de nouvelles entrées sont rajoutées, certaines anciennes entrées peuvent être supprimées.
- Encodage Huffman: Un code Huffman statique peut être utilisé afin d ’encoder toute chaîne de caractères : nom ou valeur. Ce code a été calculé pour les requêtes/réponses HTTP. Les digits ascii et les caractères minuscules ont des encodages plus courts avec un taux de compression qui peut atteindre 8:5 (ou 27,5% plus petit).

HPACK ne s’applique qu’au headers présents dans les trames HEADERS et CONTINUATION des requêtes et réponses. HPACK ne s’applique pas au body d’une requête ou d’une réponse présent dans des trames DATA.

La formation EFORT «Réseau de Signalisation HTTP/2 dans le 5GC» décrit l'architecture de signalisation associée au plan contrôle du réseau 5GC et le protocole HTTP/2 utilisé dans ce contexte.

https://www.efort.fr/formations/r%C3%A9seau-de-signalisation-http2-pour-le-r%C3%A9seau-5gc