Etat de l'art du protocole http2

Pascal Berthou, Bertrand Claude, Yannick Derroche, Jean-Philippe Duolle, Samir Medjiah

ACTION R&T: SATELLITE HTTP 2.0
R-S14/TC-0008-024

Etat de l'art du protocole http2

Document n° TSOEF53/2015/0050/BC
Toulouse, le 04/11/2015

Sommaire

Table des figures

1 - INTRODUCTION

L'évolution du protocole http2 est poussée par les acteurs majeurs de l'Internet actuel. Il est de ce fait nécessaire d'identifier dès à présent l'impact sur les échanges au travers des liaisons satellitaires, de manière à ce que les SATCOM continuent leur intégration dans l'écosystème général d'Internet de façon de plus en plus transparente. L'idée générale est de profiter des évolutions technologiques autour d'Internet pour dimensionner et configurer au plus juste les architectures satellite.

1.1 - Présentation du document

Ce document est structuré en 2 chapitres principaux présentant l'état de l'art et les scénarii d'utilisation de http2.

Le premier chapitre dresse un panorama global du protocole HTTP dans ses différentes versions. On y adresse les principes de fonctionnement majeurs en mettant en évidence les différences par rapport aux versions précédentes. On s'attarde sur l'intégration de http2 avec la couche transport, où sont finalement reportés un certain nombre des contraintes liées au satellite. Le protocole HTTP se généralisant dans son utilisation, on définit les cas d'utilisation majeurs. Enfin on explique comment certains principes de fonctionnement peuvent permettre de résoudre des limites posées par HTTP dans ses versions 1.x en particulier dans un environnement satellitaire, voire hybride satellitaire/terrestre.

Sur la base de ce premier niveau d'analyse, le second chapitre explique pourquoi les systèmes actuels et futurs des opérateurs s'appuient sur le satellite, et ont donc besoin de trouver des solutions pour l'intégrer de façon transparente dans leurs offres d'accès. Au final on dégage plusieurs pistes pertinentes pour les scénarii applicatifs qui seront analysés dans la suite de l'étude puis maquettés.

1.2 - Documents applicables et documents de référence

1.2.1 - Documents applicables

DA1 **Règlement de la consultation**

DA2 **Spécification Technique de Besoin - DCT/NT/SU 2014-0017372 Ed.1 Rev.0 - 21/10/2014**

DA3 **Cahier des Clauses Administratives Particulières (CCAP) du CNES du 30 avril 2014**

1.2.2 - Documents de référence

[DR1] http://w3techs.com/technologies/details/ce-http/2/all/all

[DR2] Belshe & al, « Hypertext Transfer Protocol version 2 HTTP/2», RFC 7540, IETF, Mai 2015.

[DR3] R. Peon and W. Chan, « SPDY: An experimental protocol for a faster web », http://dev.chromium.org/spdy/spdy-whitepaper, google white paper.

[DR4] Belshe & al, “SPDY Protocol draft-mbelshe-httpbis-spdy-00”, draft ietf, 2/2012.

[DR5] Salam, A.A.; Luglio, M.; Roseti, C.; Zampognaro, F., "SPDY multiplexing approach on long-latency links," Wireless Communications and Networking Conference (WCNC), 2014 IEEE , vol., no., pp.3450,3455, 6-9 April 2014

[DR6] Salam, A.A.; Luglio, M.; Roseti, C.; Zampognaro, F., "Resource optimization over DVB-RCS satellite links through the use of SPDY," Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), 2014 12th International Symposium on , vol., no., pp.63,69, 12-16 May 2014.

[DR7] Gauthier, Paul; Josh Cohen; Martin Dunsmuir; Charles Perkins (1999-07-28). "Web Proxy Auto-Discovery Protocol (INTERNET-DRAFT)". IETF. Retrieved 2015-02-10

[DR8] S. Loreto & al., “Explicit Trusted Proxy in HTTP/2.0”, draft-loreto-httpbis-trusted-proxy20-01, draft IETF, Feb 2014.

[DR9] Cooley, J.A.; Khazan, R.I.; McVeety, S., "Secure channel establishment in disadvantaged networks : Optimizing TLS using intercepting proxies," in MILITARY COMMUNICATIONS CONFERENCE, 2010 - MILCOM 2010 , vol., no., pp.32-37, Oct. 31 2010-Nov. 3 2010.

[DR10] http://www.deepfield.com

[DR11] http://www.statista.com

[DR12] I. Swett, J. Iyengar, “QUIC Loss Recovery And Congestion Control draft-tsvwg-quic-loss-recovery-00”, draft IETF, Juin 2015.

[DR13] Y. Elkhatib et al., The Effect of Network and Infrastructural Variables on SPDY's Performance: HTTP://arxiv.org/pdf/1401.6508v1.pdf#cite.SPDYWhitepaper

[DR14] Luca Caviglione,_, Alberto Gotta, Characterizing SPDY over High Latency Satellite Channels

[DR15] Matteo Varvello, Kyle Schomp et al., To HTTP/2, or Not To HTTP/2, That Is The Question, http://arxiv.org/pdf/1507.06562.pdf

[DR16] http://yuiblog.com/blog/2007/04/11/performance-research-part-4/

[DR17] https://www.raspberrypi.org/

[DR18] http://beagleboard.org/BLACK

[DR19] http://www.intel.com/content/www/us/en/do-it-yourself/edison.html

[DR20] http://www.etsi.org/technologies-clusters/technologies/m2m

[DR21] http://onem2m.org/

[DR22] ETSI TS 102 921, Machine-to-Machine communications; mIa, dIa and mId interfaces, Décembre 2013.

[DR23] oneM2M TS-0009, HTTP Protocol Binding, Janvier 2015

[DR24] RFC7541, HPACK: Header Compression for HTTP/2, May 2015

[DR25] W3C Web of Things Interest Group, http://www.w3.org/WoT/IG/

[DR26] Y. Cheng, J. Chu & al., "TCP Fast Open", RFC 7413, Decembre 2014.

[DR27] Mueller, C., Lederer, S., Timmerer, C., & Hellwagner, H. (2013, July). Dynamic adaptive streaming over HTTP/2.0. In Multimedia and Expo (ICME), 2013 IEEE International Conference on (pp. 1-6). IEEE.

[DR28] RFC7252, Shelby, Z., Hartke, K., & Bormann, C. (2014). The constrained application protocol (CoAP).

[DR29] Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008, January). MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on (pp. 791-798). IEEE.

[DR30] http://www.connectify.me/blog/taking-google-quic-for-a-test-drive/

1.3 - Glossaire des termes et abréviations

L'ensemble des définitions des termes et des abréviations utilisés dans la proposition est présentée dans le tableau ci-dessous :

Terme	Définition
ADSL	Asymetric Digital Subscriber Line
API	Application Programming Interface
ALPN	Application Layer Protocol Negociation
BER	Bit Error Ratio
CDN	Content Delivery Network
CNES	Centre National d'Etudes Spatiales
CNRS	Centre National de la Recherche Scientifique
COMETES	Collaboration et Organisation pour une Meilleure Entente Transverse des Entreprises du Spatial
CSS	Cascading Style Sheets
cURL	Client URL Request Library
DAMA	Demand Assigned Multiple Access
DASH	Dynamic Adaptive Streaming over HTTP
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DVB-RCS	Digital Video Broadcasting - Return Channel via Satellite
DVB-S	Digital Video Broadcasting – Satellite
DVB-S2	Digital Video Broadcasting – Satellite - Second Generation
ESA	European Space Agency
FEC	Forward Error Correction
FTP	File Transfer Protocol
GEO	Geostationary Earth Orbit
HLS	HTTP Live Streaming
HTML5	HyperText Markup Language version 5
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IETF	Internet Engineering Task Force
IoT	Internet of Things
IIS	Internet Information Services (Microsoft)
ISP	Internet Service Provider
JSON	JavaScript Object Notation
LAAS	Laboratoire d'Analyse et d'Architecture des Systèmes
LFN	Long Fat Network
LTE	Long Term Evolution
M2M	Machine to Machine
Mbps	Megabit per second
NFS	Network File System
NSP	Network Service Provider
OpenSAND	Open Satellite Network Demonstrator
PAC	Proxy Auto-Config
PDU	Protocol Data Unit
PEP	Performance Enhancing Proxy

PHP	PHP: Hypertext Preprocessor
PME	Petites et Moyennes Entreprises
POP	Point Of Presence
QUIC	Quick UDP Internet Connections
REST	Representational State Transfer
RFC	Request For Comments
ROA	Resource Oriented Architecture
RPC	Remote Procedure Call
R&T	Recherche et Technologie
RTP	Real-time Transport Protocol
RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
SAP	Satellite Access Provider
SCTP	Stream Control Transmission Protocol
SPDY	Speedy
SFTP	SSH File Transfer Protocol
SOA	Service Oriented Architectures
SOAP	Simple Object Access Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
SSU	Station Sol Utilisateur
TCP	Transmission Control Protocol
TFO	TCP Fast Open
TLS	Transport Layer Security
VoD	Video on Demand
VoIP	Voice over Internet Procotol
W3Techs	World Wide Web Technology Surveys
WebDAV	Web Distributed Authoring and Versioning
WoT	Web of Things
WPAD	Web Proxy Autodiscovery Protocol
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

2 - CARACTERISTIQUES DU PROTOCOLE HTTP2

Etude : Satellite HTTP 2.0		
Lot 1 / Tâche 1.1 : Etat de l'art		**Resp. LAAS**
Début: T0	Fin: T0 + 2 mois	

Entrées :
- Documents de références sur http2.

Objectifs :
- Expliciter le fonctionnement du nouveau protocole http2
- Implication du nouveau mode de fonctionnement sur les applications traditionnelles et nouvelles.
- Analyse du protocole http2 dans un contexte satellite.

Descriptions des travaux :
Cette sous-tâche est décomposée ainsi :
- Panorama des pratiques actuelles sur Internet : navigation web, applications connectées, streaming vidéo, etc ;
- Protocole http2 : structure du protocole, détail des fonctionnalités, mais aussi sa genèse. ;
- Redéfinition du paysage web par http2 : impact des nouvelles fonctionnalités sur les pratiques actuelles (web socket, SOAP, REST), impact sur les réseaux sous-jacents y compris les réseaux satellites (impact sur les PEP, load balancing).

Sorties :
D1.1 : Premier chapitre du dossier état de l'art :
- Etat de l'art du contexte http2.

2.1 - Introduction

Bien que conçu initialement pour le transport des applications web HTML, le protocole HTTP est aujourd'hui utilisé par un nombre important de services applicatifs : audio, vidéo, données diverses... Son utilisation s'est démocratisée au point d'en faire le protocole de transport applicatif le plus utilisé sur Internet.

Conjointement, les réseaux de communications se sont transformés avec notamment l'entrée en jeu de la mobilité. Les réseaux d'accès se diversifient et parmi eux le satellite a réussi à se positionner comme un acteur viable. Le satellite introduit cependant des spécificités techniques importantes et peu compatibles avec des exigences utilisateurs grandissantes. On peut citer entre autres le délai de propagation important et les pertes élevées comme variables majeures dans les performances finales.

Afin de pallier à ces contraintes, l'utilisation d'accélérateurs de type PEP se révèle indispensable dans les systèmes satellites bidirectionnels. Ces proxy, comme on peut le voir dans le schéma ci-dessous sont principalement utilisés au niveau des passerelles satellites, et mettent en place différentes techniques d'optimisation : caching, compression, connection reaping/multiplexage… Ils sont néanmoins difficiles à maintenir et leur intégration s'avère peu transparente dans les réseaux hybrides terrestres/satellites.

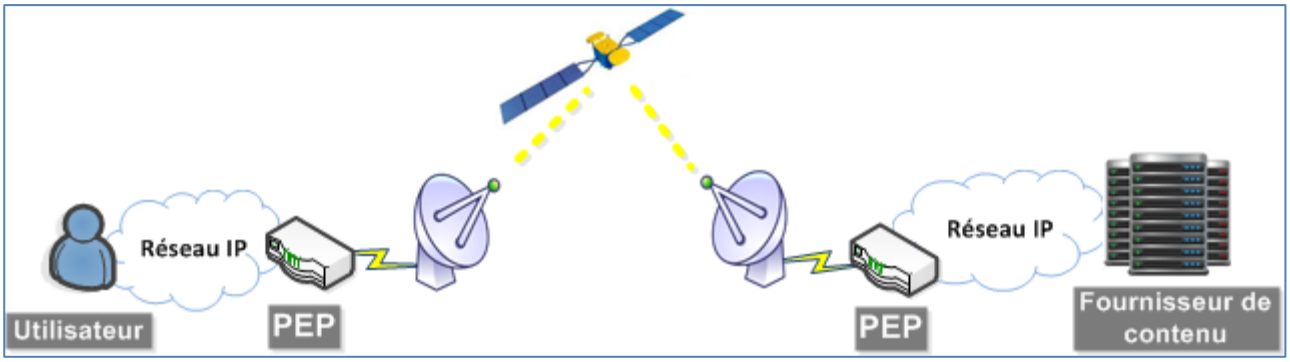

Figure 1 – Utilisation de PEP distribués au niveau des passerelles satellites

L'apparition de nouvelles technologies permettant de circonvenir les limitations de http1.0 et http1.1 offre dans ce contexte des perspectives d'améliorations des systèmes existants. Il est donc important d'identifier les verrous technologiques pouvant disparaître avec l'avènement de http2 et d'envisager le remplacement des PEP actuels par des solutions end-to-end s'appuyant sur des nouveaux procédés.

2.2 - Principes de fonctionnement du protocole http2

La validation de http2 par l'IETF a conclu près de 3 années de travail du groupe de travail httpbis. Formé en 2007, les travaux ont focalisé tout d'abord sur une évolution de http1.1, puis ceux concernant une refonte radicale du protocole HTTP ont débuté en 2012. Aujourd'hui, 6 documents référencés par les RFC 7230 à 7236, définissent le fonctionnement et les extensions du protocole http1.1. La spécification du protocole http2 [DR2] a été acceptée en Mai 2015.

En réalité, pour le développement de http2, les membres du groupe de travail ne sont pas partis d'une feuille blanche, mais se sont très fortement inspirés du protocole SPDY [DR3] [DR4] développé initialement par Google. Ce protocole fut développé et implanté dans le navigateur Chrome de Google et sur un serveur open-source dès 2012. Ainsi, le protocole SPDY est utilisé depuis 2012 dans la plateforme de Google, mais aussi plus récemment par tous les autres navigateurs (Opera, Firefox (mi 2012), Safari, Internet Explorer (fin 2013)).
De nombreuses avancées de SPDY sont de fait présentes dans la spécification de http2, tout en conservant une rétro compatibilité vers http1.1. En 2012 Google estimait SPDY 23% plus rapide sur les réseaux mobiles. Au 1er février 2015, http2, implanté depuis quelques mois (activé par défaut dans Firefox 34 – Décembre 2014), représenterait 5% du trafic web. Aujourd'hui, Google arrête SPDY lancé en 2009, pour une migration officielle à http2.

En près de 20 ans d'existence, HTTP (http1.0 [RFC 1945] dans sa version initiale proposée en 1996), a beaucoup évolué. Seulement trois ans après sa sortie, la version http1.1 est venue la remplacer, avec près de 176 pages de spécifications. Les usages du protocole ont aussi beaucoup évolué. Ainsi, si une page web contenait quelques objets (texte, image) et faisait moins d'une centaine de kilo-octets en 1996, la taille moyenne d'une page web est aujourd'hui de 1,9 Mo, et le nombre moyen d'objets est d'une centaine. La taille moyenne d'une page web a augmenté de 150% en 3 ans. Ainsi, le protocole HTTP a dû évoluer pour s'adapter à ce nouveau contexte.

Historiquement, http1.0 n'autorisait que le transfert d'éléments individuels, successifs, au travers d'une seule connexion TCP. Avec peu d'objets échangés, ce mécanisme suffisait, mais rapidement des solutions d'optimisation ont dû être trouvées. Deux techniques ont été proposées, la première consistant à paralléliser les connexions HTTP au travers de multiples connexions TCP. Ainsi, ont été autorisées entre 6 et 8 connexions TCP parallèles (depuis 2008), soit autant de connexions HTTP, pour échanger les objets. Le deuxième mécanisme, implanté dans http1.1, consiste à autoriser l'émission de plusieurs requêtes HTTP au sein d'une même connexion TCP, sans attendre la réponse des précédentes. Ce mécanisme se nomme pipelining. Ainsi, il est possible, en un seul aller-retour, de requérir de nombreux objets. Bien sûr ces mécanismes peuvent-être combinés pour plus d'efficacité.

Toutefois, ce mécanisme de pipelining n'est pas sans inconvénient. En effet, des chercheurs ont conclu de possibles interactions néfastes entre le pipelining HTTP et l'algorithme de Nagle de TCP. En effet, ce dernier, pour éviter un trop fort overhead d'encapsulation TCP, bufferise les données TCP pendant un certain temps avant de transmettre le segment dans lequel elles sont regroupées, pour avoir suffisamment de données à transmettre. Le prochain segment étant transmis à la réception de l'acquittement du précédent. Ainsi, ce mécanisme de mise en tampon au niveau 4, se rapproche du multiplexage de connexion au niveau 7, et induirait une latence d'un facteur 1,5 par rapport à une version sans Nagle. Il a donc été recommandé de désactiver l'algorithme de Nagle (TCP_NODELAY) quand le pipelining est utilisé. De plus, en cas d'utilisation d'un serveur mandataire, celui-ci doit séquencer les requêtes multiplexées, ce qui est difficile quand un client adresse plusieurs serveurs. De fait, devant toutes ces difficultés, le pipelining http1.1 est désactivé par défaut dans les navigateurs commerciaux actuels. Par contre, le nombre de connexions TCP parallèles est en moyenne de 38 aujourd'hui ! En effet, afin d'accélérer les transferts, les sites web jouent sur différents nom pour un même serveur (par exemple : www1.mondomaine.fr et www2.mondomaine.fr), afin d'autoriser l'ouverture de nouvelles connexions TCP, comme ce pourrait-être le cas pour un nouveau site.

Le nouveau protocole http2 introduit quelques nouveautés. La grande nouveauté est surtout l'adoption d'un format binaire pour les messages échangés. Cette évolution amène principalement une réduction de la taille des headers. En effet, http2 définit des trames pour la structuration de ses messages. La structure d'une trame suit un format précis décrit dans le standard. Une trame commence toujours par un header d'une longueur fixe de 9 octets, suivi du payload d'une longueur variable. Le header comporte les informations suivantes :

- **Length** (24 bits) : un entier non signé pour coder la longueur du payload ;
- **Type** (8 bits) : Type de la trame. Ce qui détermine le format et la sémantique de la trame ;
- **Flags** (8 bits) : Un champ réservé pour positionner des indicateurs concernant la trame ;
- **R** (1 bit) : Un bit réservé, toujours mis à 0x0 ;
- **Stream Identifier** (31 bits) : Un entier non signé pour identifier un flux HTTP. Le flux identifié par l'entier 0x0 est associé à la connexion (identifiant réservé).

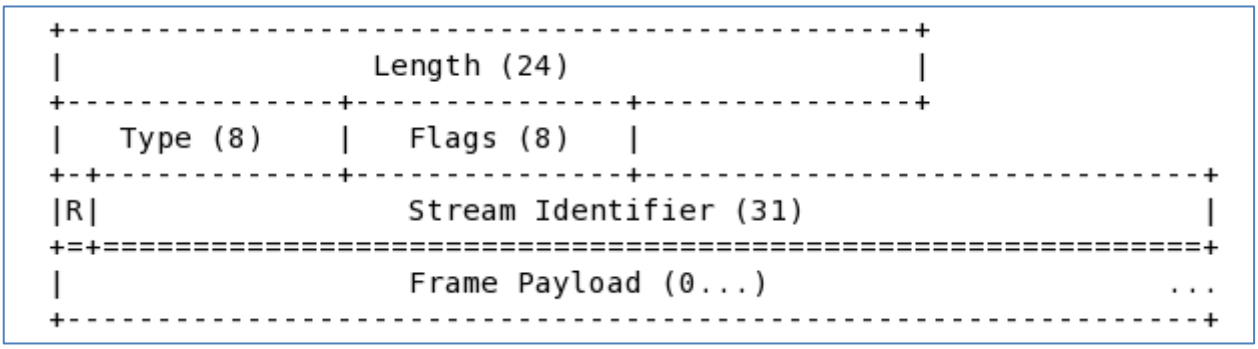

Figure 2 – Format d'une trame http2

Enfin, le contenu et le format du payload dépendent complètement du type de la trame. Parmi les trames définies dans le protocole http2, on peut citer :

- DATA (type =0x0) : sert à transporter des séquences d'octets de langueurs variables. Elle sert par exemple à transporter les payloads des requêtes ou réponses ;
- HEADERS (type =0x1) : sert à ouvrir un flux et transporte également un le bloc d'entêtes ;
- PRIORITY (type =0x2) : sert à spécifier la priorité conseillé par l'émetteur pour un certain flux ;
- RST_STREAM (type =0x3) : sert la terminaison immédiate d'un flux ;
- SETTINGS (type = 0x4) : transporte les paramètres de configuration tels que les préférences et les contraintes du client/serveur. Ces paramètres décrivent les caractéristiques de l'émetteur, qui doivent être utilisés par le récepteur ;
- PUSH_PROMISE (type =0x5) : sert à prévenir le pair distant de flux que le pair local prévoie d'initier ;
- PING (type =0x6) : sert à mesurer le RTT minimal, mais aussi à déterminer si une connexion inactive est toujours fonctionnelle ;
- GOAWAY (type =0x7) : sert à informer le pair distant d'arrêter de créer des flux sur la connexion. Une fois envoyée, l'émetteur ignorera toutes trames envoyées sur de nouveaux flux. Le but de cette trame est de permettre aux entités communicantes d'arrêter d'accepter de nouveaux flux d'une manière « gracieuse » tout en continuant à traiter les flux établis.
- WINDOW_UPDATE (type =0x8) : sert à implémenter un contrôle de flux ;
- CONTINUATION (type =0x9) : sert à continuer une séquence de de fragments d'un bloc d'entêtes.

La notion d'entête reste la même dans http2. Les entêtes sont utilisés que ce soit pour les requêtes ou pour les réponses. Un message http2 peut avoir plusieurs entêtes. Dans http2, la liste des entêtes est compressée et sérialisée au sein d'un block « entête » (header block), qui peut être divisé en un ou plusieurs fragments. A la réception, ces fragments sont réassemblés, et la liste des entêtes est obtenue après décompression. Les fragments d'un bloc d'entêtes doivent être transmis d'une manière contiguë et aucune autre trame d'un autre flux ne doit interrompre la transmission de la liste des trames composant un bloc d'entêtes. La compression des entêtes définie dans http2 est « stateful ». Un contexte de compression et de décompression est maintenu pour toute la connexion. De plus, chaque erreur dans le décodage d'un bloc d'entête doit être traitée comme une erreur de connexion.

Reposant sur une unique connexion entre un client et un serveur, les connexions au sens de http1.x se traduisent maintenant par des flux au sein de l'unique connexion http2. Un flux est une séquence de trames envoyées d'une manière bidirectionnelle au sein d'une connexion http2 entre un client et un serveur. Les flux ont les caractéristiques suivantes :

- Une seule connexion http2 peut contenir plusieurs flux concurrents ouverts ;
- Les flux peuvent être établis et utilisés unilatéralement ou partagés par le client ou le serveur ;
- Les flux peuvent être fermés des deux côtés ;
- L'ordre des trames au sein d'un flux est important ;
- Les flux sont identifiés par un entier. Cet identifiant est attribué par l'entité initiatrice de la connexion (un client utilisera des identifiants impairs, et un serveur utilisera des identifiants pairs, l'identifiant 0 est réservé pour les messages de contrôle de la connexion).

Pour sa gestion, un flux a plusieurs états possibles (inactif, ouvert, fermé, moitié-fermé).
À l'ouverture des flux, le client peut attribuer des priorités à ses flux. Cette priorité peut être modifiée durant la connexion, en utilisant une trame dédiée (trame PRIORITY). La priorisation sert surtout à permettre à un client/serveur d'exprimer à son pair comment ce dernier devrait gérer des flux concurrents. Cette priorité sert donc à choisir les flux pour transmission quand la capacité d'émission est limitée.

L'utilisation des flux pour le multiplexage des données introduit de la contention pour l'utilisation de la connexion TCP sous-jacente. Pour pallier à ce problème, http2 définit un mécanisme de contrôle de flux, utilisé pour les flux individuels tout comme pour toute la connexion. Ce mécanisme de contrôle vise à permettre l'utilisation de plusieurs algorithmes sans nécessiter la modification du protocole. Ainsi le contrôle de flux est caractérisé par les éléments suivants :

- Le contrôle de flux est spécifique à une connexion ;
- Le contrôle de flux se base sur une fenêtre de réception. Un récepteur informe l'émetteur du nombre d'octets qu'il peut recevoir pour un flux ou pour toute la connexion ;
- Tout le contrôle est assuré par le récepteur. L'émetteur doit respecter les limites imposées par le récepteur ;
- La valeur initiale pour la fenêtre de réception est 65535 octets pour les nouveaux flux ou pour la connexion ;
- Le contrôle de flux concerne uniquement les trames de données, et ne s'applique pas pour les autres trames. Ainsi, des trames de contrôle importantes ne peuvent pas être bloquées par le contrôle de flux ;
- Le contrôle de flux ne pas être désactivé.

Le contrôle de flux a été surtout introduit dans le but de protéger les entités communicantes présentant des contraintes de ressources. L'exemple type est le serveur mandataire (proxy) qui peut être amené à gérer plusieurs connexions et qui peut aussi disposer d'une connexion asymétrique (lien montant lent, lien descendant rapide). Le contrôle de flux est alors utile quand le récepteur ne peut plus traiter des données sur un flux, mais peut continuer à traiter d'autres flux au sein de la même connexion.

Parmi les nouveautés importantes introduites par le protocole http2 : l'introduction de l'asynchronisme dans les échanges entre le client et le serveur. En effet, grâce à http2 le serveur peut maintenant envoyer des messages au client sans qu'il y ait une requête préalable du client (push). Ce mode peut s'avérer très utile dans certains cas. Par exemple, un serveur web peut envoyer les éléments constituant une page web (scripts, feuilles de styles, images, etc.) juste après que le client a demandé la page (fichier html). Les communications push trouvent aussi utilisation dans le cadre des applications web où le polling est actuellement la norme. Grâce au mécanisme du push, ces applications scrutant l'état d'une ressource en continu par l'envoi de requêtes périodiques se verront grandement améliorées.

Il n'y a pas de travaux référents aujourd'hui sur le fonctionnement de http2 sur satellite. Toutefois, on peut trouver dans la littérature, quelques articles issus du projet SatNex [DR5] [DR6].

Le premier article évalue sur une plateforme d'émulation satellite le fonctionnement du protocole SPDY et conclut à une amélioration des performances, principalement dû au mécanisme de pipelining. Toutefois, ce résultat est contrasté dans la mesure où l'amélioration n'est effective qu'en cas d'un nombre d'objets conséquents échangés. Sur des petites pages web (mesuré pour 22 objets), la mise en place de la connexion sécurisée TLS (protocole SSL), quasi obligatoire en SPDY et http2, retarde l'arrivée des premiers objets. Avec 640 objets échangés (ce qui est aujourd'hui beaucoup), le gain est effectivement net.

Le second article évalue le comportement de SPDY sur un lien retour de type DVB-RCS avec différentes configurations de DAMA, et conclut de la même manière à des gains du protocole en termes de délais et d'utilisation du lien.

L'utilisation du protocole http2 sur satellite aura un impact important. En effet, le contrôle de flux introduit dans le standard pourra être d'une grande utilité pour améliorer la performance des serveurs proxy, serveurs naturellement présents dans les systèmes de communication par satellite. De plus, le nouveau mode opératoire du protocole http2, spécifiquement l'utilisation d'une seule connexion TCP, le mécanisme push et enfin le codage binaire des messages apporteront un gain certain dans quantités de données transmises par satellite (moins d'octets pour coder les messages, moins de messages, moins de connexions TCP).

2.3 - La sécurité dans http2 : couche de transport TCP-TLS

2.3.1 - Pourquoi TCP-TLS

Historiquement le protocole de la couche transport TCP était le support du protocole applicatif HTTP. TCP est un protocole offrant un service de communication fiable et ordonné sous la forme d'un flux d'octet. Avec la nouvelle version de http qui prône l'utilisation d'une connexion sécurisée, c'est naturellement le couple TCP-TLS qui s'est imposé comme support de transport.

2.3.2 - Principes des échanges

Ce paragraphe rappelle les principes de fonctionnement de TLS et les messages échangés. http2 reposant sur TLS ces détails sont utiles à la compréhension des échanges.

TLS repose sur un cryptage à clés asymétriques (clés publiques et privées). Le serveur, au travers de son certificat envoi sa clé publique au client (certifiée par une autorité de certification pour éviter les attaques de type man in the middle). Avec cette clé, le client va pouvoir crypter une pré-clé qu'il aura généré et l'envoyer de manière sécurisée au serveur. Avec ces informations, pour la session, le serveur et le client génèreront quatre clés qui seront utilisées pour le cryptage des échanges (ces clés ne sont pas échangées). Au terme de cet échange, le client envoi le message CLIENT-FINISHED, chiffré et signé a l'aide des clés ci-dessus. Cela signifie qu'à partir de ce moment, le client communique de cette manière.

Voici comment se passe le dialogue, dans l'ordre chronologique de haut en bas. En vert sont représentés les messages optionnels (par exemple si le certificat sur serveur est déjà connu):

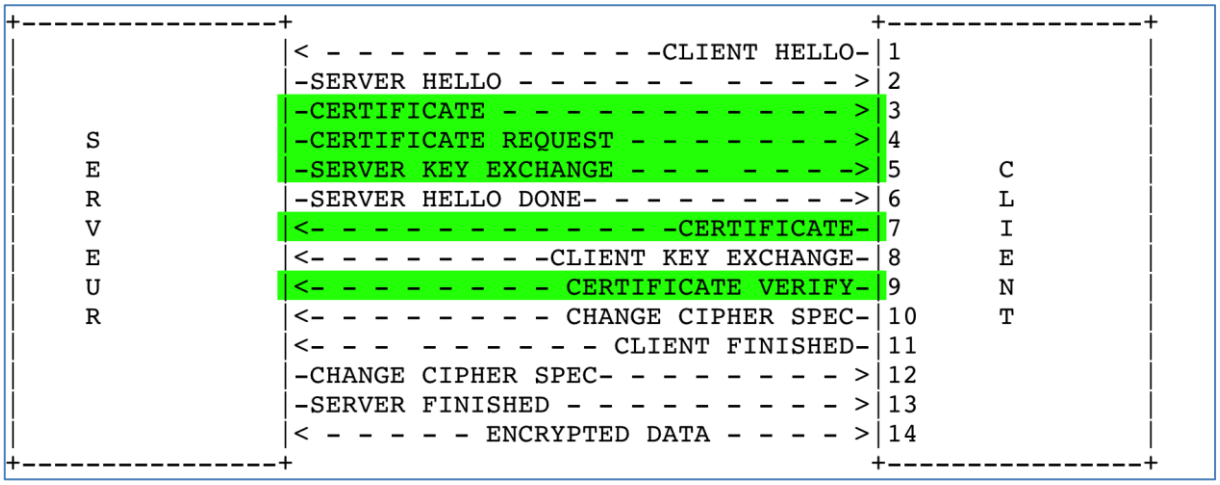

Figure 3 - Etablissement d'une connexion TLS

Voici les explications des messages envoyés :

-> CLIENT HELLO

Le client envoi la version maximale supportée (TLS = 3.1), de la suite d'algorithmes supportés (par ordre de préférence décroissant) et une valeur aléatoire de 32 octets. Le contenu du message est le suivant :

Version - La plus haute version de SSL que puisse utiliser le client.

Random - Un horodatage de 32 bits et une valeur aléatoire de 28 octets générée par le client. Le nombre obtenu va servir la signature des messages.

Session ID - Un nombre, qui identifie la connexion. Un zéro signifie la volonté du client d'établir une nouvelle connexion sur une nouvelle session. Un autre nombre signifie la volonté de changer les paramètres ou de créer une nouvelle connexion sur la session existante.

CipherSuite - Une liste, par ordre décroissant de préférence, des algorithmes que supporte le client. Il s'agit des algorithmes d'échange de clé et de chiffrement.

Compression Method - Liste, par ordre décroissant de préférence, des algorithmes de compression supportés par le client.

Puis le client attend une réponse du serveur :

<- SERVER HELLO

Le serveur effectue le choix de la version, de la suite d'algorithmes (Cipher Suite) et d'une valeur aléatoire. Le message contient :

Version - La plus haute version de SSL que puisse utiliser le client.

Random - Un horodatage de 32 bits et une valeur aléatoire de 28 octets générée par le client.

Session ID - L'Identifiant de la session qui débute.

CipherSuite - La séquence d'algorithmes choisis pour la session. Le serveur sélectionne la première suite qu'il connait dans la liste transmise par le client.

Compression Method - La méthode de compression qui va être utilisée.

Maintenant que les algorithmes sont choisis, le serveur va s'authentifier auprès du client. Il envoie pour cela son ou ses certificats (X.509) au client. Il peut pendant cette étape demander un certificat au client. Le client vérifie l'authenticité du serveur : si cette authenticité est mise en doute, la transaction est interrompue.

*** CERTIFICATE (optionnel)**

Envoi d'une chaine de certificats par le serveur. Le premier certificat est celui du serveur, le dernier est celui de l'autorité de certification

*** CERTIFICATE REQUEST (optionnel)**

Demande un certificat au client pour l'authentifier

*** SERVER KEY EXCHANGE (optionnel)**

Message complémentaire pour l'échange des clés. Ce message contient la clé publique du serveur utilisée par le client pour chiffrer les informations de clé de session

<- SERVER HELLO DONE

Fin des échanges Hello du serveur.

*** CERTIFICATE (optionnel)**

Certificat éventuel du client si le serveur demande une authentification

-> CLIENT KEY EXCHANGE

Le client produit un secret pré-maître (encrypted pre-master key) et le crypte avec la clé publique du certificat du serveur. Ces informations sont chiffrées une deuxième fois avec la clé publique du serveur (et non la clé publique du certificat du serveur) reçue dans le message SERVER KEY EXCHANGE.

A l'aide de cette pré clé, le serveur et le client génèrent quatre clés pour la session :

Server write mac secret - utilisée dans la signature des messages du serveur.

Client write mac secret - pour les messages du client.

Server write key - pour chiffrer les données émises par le serveur.

Client write key - pour le client.

Ces clés ne sont pas échangées. Si besoin est, le serveur vérifie l'authenticité du client.

*** CERTIFICATE VERIFY (optionnel)**

Message contenant une empreinte (hash) signée numériquement et créé à partir des informations de clé et de tous les messages précédents. Ce message permet de confirmer au serveur que le client possède bien la clé privée correspondant au certificat client

-> CHANGE CIPHER SPEC

Passage du client en mode chiffrée avec la clé master comme clé symétrique

-> CLIENT FINISHED

Le client envoie le message CLIENT_FINISHED au serveur. Ce message est chiffré et signé a l'aide des clés ci-dessus. Cela signifie qu' partir de maintenant, le client communique de cette manière.

<- CHANGE CIPHER SPEC

Passage du serveur en mode chiffrée avec la clé master

<- SERVER FINISHED

Confirmation au client du passage en mode chiffré. Ce message est chiffré à l'aide des paramètres de la suite de chiffrement

-> ENCRYPTED DATA

Le tunnel TLS est établit, c'est maintenant le TLS Record Protocol qui prend le relais pour chiffrer les données.

2.3.3 - Les ports et applications utilisant SSL et TLS

Voici la liste des applications exploitant SSL et TLS avec leurs ports TCP associés :

Protocoles : HTTPS, SPDY, HTTP/2
Port TCP : 443
Description : HTTP et SPDY sur TLS

Protocole : DDM-SSL
Port TCP : 448
Description : DDM-SSL

Protocole : SMTPS
Port TCP : 465
Description : SMTP sur SSL et TLS

Protocole : NNTPS
Port TCP : 563
Description : NNTP sur SSL et TLS

Protocole : SSHELL
Port TCP : 614

Description : SSL Shell

Protocole : LDAPS
Port TCP : 636
Description : LDAP sur SSL et TLS

Protocole : FTPS-DATA
Port TCP : 989
Description : FTP données sur SSL et TLS

Protocole : FTPS
Port TCP : 990
Description : FTP controle sur SSL et TLS

Protocole : TELNETS
Port TCP : 992
Description : Telnet sur SSL et TLS

Protocole : IMAPS
Port TCP : 993
Description : IMAP4 sur SSL et TLS

Protocole : IRCS
Port TCP : 994
Description : IRC sur SSL et TLS

Protocole : POP3S
Port TCP : 995
Description : POP3 sur SSL et TLS

2.3.4 - TLS et liens satellites

Quelques travaux ont abordé la problématique de TLS sur lien satellite. Historiquement, les approches de sécurisation des liaisons satellites reposaient en majorité sur des approches de type IPsec (par exemple l'adaptation SatIPsec pour les réseaux par satellite.). http2 change sérieusement la donne en imposant (la norme n'impose pas la sécurisation, mais les implémentations oui !) une sécurisation des connexions. De fait, peu de travaux ont étudié le comportement de TLS sur réseau satellite. [DR9] propose un mécanisme de proxy qui intercepte les ouvertures de connexions TLS en vue de mettre en œuvre un mécanisme de cache pour diminuer le volume de trafic lié aux échanges de certificats sur le lien satellite. Malgré une possibilité de conserver les clés TLS pour une série de connexions TLS, les auteurs montrent que se mécanisme ne réduit que de moitié le volume lié aux échanges de certificats. Le cache de proxy est une bonne idée dans le cas de TLS, bien plus facile à déployer que des adaptations de TLS spécifiques au satellite, mais ne semble pas opportun pour http2 car ce dernier ne réinitialise pas la connexion a chaque requête comme le faisait https.

2.3.5 - http2 et TLS

Le standard n'impose pas TLS pour http2, toutefois c'est la seule option fiable pour déployer un nouveau protocole en présence de nombreux intermédiaires : Proxies, TLS, accélérateurs web, etc … En effet, TLS étant de bout en bout, la connexion ne peut-être interceptée et donc garantit que même si un proxy intermédiaire n'est pas à jour il ne perturbera pas la connexion ! De fait, l'utilisation de TLS et de ALPN (Application Layer Protocol Negociation) est le mécanisme recommandé.

Le client et le serveur négocient le protocole désiré dans les options ALPN des messages TLS handshake (et plus particulièrement les client / server HELLO). De plus, les navigateurs populaires chrome, firefox ont informé qu'ils n'utiliseraient que cette méthode. Ainsi TLS et ALPN sont désormais le standard de négociation du protocole dans les navigateurs. Le choix du protocole http2 se fait donc à la mise en place de la connexion TLS sur le port 443, évitant ainsi des aller retour supplémentaires.

Exemple de message TLS/ALPN :

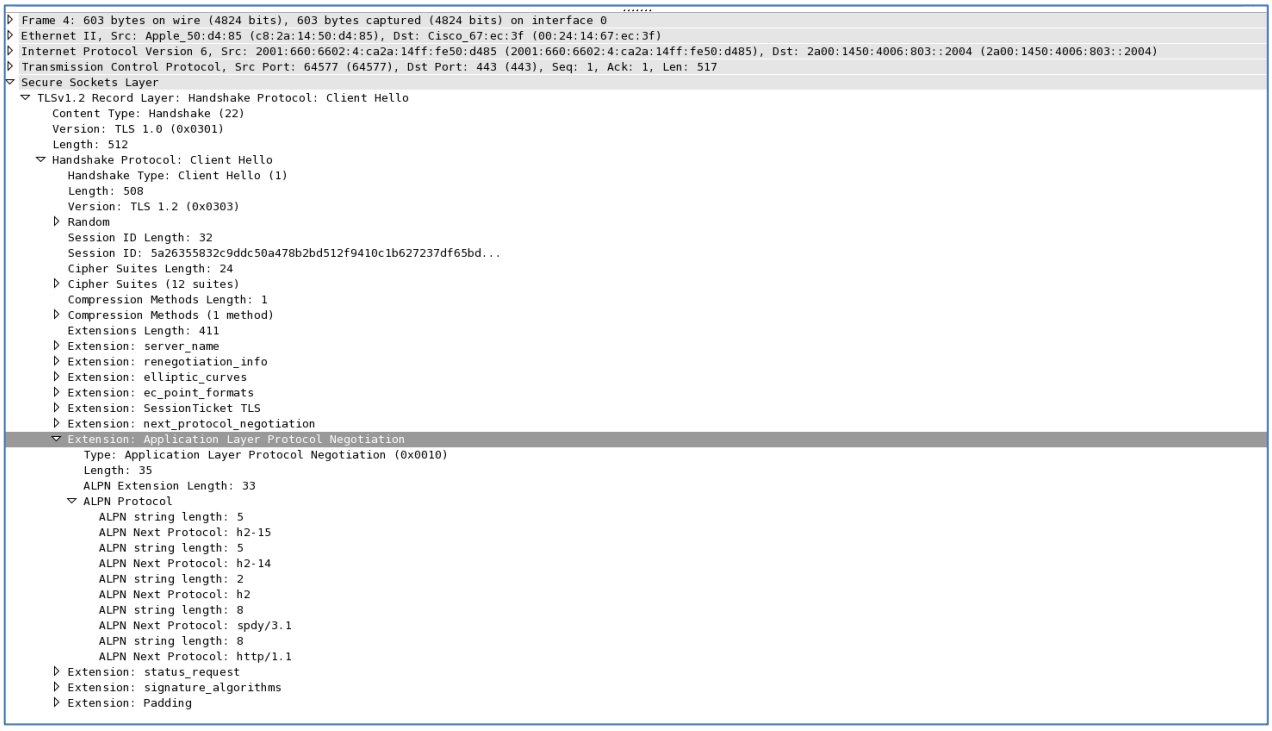

Figure 4 - Message client HELLO

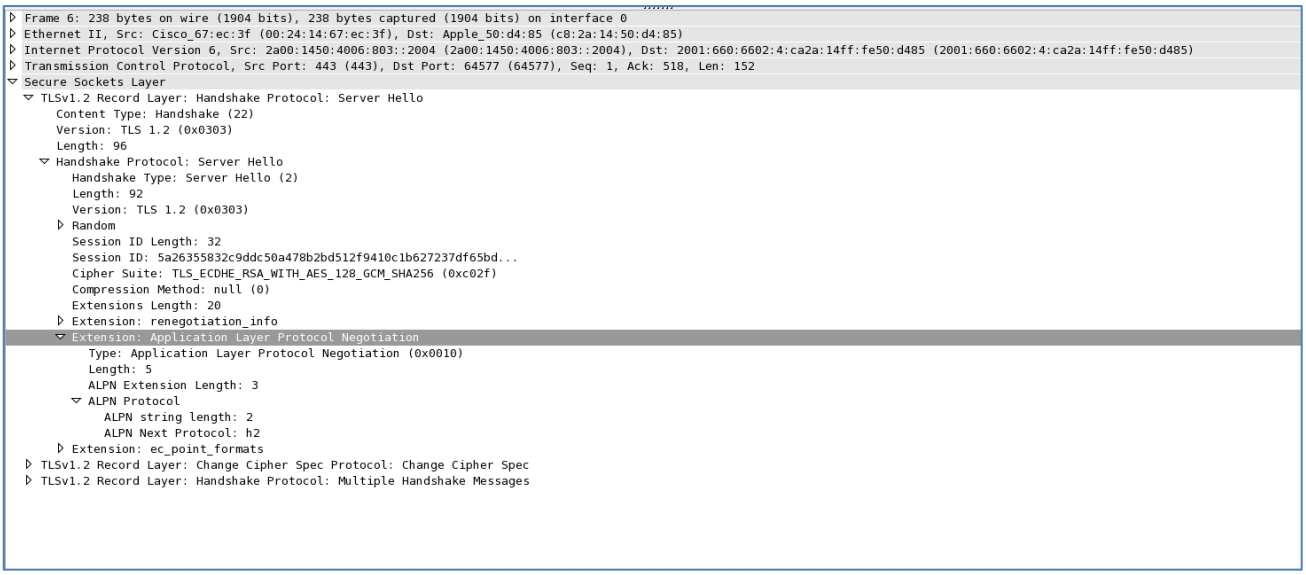

Figure 5 - Message server Hello

Le client annonce une liste de version http supportées : http2-draft 15 (h2-15), h2-14, http2 (h2), spdy puis http/1.1. Le serveur choisit http2 (h2) (voir Figure 4 et Figure 5).

2.4 - QUIC comme alternative à TCP-TLS

2.4.1 - Raison d'être

Comme indiqué précédemment le couple TCP-TLS s'est imposé comme support du protocole http2. Toutefois, sont utilisation est contestée principalement pour deux raisons :

- http2 est un protocole multi-flux, chaque flux étant contrôlé séparément, mais émis dans une seule et unique connexion transport ! Ainsi, si un paquet venait à être perdu dans le réseau, TCP étant fiable et ordonnée, toute la connexion transport serait bloquée, donc l'ensemble des flux http2 aussi, jusqu'à réception du segment manquant (ce propos néanmoins modéré par les mécanismes de SACK de TCP). En cas de perte d'un segment TCP, même si sa perte ne touche qu'un seul stream HTTP, tout le transfert de la page serait ralenti.
- http2 utilise TCP-TLS. TCP nécessite une ouverture de connexion à 3 échanges, TLS à 4 ou 5 échanges ce qui porte à 8 échanges (soit 4 aller-retour) dans le pire cas avant de recevoir la première donnée ! Quand le délai d'aller-retour est ½ seconde, 2 secondes sont nécessaires avant l'ouverture de la page !

Selon une étude sur le comportement des utilisateurs face aux temps de chargement d'une page web, un temps inférieur à 300 ms est parfait, acceptable entre 300 ms et 1 seconde, et rédhibitoire au delà. Ainsi, avec des liens transatlantiques de l'ordre de 100 ms, des temps d'accès aux réseaux 3G du même ordre (100 ms), voire des liens satellite (> 250 ms), un temps de connexion réduit serait souhaitable.

Google, propriétaire de nombreux serveurs web (Google représenterait 25% du trafic internet Américain [DR10]) et développeur du navigateur chrome, qui représente entre 30% et 50% de la part du marché navigateur en 2015, possède tous les atouts pour proposer et tester une solution alternative à TCP-TLS pour la navigation Web. En effet, en déployant sur les serveurs google et dans le navigateur chrome une nouvelle version de la couche transport pout http2, Google génère un trafic significatif et peut espérer la faire adopter à l'IETF, comme cela l'a été avec http2 basé sur SPDY initialement développé par Google.

2.4.2 - Principes du protocole QUIC

QUIC signifie Quick UDP Internet Connections. Les principales bases de ce protocole, spécialement conçu pour servir de support à http2 (ou SPDY plus exactement) sont :

- être un protocole multi-flux
- raccourcir le délai de connexion malgré une connexion cryptée

Ce protocole est donc développé au dessus d'UDP ce qui est la solution la plus simple car implémentable dans l'espace utilisateur du système, donc en tant que composant de l'application ! Pour être accepté par la communauté internet, ce protocole doit être TCP friendly, c'est à dire qu'il ne peut exploiter plus de capacité réseau que ne le ferait une connexion TCP en cas de congestion. QUIC dispose donc d'un contrôle de congestion opéré flux par flux.

La principale caractéristique de QUIC est d'être un protocole sécurisé, au même titre que TLS mais dont la mise en place du cryptage ne nécessite qu'un seul aller-retour dans le cas normal et peut-être direct dans le meilleur cas. Dans le pire cas où le serveur à lui aussi besoin d'authentifier le client, 2 aller-retour sont nécessaires. Dans un contexte satellitaire ou mobile cette propriété est très importante pour améliorer l'interactivité.

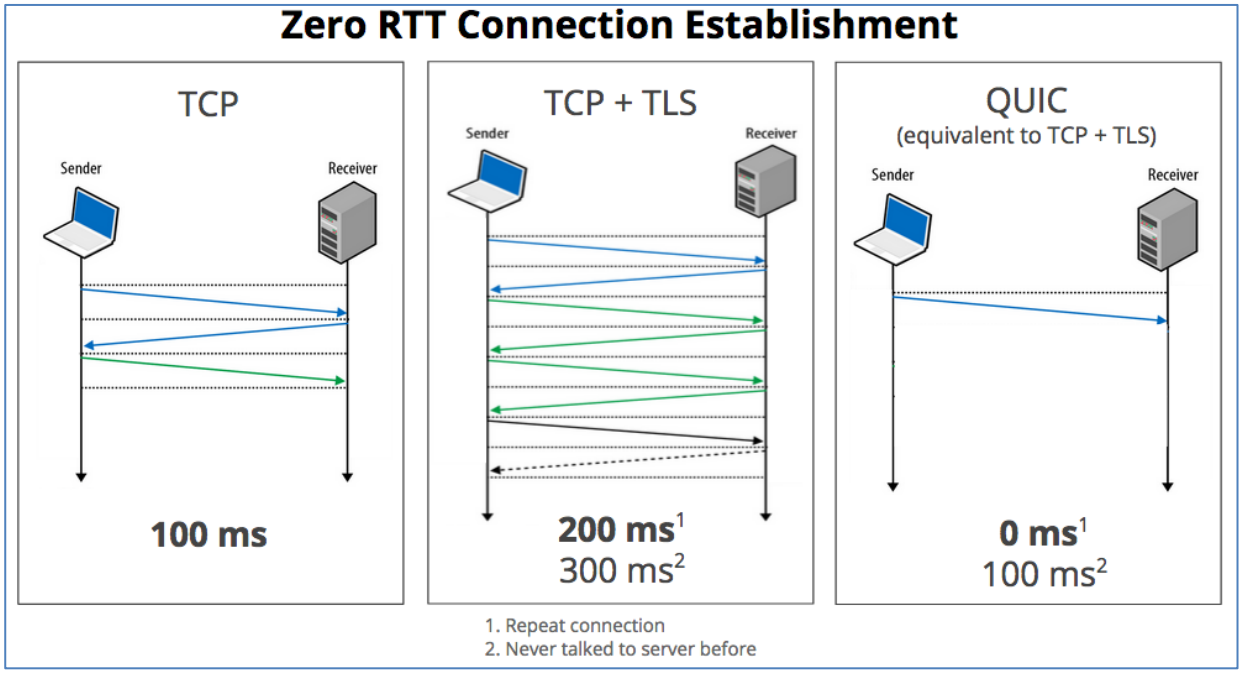

Figure 6 - Comparaison TCP, TLS et QUIC (@Google)

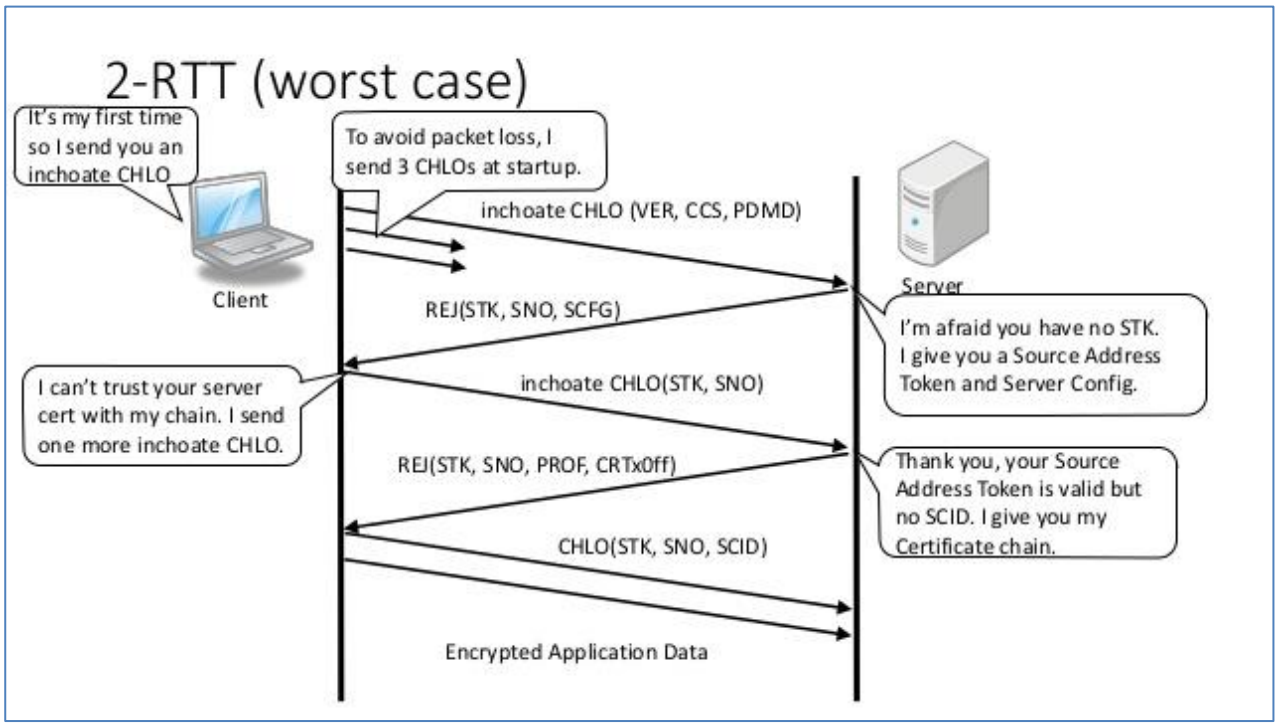

Figure 7 - Pire cas, le serveur à besoin d'authentifier le client

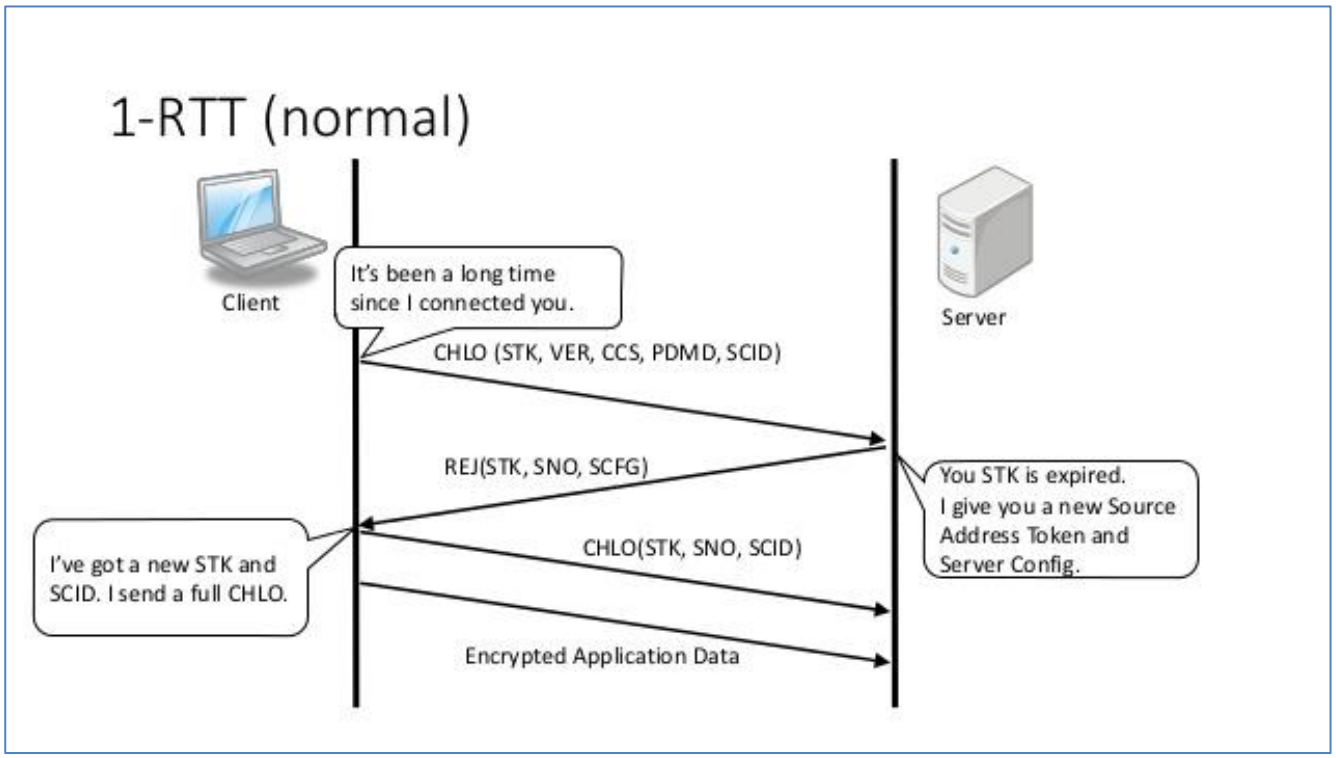

Figure 8 - Cas normal le client seulement authentifie le serveur

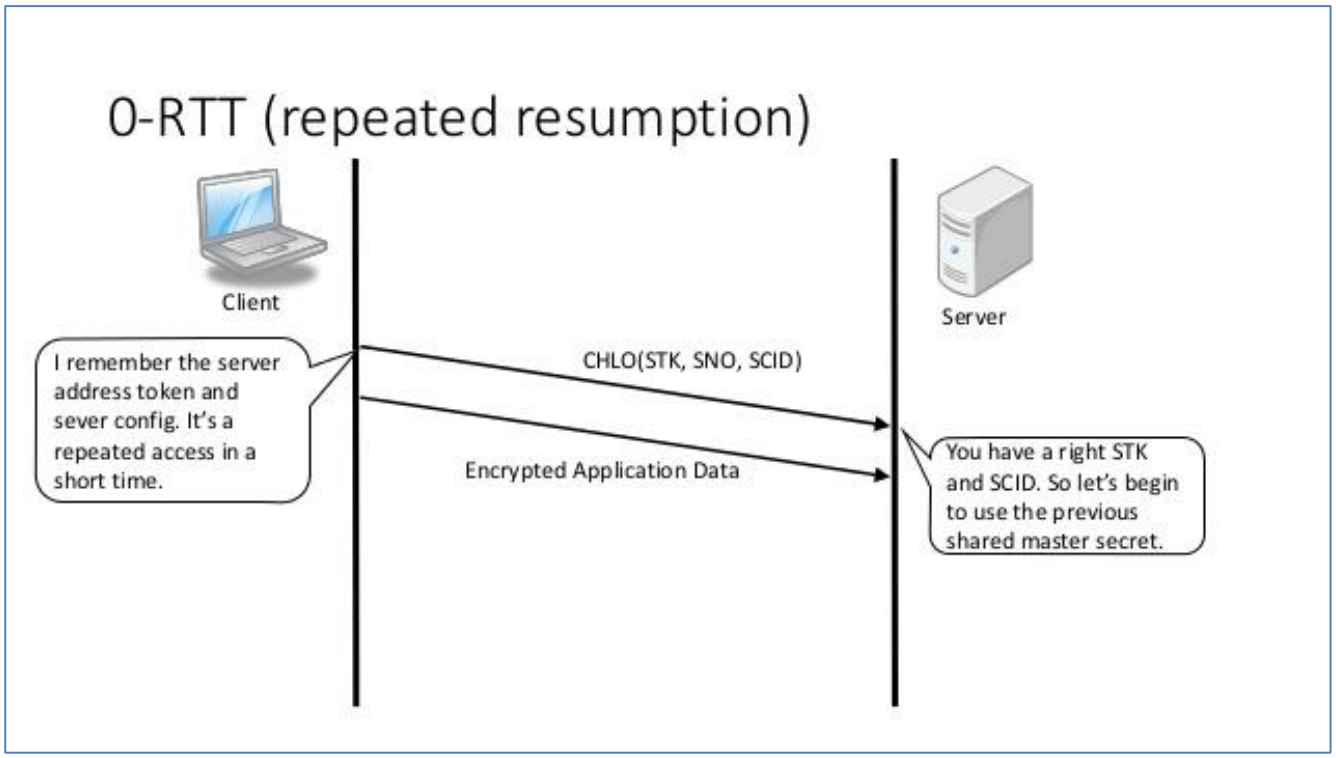

Figure 9 - Le client et le serveur se connaissent déjà

Aujourd'hui, le trafic web à destination des mobiles représenterait 30% du trafic total [DR11]. Une des particularités des systèmes mobiles est d'utiliser plusieurs interfaces réseaux (WiFi, 3/4G) et de basculer fréquemment de l'une à l'autre. Sans utilisation de protocoles spécifiques tels que mobile IP (ce qui est le cas aujourd'hui), ces changements de réseaux impliquent un changement d'adresse IP. L'adressage de la couche transport de l'internet étant basé sur le couple adresse IP et numéro de port, un changement de réseau implique obligatoirement une coupure de la connexion transport car le serveur connecté avant la mobilité est incapable d'associer la nouvelle adresse IP à l'ancienne connexion transport. Une solution pour pallier à ces réinitialisation de connexion serait de décorréler adressage transport de l'adressage IP. Ainsi, QUIC utilise un identifiant de connexion transport permettant la migration.

En plus de ces mécanismes, QUIC propose un mécanisme de FEC (Forward Error Correction) qui vise à éviter des retransmissions de paquets perdus en permettant leur reconstitution par l'ajout d'information redondante. Un mécanisme d'espacement minimum de paquet (packet spacing) vise à éviter de trop charger brutalement les routeurs et middle boxes.

2.4.3 - Problématique liée à UDP

Le trafic UDP, sans contrôle de congestion, pénalise les flux TCP en cas de congestion qui réduisent leur débit afin d'éviter d'aggraver la congestion. Cette situation a donc rendu nécessaire des mécanismes de priorisation des flux TCP sur les flux UDP dans les routeurs de l'Internet afin d'éviter une concurrence déloyale. QUIC étant implanté au dessus d'UDP, même si disposant d'un contrôle de congestion, pourrait se voir pénaliser de la même façon, à moins que par DPI ou avec un tag particulier ce protocole puisse être facilement identifié.

2.4.4 - QUIC sur satellite

Par son mécanisme d'ouverture de connexion rapide, son mécanisme de FEC et le multi-flux QUIC semble une solution idéale pour les réseaux LFN (Long Fat Network). Toutefois, QUIC implantant un contrôle de flux et un contrôle de congestion, ceux ci doivent aussi être adaptés aux longs délais.

La taille de la fenêtre liée au contrôle de flux est de 64 bits ce qui n'est pas limitant (comparativement aux 16 bits de celle de TCP). Initialement le protocole débute avec une fenêtre de 16 Ko mais l'augmente en fonction des besoins au travers d'une PDU WINDOWS_UPDATE.

Le contrôle de congestion [DR12] par défaut de QUIC est TCP Cubic. Toutefois il est possible d'activer par un mécanisme d'option TCP Reno. Il semblerait que la vision de google serait d'implanter plusieurs mécanismes de contrôle congestion et de sélectionner celui qui correspondrait le mieux au lien. Toutefois cette vision long terme n'est aujourd'hui absolument pas implantée ni spécifiée.

Selon les premiers tests [DR30] du protocole après sa sortie par des blogueurs, il semblerait que malgré des avantages sur le papier, les gains du protocole dans le contexte particulier des LFN ne seraient pas si évidents. Toutefois, ceci reste à étudier et à vérifier ! Il faut aussi noter qu'une dizaine de versions différentes ont été testées dans l'année 2014. La version actuelle est la version 25.

2.4.5 - Exemples d'entêtes QUIC

Les deux exemples de paquet QUIC, capturés à l'aide de l'extension « HTTP/2 and SPDY indicator » de chrome permettent de voir une emission de certificat X.509 pour le premier, en vu de l'établissement d'une connexion sécurisée.

Le second, représente une requête http / GET sur une ressource de type image. Ce dernier paquet montre la relation qui existe entre QUIC et http2, intimement liés.

```
+QUIC_SESSION  [dt=6040+]
--> cert_verify_flags = 6
--> host = "www.google.com"
--> is_https = true
--> port = 443
--> privacy_mode = false
--> require_confirmation = false

QUIC_SESSION_CERTIFICATE_VERIFIED
subjects                                                                    =
["*.google.com","*.android.com","*.appengine.google.com","*.cloud.google.com","*.g
oogle-
analytics.com","*.google.ca","*.google.cl","*.google.co.in","*.google.co.jp","*.go
ogle.co.uk","*.google.com.ar","*.google.com.au","*.google.com.br","*.google.com.co
","*.google.com.mx","*.google.com.tr","*.google.com.vn","*.google.de","*.google.es
","*.google.fr","*.google.hu","*.google.it","*.google.nl","*.google.pl","*.google.
pt","*.googleadapis.com","*.googleapis.cn","*.googlecommerce.com","*.googlevideo.c
om","*.gstatic.cn","*.gstatic.com","*.gvt1.com","*.gvt2.com","*.metric.gstatic.com
","*.urchin.com","*.url.google.com","*.youtube-
nocookie.com","*.youtube.com","*.youtubeeducation.com","*.ytimg.com","android.com"
,"g.co","goo.gl","google-
analytics.com","google.com","googlecommerce.com","urchin.com","youtu.be","youtube.
com","youtubeeducation.com"]

QUIC SESSION_CRYPTO_HANDSHAKE_MESSAGE_SENT
--> CHLO<
SNI : "www.google.com"
STK                                                                         :
0x428D64796E407F5B5643510099DDB98296B04616BF5C4A3A782B264CAC1D91F2DC1F2AFA7EABA1DE
789D14271EA141ED08E6F787A3946D49B6F7
VER : 'Q025'
CCS : 0x7B26E9E7E45C71FF
NONC: 0x562A2CF2149E80A6FE866780C9BE34040B148432346CD626215971F6D715B822
MSPC: 100
AEAD: 'AESG'
UAID: "Chrome/46.0.2490.71 Intel Mac OS X 10_10_3"
SCID: 0x73F42F4AED5F8E6B0D2184D592903A72
TCID: 0x00000000
PDMD: 'X509'
SRBF: 1048576
ICSL: 30
PUBS: 0xB1B1E20822439986134FD1EF12507EAB4E04930748C9803AEAD3D77367E8FC05
SCLS: 0x01000000
KEXS: 'C255'
COPT:
CCRT: 0x1519481FED66811CE2631A827D71855F400B7B90A9AE79EB
IRTT: 8414
CETV:
0x240A6E757ECB14A88336DF57C82B8F46FA7C9184BC2E70135C2D6DD1FD531EE8CE17D8F5FF3B4776
5BA3CA36ACC9F52B8CB9A39203B0209671C1D724E3AA5BCBC4279B9ED071BBD4B7F10125B7EC1C4C1F
AC5E3B0343812286FE82CFA693285D0276695DA7802F126513505BAF795353A521FE189B6CA17F6FD9
7B5805CDBF1D4CC2B6576D69E937616D2B00FCA2E910CB45CA433C8662BD07B90F9C762FA6571776CE
2F
CFCW: 15728640
SFCW: 6291456
```

Figure 10 - Premier paquet d'ouverture de session QUIC (decrypté)

```
QUIC_HTTP_STREAM_SEND_REQUEST_HEADERS
--> :authority: www.google.com
:method: GET
:path: /favicon.ico
:scheme: https
accept: */*
accept-encoding: gzip, deflate, sdch
accept-language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
cookie: [878 bytes were stripped]
if-modified-since: Tue, 01 Sep 2015 16:35:46 GMT
referer: https://www.google.com/textinputassistant/tia.png
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_3) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36
x-client-data: CKO2yQEIqbbJAQjBtskBCO+IygEI/ZXKAQiKl8oBCLyYygE=
--> quic_priority = 2
--> quic_stream_id = 5
```

Figure 11 - Paquet de requête http2/QUIC (équivalent à un get http1.1)

2.5 - État des lieux sur l'utilisation de http2

Selon W3Techs (Web Technology Survey) [DR1] l'adoption de http2 au 1er Juillet 2015 est encore faible (environ 1% des sites web l'utilisent), mais les sites les plus générateurs de trafic ont déjà effectué la transition (Google, Facebook, Twitter, Youtube, etc.). La Figure 12 compare la diffusion de http2 comparativement à d'autres technologies du web.

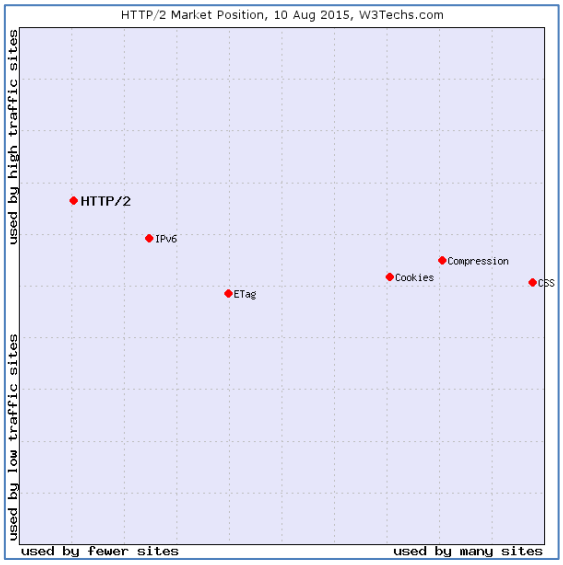

Figure 12 – Diffusion des technologies du web (@W3techs)

La principale faible diffusion de cette version tient au fait que récemment (Aout 2015), ni le serveur Web Apache, ni Microsoft-IIS ne supportaient http2. Toutefois, un module non officiel existe déjà pour Apache, et la version 10 de Windows est compatible. Cela ne laisse pas présager pour autant une fulgurante transition si l'on s'en réfère à l 'évolution des versions d'Apache par exemple. En effet, 3 ans après sa sortie, seuls 20% des serveurs Web Apache ont fait la transition. Ce chiffre est un peu supérieur pour les utilisateurs Microsoft.

Selon les auteurs, http1 continuera à dominer encore 2 ou 3 années, en attendant les mise à jour des nombreux serveurs web accessibles sur le web. Ce discours est néanmoins à modérer considérant qu'en volume, la majorité des requêtes est à destination des serveurs ayant déjà effectués la transition !

2.6 - Http2 et ses usages

2.6.1 - Compatibilité avec anciennes versions

La migration vers http2 ne sera pas instantanée et les navigateurs devront composer pendant de nombreuses années avec des serveurs implémentant une version http1.x. L'inverse est aussi vrai, mais dans ce cas, le serveur http2 sait répondre aux requêtes d'un client http1.x. La migration des serveurs et des clients n'est pas une fin en soit, car le Web est composé de nombreux éléments émettant des requêtes HTTP, comme les proxies, les applications utilisant une API http, etc. Ainsi, ce paragraphe aborde la compatibilité d'un serveur http1.x avec une requête http2.

Voici l'exemple d'un browser supportant http2 qui émet une requête sur un port 80 non sécurisé (car il n'a pas pu l'émettre sur le port 443 non disponible sur le serveur), à ce stade de départ il parle donc toujours ASCII et http1.1 :

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP/2-Settings
Upgrade: h2c
HTTP/2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

Et la réponse du serveur qui ne comprend que les deux premières lignes de la requête et va ignorer la suite :

```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
```

Si le serveur comprend la demande http2, il l'annonce en http1.1, et puis juste après la discussion passera dans le mode binaire de http2 :

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
```

Ainsi est assuré la rétro compatibilité du protocole. A noter que « h2c » correspond au mode non crypté de http2, c'est à dire sans TLS. L'option « h2 » est aussi possible, donc http2+TLS.

2.6.2 - Proxies

La compatibilité du protocole http2 avec les entités proxies très présentes dans les systèmes satellite doit être étudié afin de garantir une transition plus facile. http2 étant un protocole binaire, contrairement à son ainé, l'identification du contenu d'un échange n'est plus aussi simple et peu compliquer l'utilisation de proxy. Dans les systèmes satellites, ces proxies intègrent souvent plusieurs fonctions comme l'optimisation des connections TCP par le PEP, mais aussi des fonctions liées plus particulièrement à http, comme de caching web ou du prefetching. A noter que lorsqu'un proxy http est utilisé, une nouvelle connexion TCP est initiée entre le proxy et le serveur web distant impliquant de facto la possibilité d'adapter la connexion TCP sur le lien satellite. Comme démontré dans la R&T « Amélioration des performances applicatives dans un contexte hybride réseau terrestre / satellite, il est tout à fait possible d'influencer fortement le comportement d'un client TCP par l'emploi d'une version adéquate de TCP sur le serveur et donc de garantir un fonctionnement quasi-optimal sur la partie satellite et adaptant uniquement la version de la souche TCP sur le proxy http.

Lié à cette problématique, deux types principaux de proxies existent : les classiques proxies HTTP qui sont clairement adressés par le client (qui en a donc connaissance) ou qui sont auto-configurés et les proxies transparents qui interceptent la connexion http afin de rendre ce service invisible à l'utilisateur. Nous distinguerons donc ces deux cas dans les paragraphes suivants.

2.6.2.1 - Proxy http transparent

Le groupe en charge de l'élaboration du protocole http2 a prévu l'existante des proxies, mais l'utilisation d'une connexion TLS sécurisée rends le processus plus limité.

La solution la plus simple consiste donc à configurer son navigateur web pour adresser clairement un proxy http2, celui-ci étant précisé par le fournisseur de service dans son contrat d'abonnement. Dans ce cas, une connexion TLS est initiée entre le client et le proxy, et une autre entre le proxy et le serveur. Les requêtes http sont adressées au proxy qui essaye d'y répondre en cherchant dans son cache, au cas échéant il initiera une requête vers le serveur distant, puis relaiera la réponse au client.

Cette méthode n'est pas la plus efficace en cas de mobilité, puisque si le client a quitté le réseau satellite, il continuera néanmoins d'utiliser le service proxy (sans toutefois passer par le lien satellite). Comme pour tous types de proxy, il faut noter que le client doit avoir confiance dans le proxy, puisqu'à cet endroit, la connexion n'est plus sécurisée.

Une autre solution consiste à utiliser les mécanismes de détection automatique de proxy : PAC (Proxy Auto-Config) et WPAD (Web Proxy Autodiscovery Protocol). Le premier repose sur un fichier centralisé précisant la localisation du proxy mais n'est pas vraiment facilement déployable dans le contexte d'un fournisseur de service par satellite. Le second repose sur l'utilisation de DHCP ou du DNS pour diffuser l'existence du proxy. Il est à noter que cette technique proposée comme draft à l'IETF [DR7] mais jamais standardisée, n'est pas exempte de défaut, en particulier à cause du risque d'attaque malveillante qu'elle encourt. En effet, une usurpation de DNS ou de DHCP peut permettre de faire transiter les requêtes HTTP de manière transparente vers un proxy malveillant.

Le schéma suivant indique le principe d'un proxy adressé.

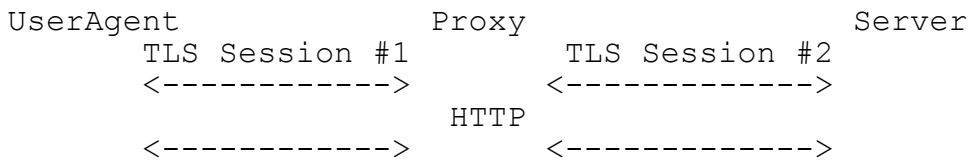

Figure 13 - Principe d'un proxy adressé

2.6.2.2 - Explicit Trusted proxy

Une proposition alternative, proposant un mécanisme de proxy transparents a vu le jour à l'IETF sous forme de draft [DR8], mais cette proposition controversée n'est encore qu'au stade de proposition.

La figure ci-dessous précise le principe. La connexion http n'est pas explicitement coupée, donc le mécanisme est transparent pour l'utilisateur. Celui doit donc pouvoir reposer sur la certitude qu'un éventuel proxy puisse être sécurisé ce qui est une hypothèse forte de ce draft.

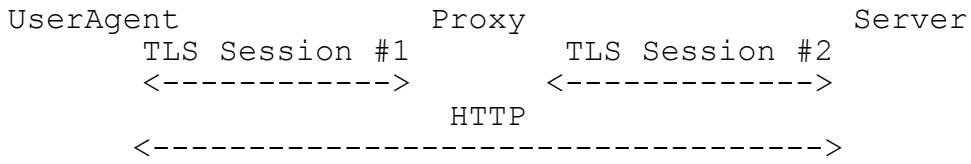

Figure 14 - Principe d'un Explicit Trusted Proxy

Deux modes de fonctionnement sont proposés :

- Avec un proxy certifié, par échange préalable de certificats, et en utilisant l'option (h2clr) explicité en amont, le proxy prend en charge le traitement des requêtes sur une connexion TLS établie indiqué précédemment. Le consentement de l'utilisateur quant à l'utilisation du proxy est obtenu pour une durée indéterminée par ces certificats.
- Avec un proxy captif, le mécanisme est un peu différent. Ici aussi, le proxy intercepte la requête TLS et analyse l'ALPN. Si l'option (h2clr) est précisée, la requête est redirigée vers un portail captif qui va s'assurer que le client est d'accord pour l'utilisation du proxy et configurer le proxy par l'échange d'un certificat et d'un fichier .pac (Proxy Auto-Config).

En cas de refus, le client a toujours la possibilité de requérir à l'option (h2), indiquant qu'il souhaite une connexion sécurisée de bout en bout.

Ce draft est controversé car il remet en cause la sécurisation de bout en bout apportée par http2. De fait, pour l'instant, les navigateurs refusant d'implanter l'option (h2clr) ce type de solution n'est pas envisageable.

2.6.2.3 - Impact sur les nouveaux services de push/streaming

L'existence de nœuds intermédiaire dans le web comme les nœuds cache et les serveurs proxy font partie des motivations qui ont poussé au développement du streaming vidéo sur http. En effet, vu que le streaming vidéo sur http peut être résumé par le téléchargement successif de petits fichiers multimédia adressés par un nom unique durant une session de streaming, l'implémentation d'un cache au niveau du serveur proxy améliore grandement les performances de streaming. Un serveur proxy servant plusieurs clients, éventuellement intéressés par les mêmes contenus vidéo, peut garder dans son cache les segments vidéos demandés par un client pour les resservir à d'autres clients. Ceci est permis par l'identification facile des segments.

Le passage au protocole http2 et ses proxies ne remet pas en cause ce principe de fonctionnement sauf dans le cas du « Explicit Trusted Proxy » où le serveur proxy aura une connaissance des requêtes et donc des segments vidéo, mais pas de leur contenus vu que les segments de données passent par une connexion http2 entre le client et le serveur du contenu. Ne voyant plus le contenu des segments passer par lui, le serveur proxy ne pourra pas cacher ces segments.

2.6.3 - CDN et http2

Les CDN ont participé à l'amélioration du chargement des pages web qui au fil du temps nécessitent de plus en plus d'objets pour la génération de l'affichage final. Le CDN assure donc la diffusion des objets, plus ou moins volumineux, d'un site web à un grand nombre d'utilisateurs en déchargeant ainsi le serveur d'origine de servir le même contenu (souvent statique et volumineux) à chaque accès client. Les serveurs CDN vont plus loin dans la diffusion en essayant de placer le contenu au plus près du client.

Avant l'arrivée de http2, les développeurs web ont développé plusieurs techniques ou « hacks » pour pousser au maximum les performance de http1.x. Ces techniques s'appuient en général sur l'existence de serveurs CDN ayant déjà une partie des ressources d'un site web (logo, bibliothèques JavaScript communes, contenu multimédia, etc.). Ces techniques incluent :

- **Domain sharding:** Le *domain sharding* est une technique pour répartir les différentes ressources d'une page web sur plusieurs domaines (DNS). En effet, quand plusieurs domaines sont utilisés, les navigateurs peuvent télécharger ces ressources simultanément ce qui donne un temps total réduit. En général, les navigateurs modernes supportent une moyenne de 6 téléchargements concurrents.

  Cependant, le passage par plusieurs domaines peut impacter les performances. En effet, les navigateurs doivent faire une résolution DNS pour chaque domaine additionnel, mais aussi maintenir cette connexion. Une étude par Yahoo [DR16] a démontré que 2 à 4 domaines est un ratio optimal.

  L'utilisation de plusieurs connexions http à différents domaines pour récupérer les objets d'un site web peut être vue comme contreproductive lors du passage au http2 vu que ce dernier offre des fonctionnalités pour multiplexer plusieurs requêtes/réponses au sein d'une seule connexion http2. Cependant, dans un contexte satellite, une seule connexion http2, comme suggéré, peut offrir de mauvaise performances vu que le problème se verrait déplacé au niveau de la couche de transport (TCP), où toutes les données vont utiliser une seule connexion TCP, alors que si plusieurs connexions TCP étaient utilisées, les performances seraient meilleures. De ce fait, le *domain sharding* reste intéressant dans un contexte satellite même avec l'utilisation de http2.

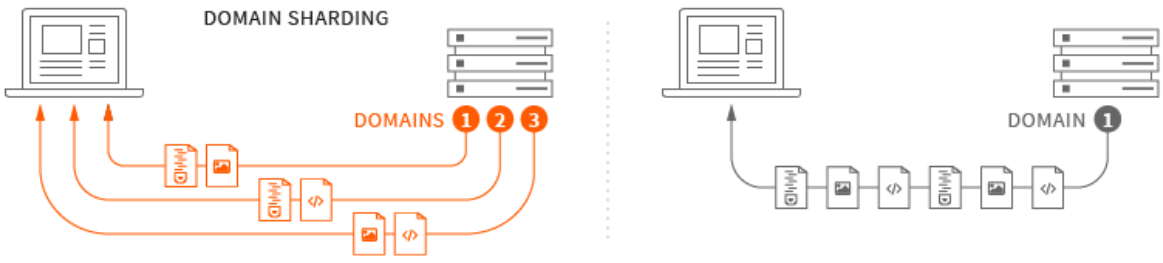

Figure 15 – Domain sharding

- **CSS Sprite :** Le CSS Sprite est technique d'optimisation du chargement d'une page web qui consiste à combiner plusieurs images en une seule appelée « «sprite sheet ». Au lieu de télécharger plusieurs fichiers avec plusieurs requêtes, un seul fichier est téléchargé et le navigateur affiche l'image (ou sprite) selon le contexte. Des exemples incluent les différentes images associées à un bouton (actif, inactif, surbrillance, etc.).

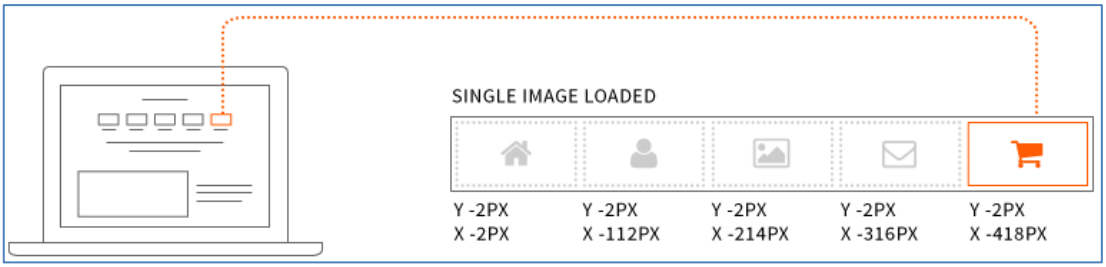

Figure 16 - CSS sprite

- **Resource combination :** La combinaison de ressources consiste à compresser et de combiner au sein d'un seul fichier plusieurs ressources utiles à une page web comme les fichiers JavaScript, les feuilles de style CSS, les images, etc. afin de retirer ces objets-là à l'aide d'une seule requête http et ainsi diminuer le nombre de connexions à établir avec le serveur pour récupérer tous ces objets.

- **Images adaptatives :** Cette technique consiste à stocker sur le serveur plusieurs résolutions d'une même image. Le code coté serveur (ex. PHP) détecte la résolution de l'écran du client et envoie donc le fichier le plus approprié. Cette technique vise donc à réduire la quantité de données échanges entre le serveur et le client selon les besoins du client. Cette technique peut être étendue à d'autres types d'objets, comme des versions légères des script utilisés sur le site, ou restreindre les feuilles de style au minimum utilisé par le client.

Même si le protocole http2 améliore le temps de chargement des pages web et de ses éléments, l'utilisation d'un CDN n'est pas pour autant à abandonner. En effet, les fonctionnalités de http2 améliorent comment un contenu web est chargé mais n'ont pas d'impact sur la distance qui sépare un contenu de son utilisateur. Ainsi, les CDN et les serveurs de cache restent la seule solution au problème de localisation du contenu au plus près de l'utilisateur.
Il est clair que le multiplexage de plusieurs requêtes au sein d'une seule connexion peut réduire l'effet du RTT mais un RTT très faible (~ 20ms), qui serait offert par un CDN, améliorerait encore la durée totale du chargement.

2.6.4 - Http2 et les nouveaux usages du web

2.6.4.1 - Http2 et l'internet des objets

Avec les progrès de la miniaturisation et le développement des plateformes matérielles à faible coût (mini-PC tels que Raspberry Pi [DR17], BeagleBone [DR18], Intel Edison [DR19]), la proportion d'objets connectés à internet a pris une importante part de l'internet aujourd'hui. Cette démocratisation a été accompagnée par le développement de protocoles de communication spécifiques dédiés à ces équipements qui ne disposaient pas de ressources suffisantes. Parmi ces protocoles, on trouve CoAP[DR28], MQTT[DR29] ou des protocoles binaires développés sur mesure pour les applications ciblées.

Cette forte évolution est derrière la fragmentation verticale du monde de l'IoT où chaque système (i.e. l'ensemble des objets connectés, des protocoles de communication, technologies d'accès réseau employés) sert uniquement l'application ciblée sans possible réutilisation pour d'autres types d'applications. Pour pallier à ce problème, plusieurs efforts de standardisation et/ou normalisation (ETSI SmartM2M[DR20], oneM2M[DR21], etc.) ont vu le jour afin de proposer des plateformes de services horizontales pour les applications de l'IoT.
Ces plateformes de service se rejoignent pour l'intégration de HTTP comme protocole principal [DR22][DR23] pour la communication entre les différents nœuds communicant (objet ↔ passerelle ↔ serveur). Cependant, le protocole http1.1 n'est pas très adapté à l'IoT pour deux raisons :

- Tous les équipements déployés ne disposent pas des ressources nécessaires pour l'exécution d'une pile TCP/IP et une application HTTP ;
- Les réseaux d'accès utilisés n'offrent pas souvent un canal de communication adapté à http/1.1 (faible débit, latence, coût au bit transmis, etc.).

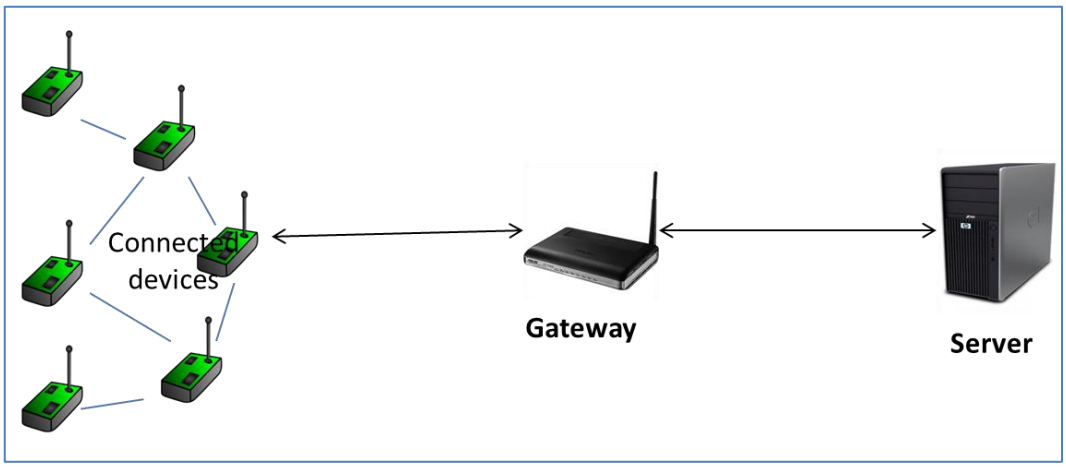

Figure 17 - M2M/IoT System

L'arrivée de http2 pourrait cependant résoudre ces problèmes ou les réduire. Il est évident que http2 ne sera pas aussi efficace que les protocoles dédiés aux plateformes embarqués, mais les efforts fournis lors de sa conception (peu de mémoire et de bande passante) rendent envisageable son utilisation pour l'internet des objets. En effet, grâce à la technique de compression de headers HPACK[DR24], http2 utilise moins de bande passante. HPACK peut aussi être configuré, par négociation, afin de limiter la mémoire utilisée pour décoder les headers http2.
L'autre avantage de http2 est qu'il pousse à utiliser une seule connexion TCP entre le client et le serveur. Toutes les requêtes et réponses sont alors multiplexées au sein de cette connexion. Par négociation, le nombre de flux simultanés peut être limité pour s'adapter aux ressources du client. Comme il s'agit souvent de petits messages échangés entre les objets connectés et les passerelles/serveurs, les avantages introduits par http2 comme la suppression du handshake à trois phases pour chaque requête/réponse, pas de renégociation TLS, suppression du slowstart, les communications seront plus fluides et utiliseront moins de bande passante.
L'utilisation de http2 pour les objets connectés bénéficient aussi de :

- **Codage binaire des messages** : utilisation réduite de la bande passante ;
- **Mécanisme du push** : ceci permet d'éviter l'utilisation du polling et son impact sur le traitement, l'espace mémoire occupé par les requêtes en attente, et enfin libérer le canal de communication de message de polling inutiles. Le mécanisme push améliore grandement la conception des applications pour l'IoT vu qu'il permet à l'objet de pousser ses données à son rythme mais permet aussi au serveur de requêter le capteur (i.e. pour avoir la dernière version des données) ou lui envoyer une commande (cas des actionneurs) ;

- **Mécanisme du Ping** : pour l'implémentation transparente des applications heartbeat sans l'utilisation d'un mécanisme supplémentaire ;
- **Ajustement des fenêtres de transmission et négociation de la taille des headers** : grâce à ces deux fonctionnalités, on peut configurer de manière précise les ressources allouées aux communications http2.

Http2 peut ne pas être la solution miracle pour intégrer cette partie de l'Internet qui n'est pas encore bien visible, mais il permet de se rapprocher encore plus de l'objectif du web des objets (WoT : Web of Things[DR25])

2.6.4.2 - http2 et le streaming multimédia

Le streaming http, et plus spécialement le streaming adaptatif, a connu un intérêt grandissant ces dernières années. Ce mode de diffusion a été rendu possible par les dernières évolutions du web à savoir les efforts de standardisation autour des codecs, l'adoption de HTML5 et son intégration native d'un élément vidéo dans les pages web qui ne nécessite plus l'utilisation d'un plugin tiers. le développement des infrastructures réseau constitue aussi une autre motivation qui a permis de passer des techniques de streaming *push-based* (le serveur de streaming pousse le contenu vers le client sans requête préalable de sa part) à des techniques *pull-based* (où le client requête périodiquement des portions du contenu multimédia), techniques qui n'étaient pas envisageables avec des communications à faible débit et grandes latences. Une autre motivation réside dans l'amélioration qu'ont connue les infrastructures web classiques par l'utilisation des proxies performants ainsi que le déploiement massif de réseaux de distribution de contenus (CDN), améliorations qui ont délocalisé la charge du serveur http vers d'autres nœuds plus performants du réseau. Ainsi, le streaming http est devenu une réalité que ce soit pour les contenus en direct (Live) ou des contenus à la demande (VoD).

Le streaming sur HTTP a réellement débuté avec le téléchargement progressif d'un contenu multimédia. Dans le téléchargement progressif, le contenu est encodé d'une manière à permettre sa lecture avant le téléchargement complet du fichier. Cette approche est supportée par n'importe quel serveur http sans modification ou augmentation. Cependant, le téléchargement progressif n'offrait pas toutes les fonctionnalités qu'on retrouve dans le streaming classique comme par exemple la navigation dans le flux. D'un autre côté, le streaming classique nécessite, jusqu'à maintenant, l'utilisation de protocoles séparés tels que le protocole RTP qui a été utilisé pour les visioconférences et les communication VoIP, et le protocole RTSP qui est utilisé pour la signalisation. Ces protocoles utilisaient des ports différents du port 80, chose qui les rendait facilement identifiables et donc facilement bloqués par les politiques de sécurité de différentes organisations pour protéger le réseau d'un contenu gourmand en ressources réseau.

Ainsi, le streaming HTTP constitue une évolution naturelle du téléchargement progressif tout en profitant des avantages du streaming classique tels que l'adaptation au débit disponible pour le streaming. Cette évolution a donné naissance à une famille de techniques et standards : le streaming adaptatif sur HTTP. Dans ce mode de distribution de contenu multimédia, le contenu d'abord encodé en plusieurs versions (ou représentations) où chaque version correspond à une certaine qualité et a un certain débit. Chaque représentation est alors décomposée en une série de petits segments (chunks). Ces segments représentent généralement une durée effective de 2 à 30 secondes, mais avec une décomposition uniforme dans chaque représentation. Les segments sont ensuite annoncés dans des fichiers d'index (liste de lecture, manifeste, etc.) et téléchargés par les clients. Chaque client, requête périodiquement les différents segments et commence la lecture dès la réception d'un ou de plusieurs segments selon la politique de mise en tampon adoptée.

Durant le streaming, le client analyse la bande passante et les performances de sa connexion avec le serveur et peut changer de représentation pour s'adapter aux évolutions de la bande passante. Cette décision d'adaptation peut être implémentée au niveau du client tout comme au niveau du serveur si on est en mode push.
Parmi ces techniques, on peut citer Apple HTTP Live Streaming (HLS), Adobe Dynamic Streaming for Flash, Microsoft Smooth Streaming, QuavStreams Adaptive Streaming over HTTP, et dans un effort de standardisation, le streaming adaptatif sur HTTP repose maintenant sur MPEG-DASH qui a été développé en 2012.

MPEG-DASH a été spécifié pour utiliser http/1.x comme protocole de transport, mais il peut supporter sans modification http2. Dans l'étude [DR27], il a été démontré que le passage au http2 a significativement amélioré les performances du streaming. Dans cette étude, les auteurs ont évalué l'overhead du protocole HTTP avec ou sans chiffrement.

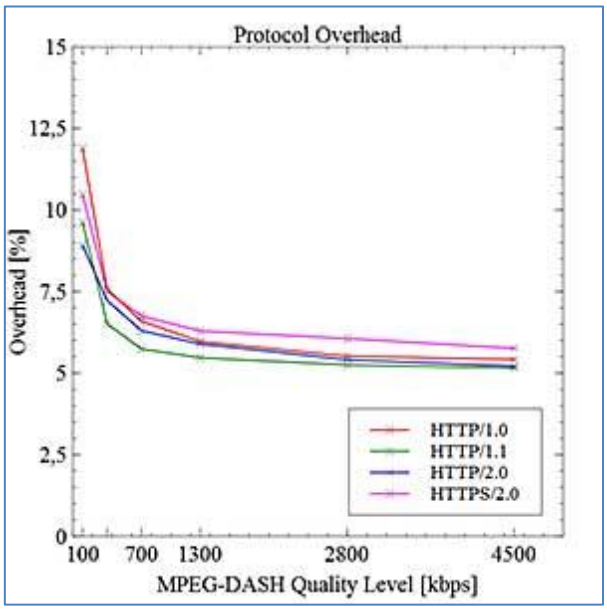

Figure 18 - Overhead selon la version de http pour le streaming MPEG-DASH [DR27]

À travers la Figure 18, on peut voir que l'overhead de communication est relativement faible (5 – 7%) pour un flux DASH décomposé en segments de 2 secondes.

La Figure 19 [DR27] montre l'effet de la valeur du RTT sur l'utilisation du lien selon la version du protocole http utilisé durant l'expérience. On voit que le protocole http/1.0 n'offre pas une utilisation efficace du lien de communication pour un RTT supérieur à 100ms. http/1.1 résout ce problème en utilisant les connections persistantes et le pipelining des requêtes et offre ainsi une tolérance à des valeurs élevées du RTT et une utilisation de la connexion supérieure à 90%. Les performances de http2 sont similaires à celles de http/1.1 vu l'utilisation des mêmes fonctionnalités (une seule connexion, pipelining). Cependant, il faut noter que ces fonctionnalités sont implicitement inclues dans http2, alors qu'avec http/1.1, elles ne sont pas disponibles tout le temps en raison de la présence de serveurs et de proxies ne les supportant pas vu qu'elles sont optionnelles dans http/1.1.

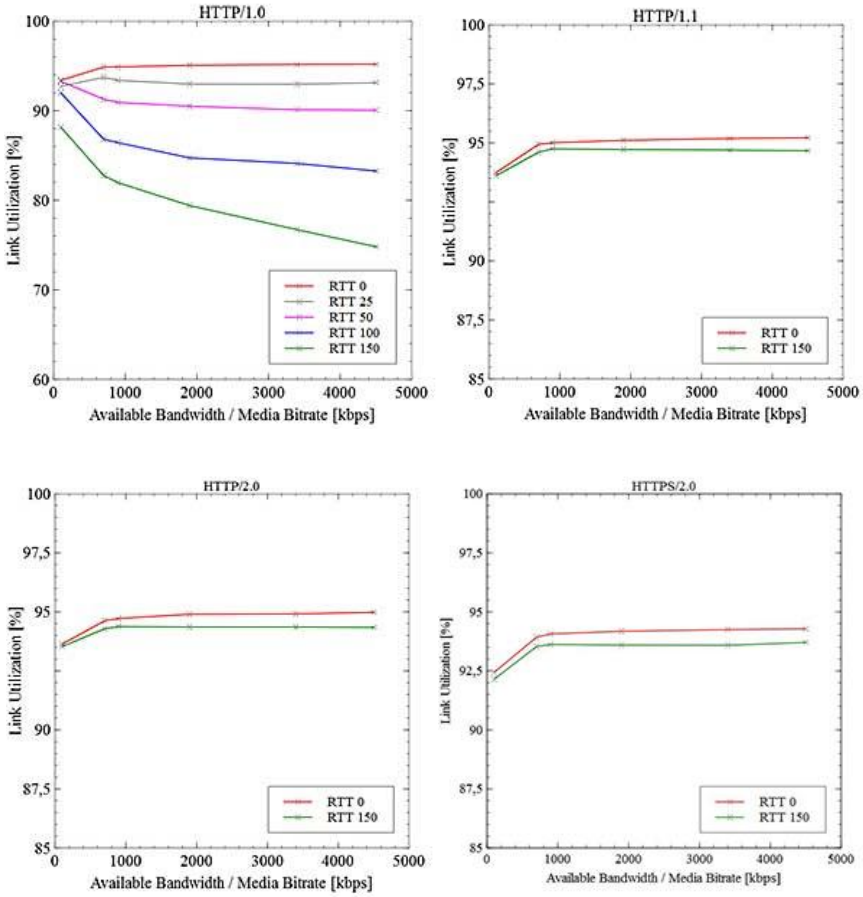

Figure 19 - Utilisation du lien selon la version de HTTP [DR27]

Le protocole http2 peut améliorer significativement le streaming sur http et plus particulièrement le streaming MPEG-DASH. En effet, les différentes fonctionnalités introduites dans http2 comme le mode push, l'utilisation d'une seule connexion, le multiplexage de plusieurs messages, et l'encodage binaire des entêtes servent déjà les besoins du streaming sur http2.

2.6.4.3 - http2 et les applications réparties

Une des tendances actuelles en développement back-end est la montée du développement à base d'API. Le terme API désigne une interface de programmation (Application Programing Interface). Celle-ci permet à un système externe d'interagir avec un autre système de données.

Le développement axé sur une API peut être considéré comme une séparation des sujets ou une externalisation dans le domaine du développement web. En résumé, à la création d'un nouveau projet (dans la plupart des cas, une application web personnalisée), l'externalisation via une API de certains services devient aujourd'hui indispensable afin de concentrer les budgets de développement sur des objectifs d'affaires plus rentables.

Il n'est pas surprenant que cette approche du développement axé sur des API externalisées soit en vogue : elle offre de nombreux avantages :

- En utilisant des API, le développeur peut se concentrer sur des fonctions du site web vraiment personnalisées plutôt que d'essayer de réinventer la roue en créant des fonctionnalités lui-même à partir de zéro.
- La qualité de l'API est en général élevée, car souvent développée par les meilleurs programmeurs du marché, ce sont des solutions éprouvées et fiables. Beaucoup d'heures de développement peuvent être économisées en utilisant des API lorsque c'est possible.
- La plupart des API sont libres d'utilisation ou ont une option gratuite limitée (souvent suffisant pour commencer).

Ce développement d'API s'est reposé sur le protocole http comme protocole d'invocation des différents services pour la construction d'application web principalement. Il a été ensuite propulsé par le développement d'applications mobiles qui se sont vues consommer les mêmes services sans pour autant repenser la conception des APIs vu que le protocole http et la présence d'un navigateur web étaient intégrés depuis le début dans tous les smartphones.

Le protocole http a longtemps été utilisé comme un protocole de messaging dans les middlewares orientés messages. En effet, le développement des services web a profité d'une infrastructure existante et fiable pour le rapport des messages entre les différents processus d'une application. L'autorisation du trafic http par tous les pare-feu a permis de s'affranchir des problèmes rencontrés auparavant dans l'approche RPC qui nécessitait l'ouverture de ports spécifiques et qui représentait de ce fait une faille de sécurité.

Actuellement, et dans le domaine d'applications connectées, on a assisté à un shift dans la conception d'une approche orientée services (*SOA : Service Oriented Architectures*) vers une approche orientée ressource (*ROA : Resource Oriented Architectures*). Si la première approche repose sur l'appel de services (web), la deuxième approche repose sur l'accès et la modification de l'état de ressources.

Le style REST (Representational State Transfer) définit des opérations de base pour la manipulation des ressources résumées dans le sigle CRUD pour CREATE, RETRIEVE, UPDATE et DELETE, auxquelles on peut ajouter l'opération EXECUTE. La facilité et la transparence de la traduction de ces opérations par des méthodes http comme PUT,GET, POST, DELETE, a propulsé le protocole http comme protocole de communication dans les architectures et API REST.

Le style REST repose sur une communication sans état entre le client et le serveur http. Ainsi, dans les API actuelles, chaque appel REST se fait à l'aide d'une connexion http dédiée. L'utilisation d'une connexion persistante dans le cas de http2 améliorerait les performances en multiplexant les différents appels au sein d'une seule connexion sans compter le gain en bande passante résultant de l'encodage en binaire des entêtes de chaque requête. Cet avantage est seulement possible dans le cas où la majorité d'appels se font avec le même serveur. Ceci deviendrait donc peu efficace dans les applications consommant plusieurs APIs fournies par différents serveurs.

Le protocole http a été et reste toujours un outil de choix pour l'invocation de services distants. Le développement d'applications réparties se basera toujours sur le protocole http comme protocole de transport. Le passage au http2 améliorerait les performances dans le cas où la plupart des appels de services (messages HTTP) se font avec un seul serveur. D'un autre côté, et en raison des connexions permanentes avec les serveurs, http2 serait moins bénéfique aux applications distribuées dans le cas où plusieurs services seraient offerts par différents serveurs.

2.7 - Problématique spécifique au contexte satellitaire

2.7.1 - RTT important

Les spécificités du satellite impactent fortement l'efficacité des protocoles classiques. La latence, caractérisée par le RTT de plusieurs centaines de millisecondes du lien satellite, met en lumière des problèmes importants dans les premières versions de HTTP. Comme décrit précédemment et afin de minimiser l'impact du RTT sur le ressenti utilisateur, http1.1 a introduit le concept de pipelining. Cette technique permet de multiplexer plusieurs requêtes HTTP dans une seule connexion TCP, évitant ainsi la démultiplication des 3-way handshake de TCP. Malheureusement, le pipelining souffre de plusieurs problèmes ; il est généralement désactivé par défaut dans les implémentations. On peut notamment citer le *Head-of-line Blocking* : les requêtes et réponses successives étant séquentielles, quand une perte apparaît c'est toutes les requêtes suivantes qui sont bloquées. C'est particulièrement important sur les liaisons satellites où le taux de perte est non-négligeable et où le RTT élevé rend l'attente de la réémission du paquet conséquente.

Http2 semble fournir une solution intéressante : le concept du pipelining est repris et amélioré. Comme exposé dans la partie 2.2 - de ce document, dans http2, l'ordre de réception des réponses est indépendant de l'ordre d'émission des requêtes ; les différents flux coexistent donc au sein d'une même connexion TCP sans que la perte d'un paquet pénalise la totalité des flux.

Selon [DR5] [DR13][DR14] qui ont étudié les principes du pipelining sur lien satellite, les résultats d'http2 en contexte satellitaire sont fortement dépendants de la couche de transport. Le temps d'attente d'affichage des éléments peut-être rallongé à cause du multiplexage des ressources sur une seule connexion TCP. Le fort taux d'erreur révèle en revanche un autre problème de *Head-of-line Blocking* mais ici à cause de TCP (voir paragraphe 2.4 -), ce dernier passant son temps à attendre les retransmissions de segments perdus. Par contre la latence n'a aucun effet rédhibitoire sur le protocole mais en réduit seulement les performances.

En cas de fort taux d'erreur sur un canal, la multiplication des connexions TCP compense les temps d'attente dus aux retransmissions. Dès lors, des techniques d'optimisation de sites web pour http1.x tel le *domain sharding* permettent de multiplier les connexions TCP et d'aider http2 en contexte à fort taux d'erreur [DR15].

2.7.2 - Chiffrement du trafic

Au-delà de la possibilité d'abandonner complètement l'utilisation des PEP, il convient de réfléchir aux problèmes qui peuvent se poser avec la démocratisation d'http2 et son intégration avec une architecture PEP classique. Bien que TLS est techniquement optionnel dans la norme http2, l'état des implémentations le rend de fait obligatoire (cf. 2.3.4 -). A ce titre, il est raisonnable de penser que le chiffrement aura lieu pour une grande part des communications utilisant ce protocole. De la même manière, les PEP, qui ont des difficultés avec les communications chiffrés de type IPsec, risquent d'être difficile à mettre en œuvre.

2.7.3 - Etat de l'implémentation d'http2

SPDY est un protocole développé par Google : il est à la base des travaux de l'IETF pour http2. Néanmoins, http2 n'est pas une version du code de Google et bien qu'il en reprenne beaucoup de concepts et de bonnes idées (multiplexage des sessions, compression des headers, intégration du TLS) il est nécessaire d'utiliser les implémentations http2, moins nombreuses que celles de son précurseur. Les modifications les plus récentes peuvent prendre du temps à être propagées dans les implémentations existantes.

2.7.4 - Analyse des couches de transport alternatives

TCP a été créé pour assurer des échanges sûrs et éviter une congestion du réseau. De par son fonctionnement, le protocole n'est pas adapté à des transmissions souffrant de fortes latences et où la perte de paquets est tout sauf occasionnelle. Sur un accès satellitaire, il n'est pas rare d'obtenir un mauvais BER ($>10^{-5}$) et donc une perte de paquets non négligeable, tout comme il est constaté une augmentation significative du RTT moyen (600ms). Dès lors les mécanismes de protection de TCP limitent la taille de la fenêtre de transmission et déclenche des retransmissions en permanence, réduisant le trafic utile à quelques Mbps quand les liaisons pourraient aisément atteindre plusieurs dizaines de Mbps.

La problématique d'un RTT élevé est également soulevée par les réseaux terrestres LTE (latence de 60 à 120ms), ce qui a conduit les acteurs principaux d'Internet à chercher des alternatives efficaces à TCP. Sont alors apparus les protocoles SCTP, QUIC et les améliorations TCP telles que TFO (TCP Fast Open [DR26]). Une analyse de ces mécanismes relativement récents permet de dégager des axes de test pour la suite de l'étude au cours de laquelle il serait aussi intéressant d'éprouver des protocoles applicatifs autres que HTTP sur ces protocoles de transport, par exemple FTP/SFTP, WebDAV, SSH, NFS.

2.7.5 - Transfert de fichiers par satellite, fiabilité et débit

Sur un accès satellitaire, l'utilisation du TCP natif sans PEP entraine une dégradation des débits assez conséquente. L'imagerie par satellite nécessitant le transfert de nombreux fichiers, pour la plupart très volumineux (quelques gigaoctets) il est très difficile et très fastidieux de mettre en place un tel type de montage sur des accès satellitaires classiques ne disposant que de quelques Mbps. De plus, la fiabilité des accès satellites peut être remise en cause en corrélation avec les aléas de la météo, entrainant une dégradation du BER pouvant conduire à des pertes de paquets et donc à des échecs de transfert de fichiers.

Dès lors, il convient de considérer d'autres alternatives pour mettre en œuvre ces transferts de fichiers. http2 permettant d'accélérer l'établissement des connexions et le multiplexage des requêtes, un mécanisme pourrait découper les fichiers à transmettre en plusieurs blocs logiques, calculer des blocs de correction d'erreur (FEC) et mettre des liens pour l'ensemble de ces blocs sur une page PHP accessible via un serveur http2.

2.8 - Utilisation et limites d'HTTP 1.0 et 1.1 sur les réseaux satellitaires

Les PEP ont été développés dans l'optique de limiter les impacts du satellite sur les sessions TCP. L'adaptation des paramètres TCP entre les entités communicantes (typiquement entre client et serveur) permet d'optimiser les échanges dans le cadre de la négociation pour la taille de fenêtre TCP. Cette négociation génère sur une base dichotomique un nombre très important de requêtes, qui ralentissent la mise en place des sessions du fait de la latence induite par la liaison satellitaire. Ces fonctionnalités de tuning de TCP, primordiales pour les échanges point-à-point entre deux entités séparées par un lien satellite, ont été par exemple implémentées par les différents constructeurs de modems satellite (e.g. iDirect ou Newtec), en allant jusqu'à la définition de nouveaux protocoles en remplacement de TCP.

Ces adaptations protocolaires présupposent la possibilité de rupture des sessions TCP, pour créer des sessions spécifiques sur le lien satellite. C'est un modèle parfaitement adapté aux opérateurs d'accès satellite fournissant directement le contenu à l'utilisateur final et cela suppose de pouvoir interrompre la session au niveau de l'ancrage satellite. Le design des architectures des systèmes de télécommunications satellitaires évolue néanmoins au fur et à mesure de l'intégration avec les réseaux terrestres.

Les principaux acteurs de l'économie de l'Internet sont désormais les fournisseurs d'applications (par exemple Google, Facebook, Twitter). Leur problématique consiste à toucher le maximum d'utilisateurs, ce qui revient à raccorder à l'Internet des régions du globe non desservies ou mal desservies. Le soutien de puissants fonds de financement leur permet désormais d'envisager sérieusement l'extension de leur domaine d'action en tenant le rôle de fournisseur d'accès, en complément de leur rôle de fournisseur de services. Il est ainsi légitime de voir ces sociétés se positionner sur le déploiement de systèmes satellitaires complets.

Ces systèmes satellitaires seront raccordés au backbone terrestre existant des fournisseurs de services. Le routage des informations est assuré par la définition de plans réseaux dédiés ; l'empilement de différents plans réseaux amène classiquement à encapsuler les données dans des tunnels IPsec empêchant ainsi toute terminaison TCP intermédiaire au niveau de la gateway à laquelle la station utilisatrice est rattachée.

L'enjeu pour les fournisseurs de service est de mettre à disposition de leurs utilisateurs du contenu hébergé sur des datacenters raccordés à l'Internet. La figure ci-dessous illustre le principe de raccordement de client pour un fournisseur de service au travers de sa propre infrastructure d'accès par satellite.

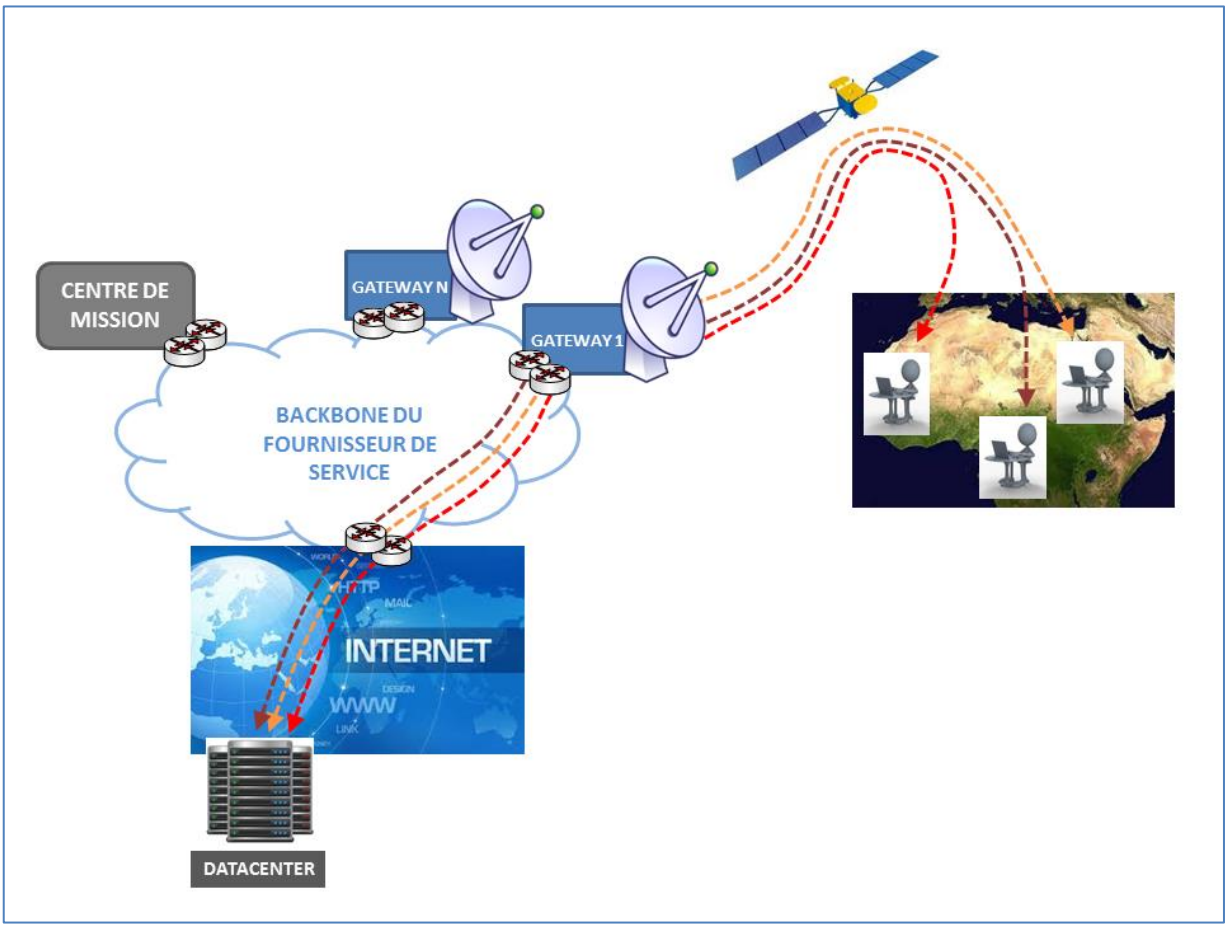

Figure 20 - Schéma de connectivité fournisseur de service

Comme il n'est pas sainement envisageable d'adapter les paramètres de TCP de l'utilisateur jusqu'au serveur de contenu, les datacenters hébergeant du contenu pour tous les utilisateurs qu'ils aient un accès terrestre ou satellite, il n'est pas surprenant de voir que ces nouveaux acteurs sur le marché spatial pousse le développement de http2.

Le besoin des fournisseurs de service est définitivement de rendre le satellite transparent dans leur infrastructure et dans le rendu vu de leurs clients au même titre que les autres médias d'accès. Dans la mesure où HTTP, dans ses versions actuelles, présente des inconvénients majeurs dans son utilisation sur satellite sans l'utilisation explicite de PEP, il est nécessaire de trouver des alternatives viables.

3 - SCENARII D'UTILISATION D'HTTP2

Dans ce chapitre, nous proposons une réflexion rapide sur l'écosystème HTTP actuel et les multiples utilisations qui peuvent être faites de ce protocole. L'objectif primaire sera de dégager des scénarios des tests permettant d'identifier les verrous techniques débloqués par l'arrivée du http2.
De plus, en se basant sur une architecture SATCOM classique, nous définissons un socle représentatif pour la mise en place des tests (tâche 3) qui viseront à mettre en avant les apports du http2.

3.1 - Cadre d'application

HTTP est devenu le protocole référence pour délivrer du contenu au travers d'Internet. En effet, les acteurs émergents d'Internet sont les fournisseurs de services, des services qui sont accessibles depuis les navigateurs Internet classiques. Un atout majeur du protocole HTTP est d'être finalement le seul protocole autorisé à traverser les pare-feux, une problématique qui se pose non seulement pour les infrastructures d'accès datacenter mais aussi au niveau des utilisateurs domestiques.
Outre l'accès à des sites informatifs ou de commerce, HTTP permet de récupérer du trafic multimédia audio ou vidéo, en streaming ou en téléchargement. D'autre part, l'accès à des applications sur Internet se fait communément via un accès HTTP à un portail d'accès. Les accès au travers de portails applicatifs facilitent le design d'implémentation des datacenters en positionnant ces serveurs dans des zones de sécurité dédiées.

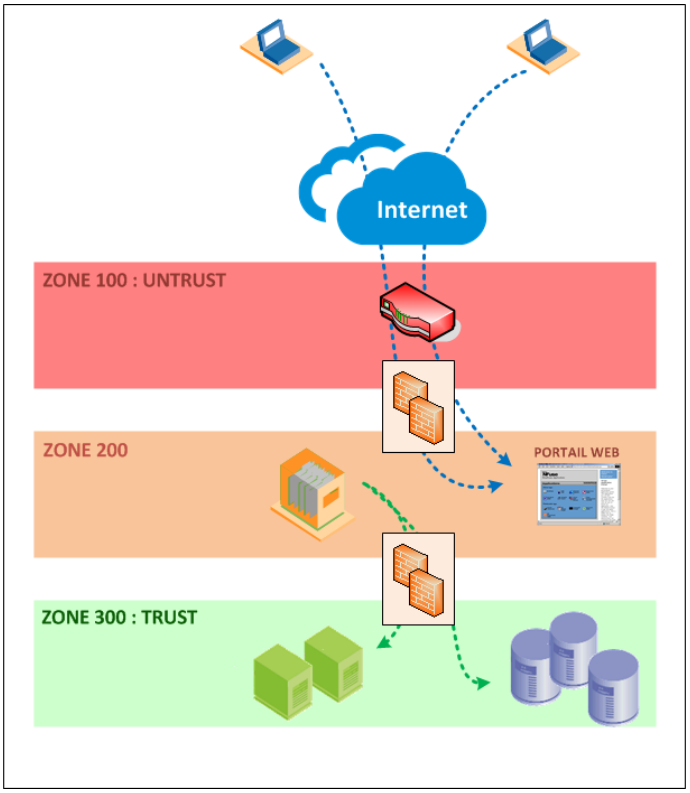

Figure 21 - Accès portail web

De telles implémentations permettent par ailleurs une rupture protocolaire entre le flux en provenance du client et l'infrastructure cœur du datacenter. L'objectif est de séparer physiquement le rendu applicatif client et le traitement interne des données, cette séparation permettant de garantir les exigences de sécurité élémentaires requises par le datacenter (telle l'isolation des clients entre eux, ou la protection du cœur du datacenter vis-à-vis d'attaques externes).
L'Internet des objets semble, pour l'heure, moins concerné par cette généralisation de l'utilisation du protocole HTTP. Il est toutefois commun d'utiliser SOAP au-dessus d'une couche applicative implémentée en HTTP, pour éviter les problèmes posés par la traversée des pare-feux et des proxies. De la même manière il est tout à fait envisageable de considérer des architectures REST s'appuyer sur HTTP pour les échanges entre client et serveur.

3.1.1 - Promotion de l'accès satellitaire pour pallier les contraintes filaires

De nombreuses démarches actuelles visent à élargir la couverture et à fournir au plus grand nombre l'accès à Internet. Dans le cadre du Plan France Très Haut Débit, le projet « Connect'Ecoles » vise à faire bénéficier de connexions satellite ou WiFi 9000 écoles et établissements du second degré non ou mal desservis par les accès Internet filaires.
Si les différents pays européens, ainsi que l'Union Européenne consacrent depuis plusieurs années des budgets à l'élargissement de cette couverture, les opérateurs d'accès se positionnent ouvertement sur ces marchés. Selon Satellite Solutions Worldwide/Europasat, en Europe, 15% des foyers n'ont pas l'accès à Internet en haut débit filaire. L'accès à Internet est devenu incontournable non seulement pour les professionnels mais aussi pour les utilisateurs individuels pour lesquels la connectivité à l'Internet mondial est l'accès incontournable aux différents services administratifs, culturels, commerciaux voire sociaux. Les revenus potentiels pour les opérateurs sont importants et les encouragent à élargir leurs offres de couverture.
A titre d'exemple, KA-SAT propose aujourd'hui son offre Sat2Way aux particuliers pour un accès Internet haut débit par satellite. Plusieurs formules sont disponibles, dimensionnées par le type de modem utilisé et la configuration de la liaison associée. En juillet 2015, victime de son succès, KA-SAT, a annoncé suspendre la commercialisation de ses offres dans 28 départements.

3.1.2 - Positionnement des fournisseurs d'accès et des fournisseurs de services

Pour se raccorder à Internet, il est nécessaire de souscrire un contrat avec un fournisseur de service Internet (ISP Internet Service Provider). Cette connectivité au réseau de l'ISP permet de faire partie de son réseau global, et à ce titre d'être en visibilité sur l'Internet mondial grâce aux connectivités générales de l'ISP. A ce titre les fournisseurs d'accès satellite (SAP/Satellite Access Provider) déploient des réseaux de SATCOM interconnectés vers un ou plusieurs ancrages, eux-mêmes raccordés aux NSP (Network Service Providers). Les points de raccordement entre l'infrastructure d'accès du NSP et le cœur de réseau, ou backbone, de l'ISP se fait au travers de POP (Points of Presence). L'offre globale d'un ISP peut alors consister en un réseau d'accès terrestre sous la responsabilité du NSP raccordé au cœur de réseau, ou backbone de l'ISP, ce qui octroie ainsi aux utilisateurs les droits et moyens d'accéder aux ressources et contenu applicatif général. L'extension d'accès au moyen d'une offre satellitaire intégrée au portefeuille de l'ISP ou alors sous traitée à un SAP attitré permet d'assurer une couverture la plus large possible.

L'explosion de la bulle Internet au début des années 2000 a vu de nombreuses fusions et restructurations de l'écosystème des acteurs proposant des infrastructures réseau et des connectivités à Internet au grand public ainsi qu'aux entreprises. La frontière entre fournisseur de service et fournisseur d'accès est alors devenue assez opaque, chaque acteur ayant progressivement diversifié et élargi son champs d'action pour survivre économiquement.
Les fournisseurs de services proposent des accès à Internet, généralement en contrepartie d'un abonnement, le tout accompagné d'une campagne publicitaire ciblée promouvant un téléchargement automatique et incitatif aux applications de la société. Les fournisseurs de services typés réseaux sociaux sont devenus ces dernières années les acteurs incontournables du marché de l'Internet et, sous couvert d'une philanthropie affichée d'offrir un raccordement émancipateur à Internet, peuvent ainsi, grâce aux liaisons satellitaires, toucher des dizaines de millions d'utilisateurs supplémentaires.
Eutelsat et Facebook ont par exemple annoncé début octobre leur décision de construire un système satellite haut débit en utilisant l'intégralité des capacités de la charge utile broadband du futur satellite géostationnaire AMOS-6. L'objectif est d'apporter la connectivité à Internet aux régions africaines dépourvues d'accès et de fait privées des bénéfices économiques et sociaux d'Internet.

3.1.3 - Evolution de l'offre satellitaire pour accéder à Internet

Si l'accès se fait à ce jour à travers l'utilisation de satellites GEO, les constellations vont gommer les inconvénients inhérents aux liaisons géostationnaires et rendre d'autant plus attractifs les accès satellitaires. Néanmoins à ce jour les communications sur des liens satellite GEO peuvent être optimisées avec la gestion statistique et l'usage prédictible de l'accès au contenu par les utilisateurs. L'exploitation des données des utilisateurs permet de définir des profils d'utilisation statistiques de l'accès Internet et aux ressources. Il est ainsi possible d'optimiser les architectures système pour dimensionner et approvisionner en contenu les serveurs de caching CDN en tenant compte des plages horaires et des contraintes d'utilisation des différents medias physiques constitutifs de l'architecture du réseau de l'opérateur.

3.2 - Définition de scénarii d'utilisation

3.2.1 - Objectifs

Dans la perspective de déterminer les scénarii qui seront étudiés et mis en place au cours de cette étude, il s'agit dans un premier temps de dégager des cas d'utilisation du protocole http2 pertinents selon plusieurs axes, à partir des problématiques décrites dans l'état de l'art :

- Mise en évidence de l'apport de cette solution protocolaire vis à vis du problème dit de « *Head-of-line Blocking* » en ciblant des cas où un ensemble ordonné de ressources est nécessaire pour une fonction donnée et où un rapatriement en parallèle de ces dernières permettrait de gagner en efficacité ;

- Popularité en termes d'utilisation courante pour cibler les besoins d'un large public afin de rester représentatif des usages et de leur correspondance en volume de trafic sur les réseaux ;

- Pertinence pour des problématiques plus orientées métiers du domaine satellitaire (Télécommunications, Observation…) afin de promouvoir cette solution auprès de futures architectures systèmes.

3.2.2 - Architecture cible

Cette R&T étant axée sur l'utilisation de la version 2.0 du protocole HTTP, il s'agit de montrer comment l'utilisation typique de HTTP au travers de liaisons satellitaires va profiter de cette évolution du protocole. A cet égard, il conviendra de comparer les performances du protocole dans ses différentes versions, mais aussi de montrer comment cette évolution devrait permettre au satellite de rivaliser avec les communications filaires et d'inscrire le satellite comme un medium pouvant être utilisé de façon quasi transparente par les opérateurs.
Au final, il nous semble important de privilégier un environnement de test représentatif d'une architecture assez ouverte où le satellite est utilisé comme un lien d'accès à des zones peu ou mal desservies par les opérateurs terrestres. Sur la base de cette architecture, du type multi-gateway, il faudra pouvoir émuler différentes applications représentatives de l'utilisation du protocole HTTP. Les utilisateurs simulés utiliseront de manière comparative soit des accès émulés type ADSL, soit des liens type SATCOM (ces liaisons seront modélisées à l'aide de l'outil OpenSAND), pour accéder à du contenu web chez un opérateur de service.
Dans cet environnement, les liaisons satellitaires permettront aux opérateurs de couvrir des zones difficilement accessibles par d'autres moyens, mais aussi de fournir des solutions pour répondre à des besoins particuliers. En effet, pour la fourniture d'accès Internet domestiques ou professionnel, les liaisons satellitaires peuvent tout à fait suppléer les accès filaires dans des régions rurales mal desservies par les réseaux opérateur ADSL. Mais les capacités de mobilité que présentent les accès Internet par satellite, sont aussi intéressantes pour tout scénario nécessitant un déport rapide d'une infrastructure raccordée à Internet :

- Certains événements culturels ou sportifs importants nécessitent une connectivité avec de grosses contraintes en termes de débit sur une période assez courte. L'objectif est alors de fournir aux visiteurs des moyens pour se connecter à Internet pendant toute la durée de l'événement. L'utilisation consiste essentiellement à fournir un accès Internet classique à un grand nombre d'utilisateurs, par exemple sur la base d'une authentification via un accès sans fil type WiFi au point de connectivité satellite ;

- Dans le cas d'une catastrophe naturelle, des soins doivent être apportés au plus tôt et une connectivité à Internet est alors nécessaire pour fournir un vecteur de communication entre les secours sur place et les différents hôpitaux ou administrations concernés. On peut imaginer des accès à des portails de références dédiés à ces urgences pour le personnel médical et plus généralement le personnel d'intervention (pompiers, gendarmerie) ainsi que les organes de presse ;

- Des besoins de connectivité peuvent aussi se présenter pour des utilisateurs uniques, ne nécessitant pas une agrégation d'utilisateurs sur un même point, mais un moyen dédié, par exemple une voiture de presse ou une ambulance. Ces utilisateurs ayant seulement besoin d'établir une connexion à leur organe de rattachement.

Ces cas d'utilisation sont des exemples caractéristiques montrant que le satellite peut s'avérer un constituant essentiel d'un réseau opérateur. En effet, le satellite permet d'adresser des problématiques bien spécifiques pour lesquelles les accès filaires n'offrent pas suffisamment de flexibilité. En premier lieu la couverture satellite permet d'assurer un service mobile, dans le sens où les récepteurs peuvent bouger et ne sont pas assujettis à un emplacement physique précis. Si la mobilité est une caractéristique majeure des services fournis par des raccordements 4G voire 5G, les liaisons SATCOM présentent l'avantage d'être indépendantes des infrastructures terrestres.

Ainsi en cas de catastrophe naturelle, le recours au satellite reste le moyen le plus flexible à déployer rapidement. D'un point de vue opérateur, le déploiement des infrastructures SATCOM présente l'avantage majeur de proposer une large couverture pour un coût de déploiement constant, indépendant du relief et de la géographie. En d'autres termes, hormis les coûts de construction, de lancement et de gestion du satellite, le plan de financement pour le déploiement des stations sol utilisateurs (SSU) est complètement prédictible. Les deux derniers atouts majeurs sont la sécurité et la qualité de service offertes par ce type de liaisons, des atouts indéniables pour des communications sensibles.
Il faudra au final s'attacher à vérifier que les liaisons satellitaires pourront, à terme, faire partie intégrante de l'offre de couverture d'accès des opérateurs. Le ressenti utilisateur doit être sensiblement le même quel que soit le médium physique permettant d'accéder à l'information.
Nous proposons ainsi de nous concentrer sur une architecture multi gateway du type suivant :

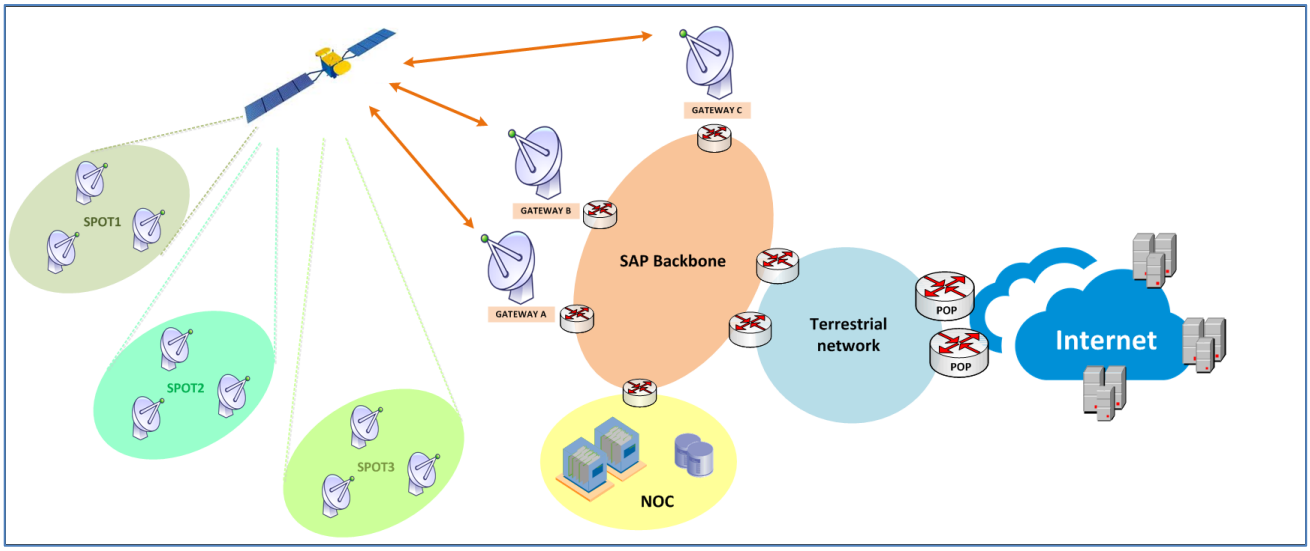

Figure 22 - Architecture ciblée

Dans ce modèle, les SSU sont raccordées à des gateways selon leur spot de couverture. Le fournisseur d'accès satellite traite toutes les données sur son backbone propre. Ce backbone connecte toutes les gateways entre elles, rendant ainsi possibles les communications entre SSU. Le backbone est par ailleurs rattaché à un réseau terrestre d'accès fournissant à l'ISP les moyens de relier le réseau d'accès satellite à Internet et à ses ressources propres (en termes d'applications et de services). Ainsi il est possible pour les utilisateurs des SSU d'accéder au contenu des serveurs HTTP quelle que soit leur localisation.
Cette base d'architecture permettra d'une part de comparer les performances du protocole http2 pour des utilisateurs dont l'accès est fourni au travers d'un accès satellite avec celles d'utilisateurs raccordés directement sur le réseau terrestre. Les comparaisons et modélisations pourront s'effectuer sur les différentes applications envisagées dans les scénarios.
Les SSU hébergent les clients utilisateurs des applications HTTP, alors que les serveurs sont localisés sur Internet. Dans cet environnement il sera possible de comparer les performances des accès HTTP pour des clients hébergés sur des SSU (couvert par le réseau d'accès satellite), avec celles des clients raccordés directement sur le réseau terrestre. En raison des contraintes de chiffrement activées par défaut pour http2, nous aurons vraisemblablement intérêt à fonctionner avec une plateforme fermée sur elle-même, en essayant de simuler les accès Internet avec des serveurs que nous implémenterons nous-mêmes. Le contrôle de la chaîne de bout en bout nous fournira une plus grande flexibilité dans nos tests et permettra d'éliminer un maximum d'inconnues.

3.2.3 - Tour d'horizon des besoins

Afin de dégager des scénarios de tests pertinents pour cette étude, une réflexion a été menée sur les cas d'utilisation du HTTP.

Parmi les usages les plus courants, citons des applications telles que :

- Navigation web (HTTP/HTTPS), le plus classique ;
- Streaming audio et/ou vidéo adaptatif (HLS/MPEG-DASH), où une ressource donnée est découpée en segments pouvant être transmis en parallèle pour alimenter une mémoire tampon ;
- Messagerie instantanée (XMPP over HTTP) ou réseaux domestiques (UPnP), consistant souvent en du transport de flux XML ;
- Tunnels HTTP, pour passage de pare-feux ;
- Cartographie (API Google Map, OpenGIS, OpenLayers…), où une image est découpée en tuiles pouvant être récupérées en parallèle ;
- Rapatriement de données via un portail/catalogue Web ;
- Architecture multi-tiers telles SOA (Service Oriented Architecture) / WOA (Web Oriented Architecture) via l'utilisation de SOAP/JSON encapsulé dans du HTTP ;

On peut en outre répertorier le besoin émergent à fort potentiel qu'est l'Internet of Things (IoT) et son extension Web of Things (WoT) qui intègre des objets suffisamment intelligents pour pouvoir gérer la couche applicative et donc HTTP.

Parmi les différents cas énumérés plus haut, certaines utilisations de HTTP semblent peu enclines à être améliorées, comme :

- Les architectures multi-tiers (utilisation de SOAP), dont la philosophie actuelle repose sur le principe « une fonction = un webservice » et dont le questionnement se résume en l'envoi d'un XML dans un GET et une réponse sous forme d'XML, soit le rapatriement d'une seule ressource à la fois ;
- Des outils de supervision (idem pour l'IoT), collectant une ressource simple (comme un statut) pour chaque élément interrogé, le seul intérêt pouvant résider dans l'élimination des multiples ouvertures de connexion grâce au multiplexage de flux dans une connexion TCP unique.

3.3 - Conclusion : scénarii envisagés

Nous avons identifié 4 applications qui seront analysées en détail et maquettées dans la suite de l'étude. Les architectures cibles seront définies en accord avec le CNES, le chapitre qui suit présentant essentiellement le contexte applicatif des scénarios envisagés.

3.3.1 - Navigation web

La recrudescence du nombre d'objets dans une page web à l'heure actuelle pose des problèmes, notamment dans les réseaux à forte latence et/ou fort taux d'erreur tels que le satellite. Le paradigme requêtes/réponses séquentielles du http1.x n'est plus adapté. Pour remédier à ce problème, certains sites web utilisent différentes techniques comme la concaténation d'objets (images, scripts) ou encore le *domain sharding*, c'est-à-dire la répartition des différentes ressources sur des noms de domaines différents, et par extension des connexions TCP différentes.

Au premier abord, le *domain sharding* semble redondant avec le multiplexage des connexions offert par http2. Cependant, comme suggéré dans la partie 2.6.3 - , l'utilisation d'une seule connexion http2 en contexte satellitaire déplace les problèmes au niveau de la couche transport (TCP). Le *domain sharding* semble donc complémentaire à l'utilisation du http2 dans ce cas de figure.
Il sera donc intéressant de mesurer les apports réels de http2, notamment en fonction du nombre de sessions TCP utilisées.

Le scénario envisagé devra étudier la mise à disposition sur un serveur d'une ou plusieurs pages web classiques composées de plusieurs éléments (HTML, CSS, javascript, images…). Le serveur sera accessible au travers d'une liaison satellite simulée par l'outil OpenSAND, comme représenté sur la figure ci-dessous. Une comparaison entre http1.x et http2 sera ensuite réalisée.

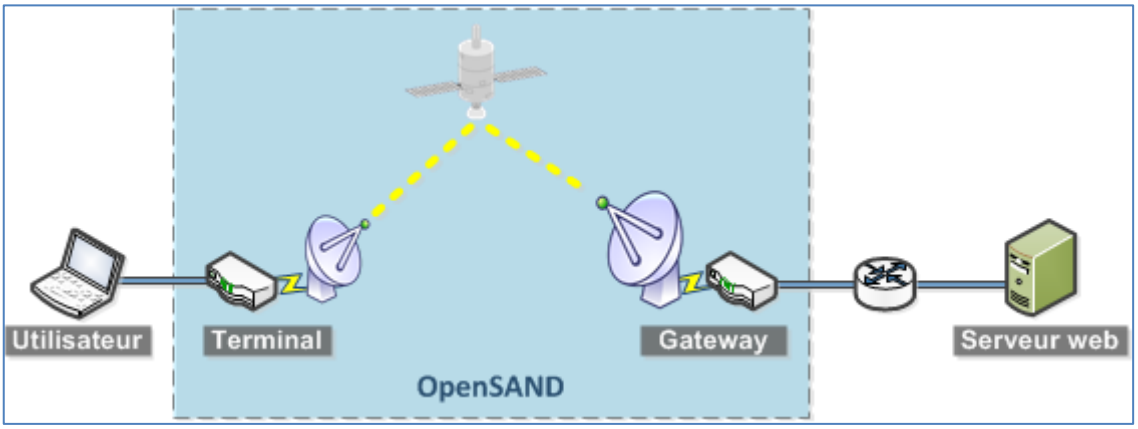

Figure 23 - Architecture applicative

3.3.2 - Streaming adaptatif

Le problème du H*ead-of-line Blocking* se fait aussi ressentir dans le domaine du streaming vidéo. Bien que le *pipelining* de http/1.1 puisse sembler intéressant dans le cas de la transmission de multiple segment vidéo en parallèle, cette fonction n'est en pratique pas utilisée ; un segment vidéo perdu peut en effet retarder le reste de la chaîne.

Tout comme la navigation web, le multiplexage de flux http2 semble prometteur dans le cadre du streaming vidéo. La fonctionnalité *push*, si utilisée par le serveur pour envoyer de manière préventive des données, peut améliorer l'expérience utilisateur en remplissant le buffer d'une vidéo plus efficacement.

On définira pour les besoins de l'étude une architecture similaire à celle analysée dans le scénario précédent, le serveur fournissant des flux vidéo type HLS ou MPEG-DASH.

3.3.3 - Cartographie

L'utilisation d'http1.x pour des services de cartographie au travers d'une liaison satellite est source de frustration pour l'utilisateur final ; les différentes tuiles d'une carte étant délivrées une par une, le rendu global de l'image s'en retrouve retardé au travers d'une liaison à forte latence.

Le téléchargement en parallèle des différentes tuiles d'une carte au sein d'une même connexion TCP, fournie par le multiplexage http2, promet une amélioration significative pour l'expérience utilisateur au travers d'une liaison satellite typique.

Un serveur de cartographie sera mis à disposition dans l'architecture ciblée par les scénarios précédents.

3.3.4 - Collecte de données/API REST

Comme vu dans la partie 2.6.4.1 - , http dans sa version 1.1 n'est pas très adaptée à l'IoT, aussi bien du côté ressources machines que du côté des caractéristiques des réseaux d'accès (faible débit, latence, coût au bit transmis).

Le format binaire et la compression des headers dans les messages http2 permettent des avancées non négligeables sur les utilisations mémoire et bande passante du protocole. Ces deux points ouvrent de nouvelles possibilités dans le domaine de l'IoT et il est donc intéressant de quantifier les apports de http2 vis-à-vis de http1 dans le domaine de la collecte de données.

Un serveur accessible au travers d'une API REST sera mis en place au travers d'une liaison satellite simulée.

4 - CONCLUSION

Cette partie sera définie à l'issue des échanges que nous aurons lors de la réunion de présentation du lot 1, afin d'orienter la suite de l'étude dans une direction agréée par tous.