PROTOCOLE DE COMMUNICATION

Exemple: HTTP

- Fonctionnement et vérification
- Programmation d’une communication en HTTP

Protocoles de communication

- Protocoles de bas niveau:
  - *Assurent les connexions et le contrôle des transfert des données et fournissent les services de liaisons*
    - TCP: une partie du protocole Internet TCP/IP qui garantit la remise des données en séquence.
    - UDP: permet d'établir des connexions à tolérance de perte
    - IP: est la partie du protocole Internet TCP/IP qui achemine et route les paquets
    - Etc.

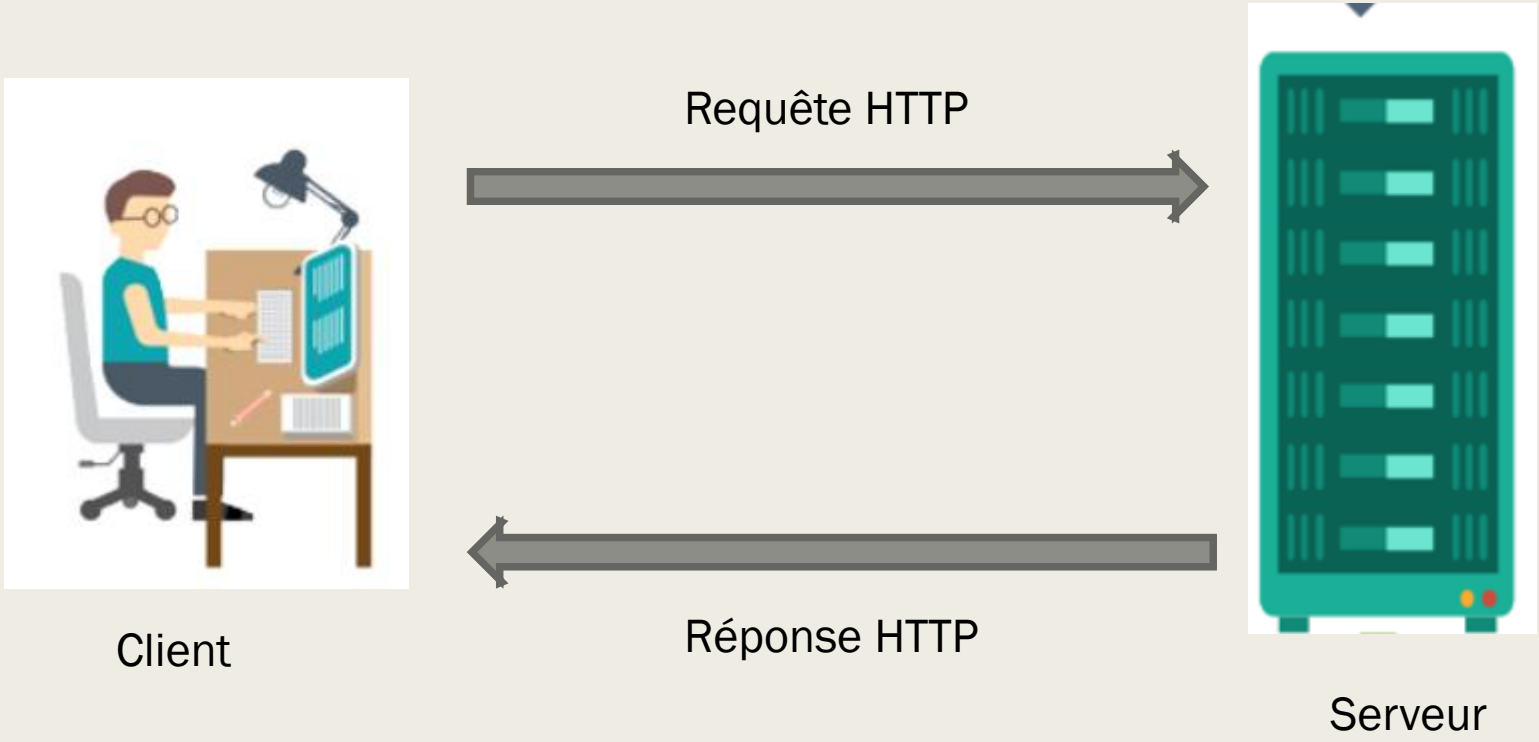
- Protocoles de haut niveau
  - Les protocoles de la catégorie APPLICATION garantissent l'interaction et l'échange des données :
    - HTTP: assure la communication entre un client (exemple: navigateur) et un serveur sur le World Wide Web (WWW).
    - FTP: (File Transfer Protocol) permet le transfert de fichiers.
    - SMTP: (simple Mail Transfer Protocol) est un protocole Internet pour l'envoie de messagers électronique.
    - POP3 et IMAP: permettent de se connecter à un serveur de messagerie et de récupérer son courrier électronique
    - SNMP (Simple Network Management Protocol) permet au administrateurs réseaux de gérer les équipements de leur réseau.
    - TELNET permet de communiquer avec un serveur distant en échangeant des lignes de texte et en recevant des réponses également sous forme de texte.
    - Etc.

Protocole HTTP

- HTTP signifie HyperText Transfer Protocol
- Il s’agit d’une norme régissant les échanges entre un client web (un navigateur par exemple) et un serveur web
- HTTP est un protocole qui utilise une liaison TCP pour la communication entre le client et le serveur

Communication HTTP

- Un scénario classique de communication HTTP commence avec une requête envoyée par le client et se termine par une réponse du serveur



Communication HTTP (suite)

1- Une URL d'un site web commençant par http est saisie dans un navigateur (Chrome, Firefox, etc)

- Exemple: http://java.sun.com/index.html

2- Le navigateur examine la partie de l'URL entre le double slash et le slash simple

- java.sun.com
- Ceci représente la machine au quel l'utilisateur veut se connecter
- Le navigateur envoie une requête au serveur DNS pour obtenir l'adresse IP de java.sun.com

3- À l'aide du préfixe http, le navigateur conclut qu'il faut utiliser le protocole HTTP et en conséquence le port 80

- *Il établit une connexion TCP/IP au port 80 avec l'adresse IP*
- *Examine la partie restante de l'URL (/index.html) et envoie une requête HTTP au serveur pour demander de renvoyer cette page*

4- Le serveur web reçoit la requête du navigateur et la décode

- Il cherche le fichier index.html et le renvoie vers le navigateur à l'aide d'une réponse HTTP

5- le navigateur affiche le contenu du fichier à l'utilisateur

- Si c'est un fichier HTML, alors il traduit le code HTML et affiche son contenu

Format d’une requête/réponse HTTP

- Les requêtes et les réponses HTTP ont une structure semblable, composée par:
  1. *Une ligne de début*
  2. *Des en-têtes HTTP*
  3. *Une ligne vide*
  4. *Le corps*

Format d'une requête/réponse HTTP (suite)

- Les requêtes et les réponses HTTP sont formées par:

1. Une ligne de début:

- La composition de la ligne de début change dépendamment s'il s'agit d'une requête ou d'une réponse
- Remarque:
  - *La ligne de début doit s'écrire sur une seule ligne et doit finir par un retour chariot suivi par un retour à la ligne (\r\n)*

A. La ligne de début d'une requête HTTP:

- *indique la commande à envoyer au serveur.*
- *Cette ligne est formée par: Méthode URL versionHTTP*
- *Plusieurs méthodes possibles:*
  - GET : les requêtes utilisant cette méthode permettent de retrouver une ressource sur le serveur
  - POST: Ces requêtes permettent d'envoyer des données vers le serveur et de modifier la ressource
  - Liste de toutes les méthodes: ici
- *Exemple: GET content/page1.html HTTP/1.1*
  - demande une ressource appelée content/page1.html du serveur

Format d'une requête/réponse HTTP (suite)

B. La ligne de début d'une réponse HTTP:

- *indique si la demande a pu être satisfaite*
- *Cette ligne est formée par : VersionHTTP Code-d'état Texte-réponse*
  - VersionHTTP : la version HTTP du serveur
  - Code d'état : le code retourné par le serveur pour indiquer le succès ou l'échec de la requête (par exemple 200, 403, 404, 500)
  - Texte-réponse : le texte associé au code (respectivement pour les exemples précédents : "OK", "FORBIDDEN", "NOT FOUND", "INTERNAL ERROR").
- *Exemple: HTTP/1.1 200 OK*

Format d'une requête/réponse HTTP (suite)

B. La ligne de début d'une réponse HTTP (suite):

- Les codes d'état sont des standards définis par Internet Engineering Task Force (IETF):
  - *1xx : requête reçue et en cours de traitement*
  - *2xx : requête bien reçue par le serveur et acceptée*
  - *3xx : accès impossible, redirection nécessaire*
  - *4xx : erreurs liées au client*
  - *5xx : erreurs liées au serveur*

- Liste de tous les codes: ici
- Remarques: Les codes peuvent être utiles pour le diagnostic des problèmes et pour la vérification du fonctionnement de la communication

Code	Signification
200	Opération effectuée avec succès
202	Requête acceptée mais sont traitement n'est pas encore terminée
301	Le document réclamé a été déplacé et se trouve maintenant dans une autre adresse mentionnée dans la réponse
400	Le serveur n'a pas compris la requête (exemple: erreur de syntaxe)
501	Le serveur ne prend pas en charge la méthode de la requête

Format d'une requête/réponse HTTP (suite)

- Les requêtes et les réponses HTTP sont formées par:

  2. Les en-têtes:

  – *permettent au serveur (et au client) de passer des informations additionnelles avec la requête (ou la réponse)*

  – *S'écrit sous la forme: Nom: valeur*

  – *Voici quelques-unes :*

    - Connection: Définit le type de connexion (réponse ou requête)
      - **Close** : la connexion est fermée après la réponse
      - **Keep-Alive** : crée une connexion persistante. Avec ce type de connexion, il est même possible d'envoyer une requête sans attendre la réponse à la précédente.
      - Exemple: Connection: keep-alive
    - Content-type: Spécifie le type du corps de réponse ou de requête (cas de de l'envoie d'un formulaire dans une requête)
      - *text/html*
      - *text/javascript1.0*
      - *Etc*
      - *Exemple:* Content-Type: text/html
    - Content-Length : Spécifie la longueur du corps en octet (réponse ou requête)
      - *Exemple:* Content-Length: 348
    - Accept : formats acceptés par le client (requête)
      - Exemple: Accept: text/html

- *La liste de toutes les en-têtes: ici*

  – *Remarque:*

    - Chaque entête doit s'écrire sur une seule ligne et doit finir par un retour chariot suivi par un retour à la ligne (\r\n)

Format d'une requête/réponse HTTP (suite)

Exemples d'entête d'une requête et de sa réponse:

- Envoi des informations d'entête de la requête: informer le serveur:
  - *Configuration du client*
  - *Documents acceptés*

```
User-Agent: Mozilla/5.0 (compatible;MSIE 6.0;Windows NT 5.1)
Host: www710.univ-lyon1.fr
Accept: image/gif, image/jpeg
```

- Envoi des informations d'entête de la réponse: informer le client:

```
Date: Tue, 30 Sep 2008 06:11:28 GMT
Server: Apache/1.3.34 (Debian) PHP/5.2.1
Last-Modified: Tue, 30 Sep 2008 06:11:14 GMT
ETag: "600593b3-61-48e1c302"
Accept-Ranges: bytes
Content-Length: 97
Content-Type: text/html; charset=iso-8859-1
```

Format d'une requête/réponse HTTP (suite)

- Les requêtes et les réponses HTTP sont formées par

3. Une ligne vide:

  - *Permet d'indiquer la fin de l'entête et de la séparer du corps*
  - *Représentée par un retour chariot suivi par un saut de ligne* \r\n

Format d’une requête/réponse HTTP (suite)

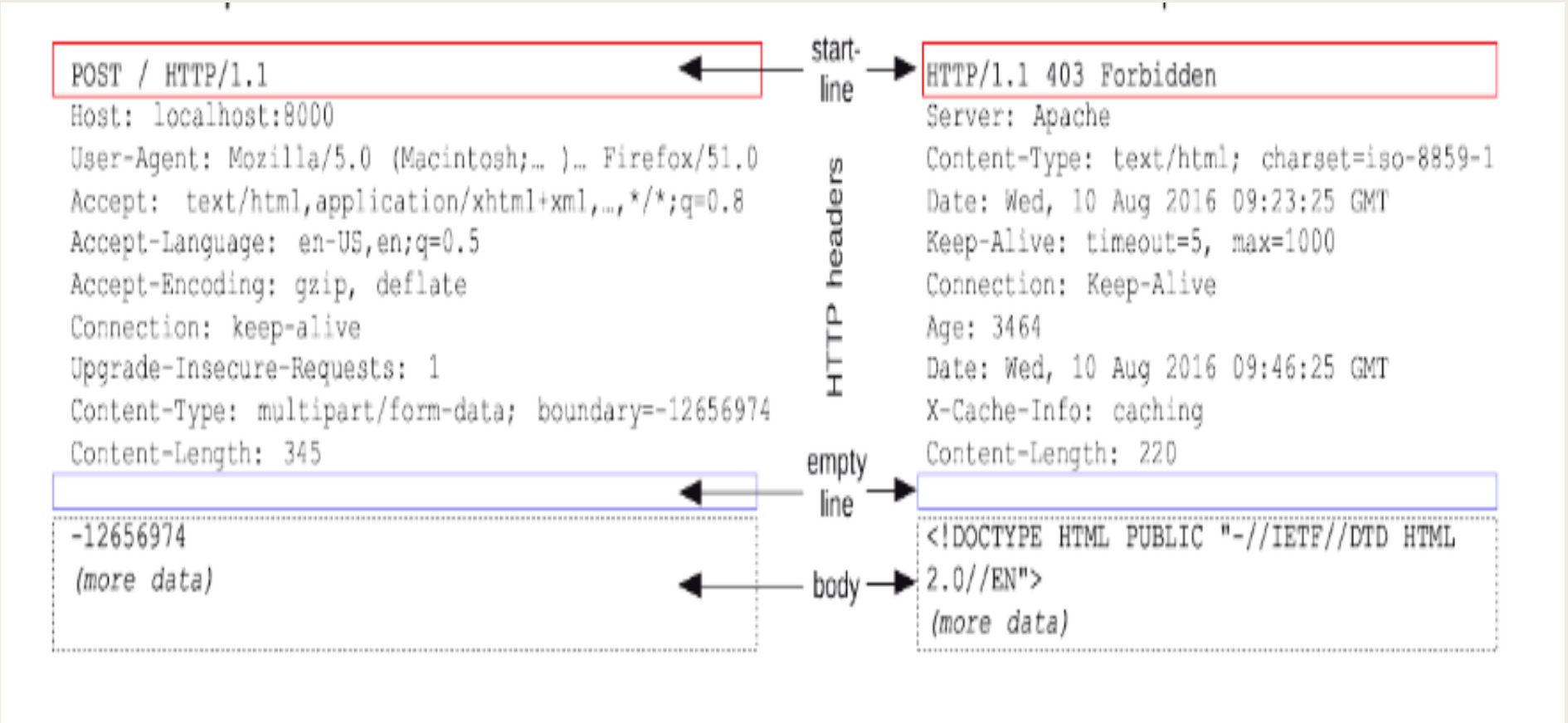
- Les requêtes et les réponses HTTP sont formées par

4. Le corps:

  - *Pour terminer la requête, on envoie le corps de requête. Il peut contenir, par exemple, le contenu d'un formulaire HTML envoyé en **POST**.*
  - *Dans le cas d’une réponse HTTP, le corps peut contenir* le contenu du fichier, le HTML d'une page par exemple
  - *Le corps est optionnel. Il peut être vide*

Format d'une requête/réponse HTTP (suite)

Exemple complet d'une requête et de sa réponse:



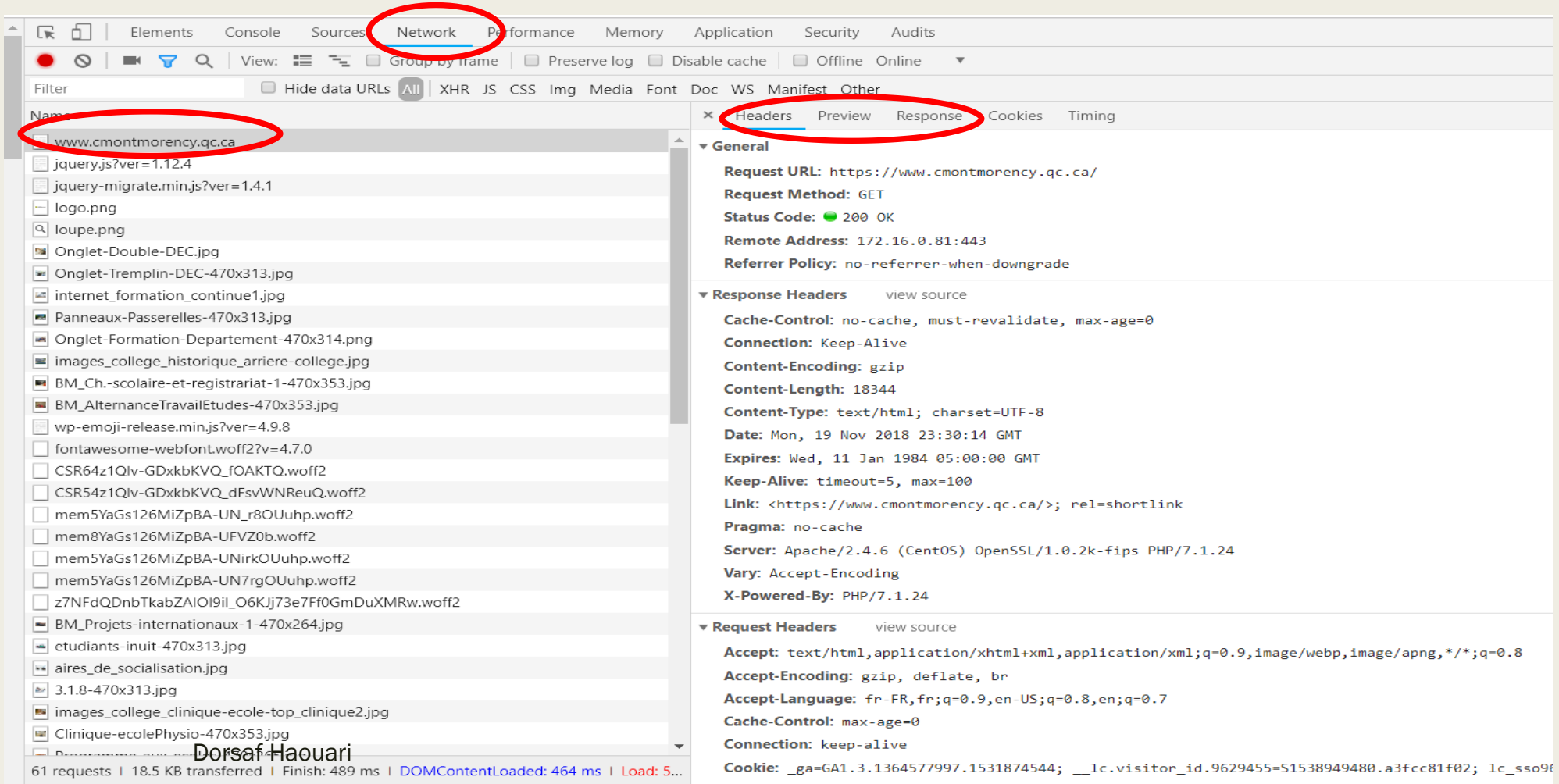
https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Vérification des requêtes et des réponses à partir du navigateur

1. Ouvrir une page dans le navigateur Chrome (www.cmontmorency.qc.ca)
2. Ouvrir la console du développeur:
   - *Clic droit – Inspecter*
3. Onglet Network – ensuite recharger la page
   - *La liste des ressources demandées s'affichent:*
     - Fichiers html, js, css, images, etc.
4. Cliquer sur une ressource de la partie gauche et observer les onglets **Headers**, **Preview**, **Response** de la partie droite

- Cet outil permet de visualiser des informations sur une communication HTTP et d'en faire la vérification

Vérification des requêtes et des réponses à partir du navigateur (suite)



Versions HTTP

- Version d'origine : HTTP/0.9
  - *Une seule méthode : GET*
  - *Pas d'en-têtes*
  - *Une requête = une connexion TCP*
- HTTP/1.0 :
  - *introduction des en-têtes*
  - *nouvelles possibilités : utilisation de caches…*
- HTTP/1.1 :
  - *mode de connexions persistantes par défaut*
  - *introduction des serveurs virtuels*
    - la directive Host dans la requête est nécessaire
- HTTP/2: (apparu en 2015 et supporté par tous les navigateurs)
  - *Permet la compression des en-têtes*
  - manière dont la donnée est segmentée et transportée entre le client et les serveurs
  - Gain de vitesse de transmission

Programmation d’une communication en HTTP:

- Exemple: implémentation d’un serveur web et communication en HTTP avec un navigateur

Rappel: Socket

- Un socket est un objet de communication par lequel une application échange des paquets de données avec une autre application à travers un réseau à l'aide des mécanismes d'E/S (java.io)
- Utilise le paradigme client/serveur
- La classe java.net.SocketServer permet de programmer l'interface côté serveur en mode connecté.
- La classe java.net.Socket permet de programmer l'interface côté client et la communication effective par flot via les sockets.

Rappel: Socket
Intéraction des sockets: client/serveur

Le Serveur	Le client
1. **Crée un socket serveur** (port , nombre max connexions) `ServerSocket socketServeur= new ServerSocket (1025,5)`	
2. **Attend une connexion du client par accept().** Cette méthode est bloquante `Socket socketCommunication= socketServeur.accept()` accept() retourne un Socket de communication pour échanger avec le client.	3.**Demande une** connexion au serveur • Crée localement une socket TCP • Spécifie **l'adresse IP** et le numéro de **port** du processus du serveur. • Une fois la socket créé, le TCP du client établit une connexion avec le TCP du serveur `Socket socketClient=new Socket (InetAddress.getByName("192.168.121.14"), 1025);`
4 Utiliser le socket de communication pour envoyer/recevoir des données au client	**4.Utiliser le socket pour envoyer/recevoir des données au serveur**
5-Fermeture du socket de communication `socketCommunication.close();`	**5-Fermeture du socket client** `socketClient.close();`

Exemple1: exemple d'une requête du client

- Voir le package exemple1 fourni en ressource
  - *Création d'un serveur web en écoute*
  - *Le serveur web reçoit une connexion du client et affiche la requête sur la console*
- Instructions d'exécution:
  - *Exécuter la classe HTTPServer.java depuis Eclipse*
  - *Ouvrir un navigateur*
  - *Taper l'adresse http://localhost:8080*
  - *observer la requête du navigateur dans la console d'éclipse*

Exemple2: exemple d'une réponse du serveur

- Voir le package exemple2 fourni en ressource
  - *Le serveur envoie une réponse html au client*
- Instructions d'exécution:
  - *Exécuter la classe HTTPServer.java depuis Eclipse*
  - *Ouvrir un navigateur*
  - *Taper l'adresse http://localhost:8080*
  - *observer la réponse du serveur dans le navigateur*

- Remarque:
  - *Les deux exemples précédents ne sont pas terminés au complet:*
    - Il manque le parseur de la requête pour récupérer le nom de la ressource
    - Retourner la ressource
  - *Ça sera un travail à faire pour le TP*