Application distribuée *via HTTP*
Mise en œuvre minimale ...

Problème de la distribution
Le protocole HTTP

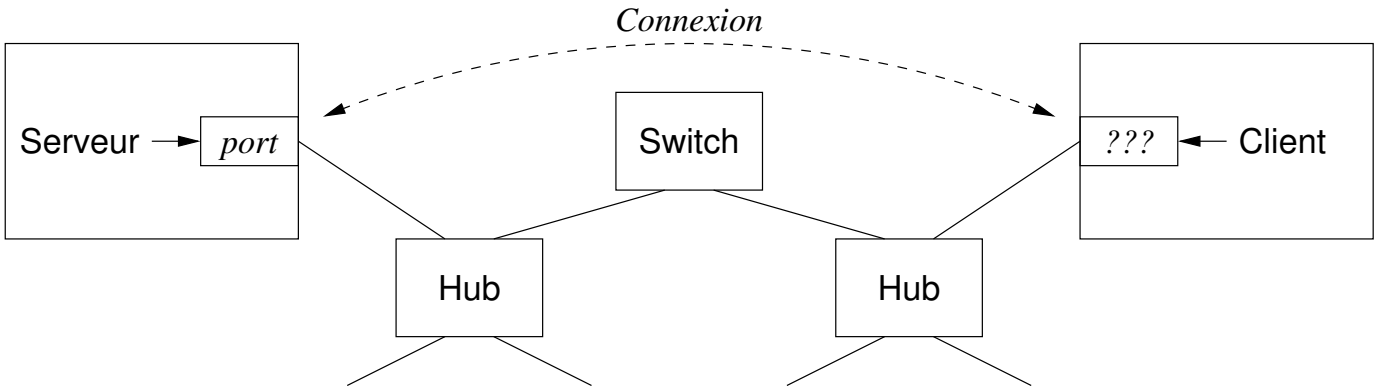
Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Distribuer une application

- **Principe**
  - Des processus sur plusieurs machines
  - Échanger des données applicatives par réseau
    - *TCP* ou *UDP* sur des ports dédiés
    - Réseau local → peu de contraintes
    - *Internet* → contraintes de performances et d'accès
- **Contraintes d'*Internet***
  - Performances → on n'y peut pas grand chose `:-(`
  - Accès aux services
    - Un *serveur* doit pouvoir accepter les connexions d'un *client*
    - Un *client* doit pouvoir se connecter à un *serveur*
    - Les *"relais"* doivent accepter de *"relayer"* la communication

Distribuer une application

- **Situation de départ**
  - Un processus *serveur* écoute sur un port dédié
  - Un processus *client* se connecte sur ce port
  - Dans un réseau local → pas de problème majeur
    - Accès direct par *hubs/switchs*
    - Sous-réseaux avec filtrage par *firewalls*
      → voir éventuellement l'administrateur, une fois pour toutes !



Contraintes liées à *Internet*

▷ **Du côté des *clients***

◇ Ils peuvent être sur plusieurs sites distincts (entreprises, écoles . . .)

◇ Filtrage très probable des services disponibles pour les utilisateurs

○ Logiciels de téléchargement *peer-to-peer*

○ Jeux en réseaux

○ Serveurs *HTTP "pirates"* sur ports non standards ($\neq$80)

○ Virus/*spywares* sur ports dédiés !!!

◇ Autorisation de **sortie** uniquement pour quelques services choisis

○ *HTTP*, *FTP*, *SSH* . . . pour les utilisateurs

○ D'autres pour les services *"administratifs"*

◇ Filtrage de la demande de connexion au *serveur* sur port dédié

○ Ne pas demander une dérogation à chaque administrateur !

○ Réutiliser un port standard $\rightarrow$ le plus courant : 80 (*HTTP*)

Contraintes liées à *Internet*

- **Du côté du *serveur***
  - Filtrage très probable des services accessibles depuis l'extérieur
  - Autorisation d'**entrée** uniquement pour quelques services choisis
    - *DNS*, *SMTP*, *HTTP*, *FTP*, *SSH* ...
    - Chaque service est *a priori* associé à une machine particulière
  - Autoriser l'accès au port 80 de notre machine *serveur*
    - Machine différente du *"vrai" serveur HTTP*
    - Demander à l'administrateur
    - Intervention nécessaire uniquement sur le site du *serveur*
  - En théorie, connexion possible de *clients* quelconques à notre *serveur*
    - Vrai au sens *transport/session* (*OSI*:4/5, *TCP/IP*:3)
    - Généralement faux du point de vue *application* (*OSI*:7, *TCP/IP*:4)

Contraintes liées à *Internet*

- **Retour du côté des sites *clients***
  - Utilisation très probable d'un *serveur mandataire* (*proxy*)
    - Mise en *cache* des pages *web* rapatriées (performances)
    - Constitution de journaux d'activité (*logs*)
    - Interdiction de visite de certains sites (*blacklist*)
    - Filtrage de contenu selon des mots-clefs (raciste, cochon ...)
  - Fonctionnement du *proxy*
    - Le *client* fait sa requête au *proxy*
    - Le *proxy* analyse la requête et la relaye sur *Internet*
    - Le *proxy* analyse la réponse et la relaye vers le *client*
    - *Cache*, *log*, filtrage éventuels dans les étapes de relais

Contraintes liées à *Internet*

▷ **Implications de l'utilisation d'un *proxy***

◇ Connexions directes vers *Internet* sur le port 80 interdites !

◇ Le *client* **doit** passer par le *proxy*

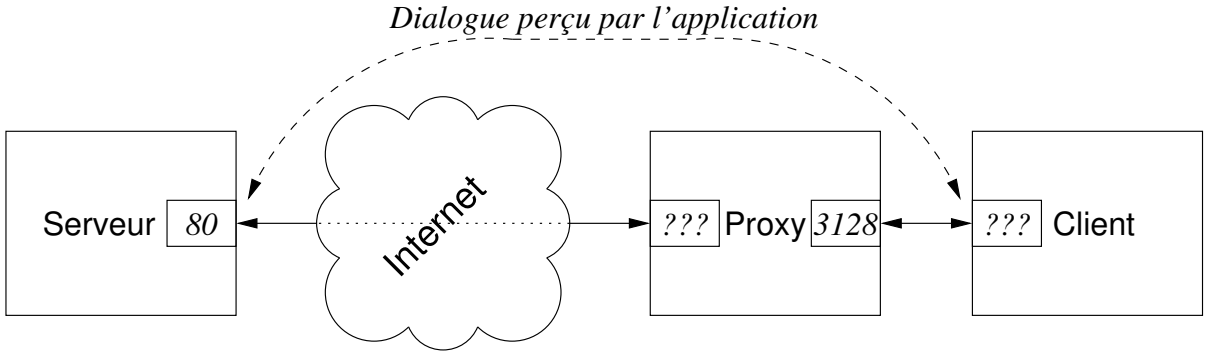
◇ Il ne s'agit pas d'un simple relais de connexion

○ Le *proxy* analyse le contenu des requêtes/réponses

○ La requête du *client* **doit** être compréhensible par le *proxy*

○ La réponse du *serveur* également

→ Encapsuler les données applicatives dans *HTTP*



Hypertext Transfer Protocol

▷ Caractéristiques

- ◇ Protocole applicatif de transfert de données (1990)
- ◇ Décrit par la *RFC-1945* (1.0) et les *RFC-2068* et *RFC-2616* (1.1)
- ◇ À l'origine pour des documents de type *hypertexte*
  - ○ Dans la pratique utilisable pour des données quelconques
- ◇ Un entête lisible en mode texte (lignes terminées par `\n` ou `\r\n`)
  - ○ Une première ligne de requête ou de réponse
  - ○ Des champs d'information optionnels (1 par ligne)
  - ○ Une ligne vide suivie des données à transmettre
- ◇ Les opérations usuelles sont faciles à mettre en œuvre
  - ○ Les données ne sont pas nécessairement des pages *web*
  - ○ Le *serveur* n'est pas nécessairement un *serveur web*
  - ○ Le *client* n'est pas nécessairement un *navigateur web*

L'entête *HTTP*

▷ Structure générale d'une requête

```
méthode ressource HTTP/1.X                    (1.0 ou 1.1)
option1 : valeur1
...
optionN : valeurN
                          (ligne vide → fin de l'entête)
... données éventuelles à transmettre ...
```

▷ Structure générale d'une réponse

```
HTTP/1.X status description                   (1.0 ou 1.1)
option1 : valeur1
...
optionN : valeurN
                          (ligne vide → fin de l'entête)
... données éventuelles à transmettre ...
```

L'entête *HTTP*

- **Les méthodes des requêtes**
  - `GET` : réclamer le contenu de la ***ressource*** (fichier ...)
  - `HEAD` : réclamer juste l'entête de la ***ressource*** (taille, date ...)
  - `POST` : fournir des données à la ***ressource*** (*cgi*, *php* ...)
  - `PUT`, `DELETE`, `TRACE`, `CONNECT` : *HTTP/1.1*, non traitées ici
- **Les *status/descriptions* des réponses**
  - `1`***XX*** : information
  - `2`***XX*** : succès (`200 OK`, `201 Created` ...)
  - `3`***XX*** : redirection (`301 Moved Permanently` ...)
  - `4`***XX*** : erreur du *client* (`403 Forbidden`, `404 Not Found` ...)
  - `5`***XX*** : erreur du *serveur* (`501 Not Implemented` ...)

L'entête *HTTP*

▷ **Les informations optionnelles**

- ◇ `Content-Length` : taille des données (en décimal) après l'entête
- ◇ `Content-Type` : nature des données transmises (type ***mime***)
  (`text/html`, `text/plain`, `image/gif`, `application/pdf` ...)
- ◇ `Expires` : date d'expiration des données (gestion du *cache*)
- ◇ `Host` : *machine*:*port* du *serveur* où le *client* souhaite se connecter
- ◇ `Server` : identification/version du *serveur* délivrant la réponse
- ◇ `User-Agent` : identification/version du *client* émettant la requête
- ◇ De nombreuses autres, non traitées ici
- ◇ Quelques unes sont requises pour certaines opérations

Exemple de requête `GET`

▷ **Obtenir le contenu de** `http://www.ietf.org/rfc.html`

◇ Le *client* se connecte et envoie la requête :

```
GET /rfc.html HTTP/1.1
Host: www.ietf.org:80
Connection: close
```

(ligne vide)

◇ Le *serveur* accepte la connexion, analyse la requête et répond :

```
HTTP/1.1 200 OK
Date: Fri, 19 Nov 2004 17:51:29 GMT
Server: Apache/2.0.46 (Red Hat)
Last-Modified: Thu, 09 Sep 2004 20:29:00 GMT
ETag: "414168-ce5-14a05b00"
Accept-Ranges: bytes
Content-Length: 3301
Connection: close
Content-Type: text/html; charset=UTF-8
```

(ligne vide)

```
<HTML> ... </HTML>
```

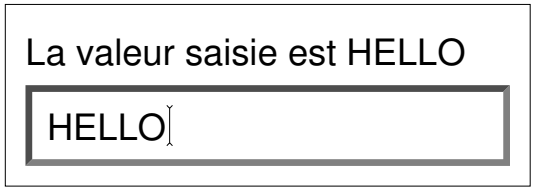
(le serveur ferme la connexion)

Exemple de requête `POST`

▷ **Utiliser le formulaire de** `http://www.cerv.fr/myScript.php`

◇ Le formulaire est le suivant :

```
<?php $value = $_POST['value']; ?>
<html>
La valeur saisie est <?php echo $value; ?><br>
<form name="form" method="post" action="myScript.php">
  <input type="text" name="value"
         value="<?php echo $value; ?>">
</form>
</html>
```



◇ Le *client* se connecte et envoie la requête :

```
POST /myScript.php HTTP/1.1
Host: www.cerv.fr:80
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

value=HELLO
```

(ligne vide)

Exemple de requête `POST`

▷ **Utiliser le formulaire de** `http://www.cerv.fr/myScript.php`

◇ Le *serveur* accepte la connexion, analyse la requête et répond :

```
HTTP/1.1 200 OK
Date: Fri, 19 Nov 2004 19:30:22 GMT
Server: Apache/1.3.31 (Debian GNU/Linux) PHP/4.3.4 ...
X-Powered-By: PHP/4.3.4
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

(ligne vide)

```
<html>
La valeur saisie est HELLO<br>
<form name="form" method="post" action="myScript.php">
  <input type="text" name="value"
         value="HELLO">
</form>
</html>
```

(le serveur ferme la connexion)

Quelques informations (presque) optionnelles

- **Le champ** `Host`
  - Pas indispensable mais recommandé
  - Nécessaire lors de l'utilisation d'un *proxy* (traité plus loin)
- **Le champ** `Content-Type`
  - Requête `GET` : pas de données après l'entête
    - `Content-Type` est inutile (pas de contenu)
  - Requête `POST` : données attendues après l'entête
    - `Content-Type` **devrait** être présent
    - Pas strictement indispendable mais quelquefois nécessaire (par exemple pour le formulaire *php* précédent)
  - Réponse du *serveur* : données attendues après l'entête
    - `Content-Type` **devrait** être présent
    - Exploitation des données par le client (texte, image, vidéo ...)

Quelques informations (presque) optionnelles

- **Le champ** `Content-Length`
  - Si absent, données lues jusqu'à `EOF` (fermeture indispensable)
  - Requête `GET` : pas de données après l'entête
    - La connexion **doit** rester ouverte pour recevoir la réponse
    - `Content-Length` est inutile (forcément nul)
  - Requête `POST` : données attendues après l'entête
    - La connexion **doit** rester ouverte pour recevoir la réponse
    - `Content-Length` **doit** être présent
  - Réponse du *serveur* : données attendues après l'entête
    - La connexion peut être fermée après
    - Données statiques → `Content-Length` généralement présent
    - Données dynamiques → `Content-Length` généralement absent

Requête à travers un *proxy*

- **Influence sur la connexion**
  - Le *client* ne se connecte plus directement au *serveur* (sur le port 80)
  - Il se connecte au *proxy*, généralement sur le port 3128
- **Influence sur l'entête**
  - La ressource de la requête (après `GET`/`POST`) est l'*URL* complète
    - Sans *proxy* : `GET /rfc.html HTTP/1.1`
    - Avec *proxy* : `GET http://www.ietf.org/rfc.html HTTP/1.1`
  - La requête **doit** préciser l'information `Host`
    - Le port est optionnel (80 par défaut), `Host: www.ietf.org:80`
  - Les requêtes/réponses qui émanent du *proxy* peuvent contenir :

```
Via: 1.0 tuba:3128 (squid/2.5.STABLE3)
X-Forwarded-For: 192.168.25.1
Cache-Control: max-age=259200
X-Cache: MISS from tuba
...
```

Interagir *"à la main"* avec les *clients/serveurs*

- **Utilisation du programme `nc` (*netcat*)**
  - `nc` ***`host port`*** (ou `telnet` ***`host port`***)
    - Connexion au port indiqué sur la machine ***`host`***
    - Saisie depuis la console → envoi au *serveur*
    - Réception depuis le *serveur* → écriture dans la console
  - `nc -l -p` ***`port`***
    - Écoute sur le port indiqué de la machine locale
    - Attente de la connexion d'un client (une seule)
    - Réception depuis le *client* → écriture dans la console
    - Saisie depuis la console → envoi au *client*

Interagir *"à la main"* avec les *clients/serveurs*

- **Utilisation du programme `nc` (*netcat*)**
  - *Client/serveur* générique
    - Protocoles ***http*** (80), ***pop*** (110), ***smtp*** (25) ...
    - Adapté aux protocoles en mode texte ! (***ssh*** ?!?!)
    - Aucune signification *a priori* pour les lignes échangées
    - Les données échangées passent *"en clair"* !!!
  - Permet d'expérimenter facilement tout ce qui vient d'être vu (et bien d'autres choses ...)

Mise en œuvre minimale de *HTTP*

- **Quelques lignes de code en *C***
  - Pas besoin d'implémenter tout *HTTP*
    - Juste les fonctionnalités courantes
  - Beaucoup plus léger que la technologie ***web*** habituelle !
  - Moyen d'interaction simple pour une application embarquée
  - Les éléments du cours sur l'*API socket* sont suffisants
- **Utiliser ou non un *proxy***
  - Le *client* décide selon un réglage qui lui est propre (*IHM*, fichier de configuration, variable d'environnement . . .)
  - Solution courante : variable d'environnement
    - `HTTP_PROXY` ou `http_proxy`
    - Contenu de la forme `http://proxy.enib.fr:3128`

Mise en œuvre minimale de *HTTP*

▷ Bilan / Limitations

- ◇ Simple et peu coûteux à mettre en œuvre dans n'importe quel projet
- ◇ Passe les filtres habituels (*firewalls* & *proxys*)
  - ○ Pas de configuration sur les sites *clients* (juste le *serveur*)
- ◇ Communication toujours à l'initiative des *clients* !
  → Interrogation périodique du *serveur* ...
- ◇ Une connexion *TCP* à chaque échange !
  - ○ Option pour plusieurs requêtes/réponses sur la même connexion
    `Connection: Keep-Alive`
  - ○ Méthode `CONNECT` (voir le cours *SSL*)
  - ○ Attention ! le *proxy* peut décider de couper !
- ◇ Filtrage sur le type des données par certains *proxys* (rare)
  → Encapsuler les données dans des données autorisées (image ...)