Plan du cours

Service réseau étudié:

«Service de transmission et de consultation de documents distribués et multimédia»

- Le Protocole étudié:
  - HTTP
- Les services mis en œuvre:
  - Pour la mise à disposition: serveur WEB Apache
  - Pour la connexion: service de relayage (proxy cache Squid, proxy Socks…)
  - Pour le filtrage site urls…(Acls Apache , SquidGuard, Dansguardian)
  - Outils Annexes (outils de Stats, Interface de configuration)

Le protocole HTTP

HTTP: HyperText Transfer Protocol est un protocole de niveau application suffisamment léger et rapide, pour la transmission de documents distribués et multimédia.

Client et Serveur HTTP

les fonctionnalités du protocole sont utilisées dans une implémentations de type« client/serveur ».

Client: utilitaire graphique ou texte.

Serveur: service http, httpd, serveur web.

Client / Serveur s'échangent des messages à travers le protocole HTTP.

Caractéristiques du protocole

Il existe plusieurs versions du protocole:

A l’origine un jeu de commandes très basiques qui permet d'échanger du texte (pas de définitions explicites des types de données transportées). Aujourd’hui un jeu de commandes plus étoffé et des mécanismes qui permettent d’identifier le type des données transportées (encapsulées) et bien plus encore (MIME, cookie…)

Comprendre HTTP ! Pourquoi faire ? Juste un exemple simple…

- configurer un serveur
- tester un serveur
- comprendre une erreur
- Sécuriser
- utiliser toute les possibilités d'un langage orienté Web
…

Un exemple pour le PHP :

```
<?php

if ( strpos( $_SERVER['HTTP_USER_AGENT'], 'Firefox' ) !== FALSE ) { echo " Firefox"; }
elseif ( strpos( $_SERVER['HTTP_USER_AGENT'], 'Opera' ) !== FALSE ) { echo " Opera"; }
elseif ( strpos( $_SERVER['HTTP_USER_AGENT'], 'Safari' ) !== FALSE ) { echo "Safari"; }
elseif ( strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE' ) !== FALSE ) { echo "Internet Explorer"; }
else { echo "navigateur non reconnu"; }

?>
```

Expliquez ce programme mais surtout le contenu de la variable HTTP_USER_AGENT (où, quoi, comment…)

Même chose ip, cookie, session…

Caractéristiques du protocole

Quelque soit l’information transmise (texte, image, vidéo…), elle est transportée dans le bloc http. C'est l 'analyse du bloc http qui va permettre d'identifié le type de l'information (étiquette au format MIME par exemple). Par contre, le protocole HTTP n’identifie pas de cible pour la traiter. C’est une couche supplémentaire intégrée au client qui va le permettre (plug-ins, application externe…).

Les versions du protocole

HTTP a été utilisé à l'initiative du World-Wide Web depuis 1990. Les spécifications ont décrit le protocole sous les références HTTP/0.9. HTTP/1.0. Puis HTTP/1.1.

0.9 Une seule requête « GET »

1.0 Apporte des requêtes supplémentaires et un entête à la requête. Elle pose les bases des bonnes pratiques en termes d'en-têtes et de formatage des réponses, elle normalise les codes d'erreurs.

1.1 Requêtes suplémentaires et introduit surtout une "personnalisation" des requêtes avec la négociation de contenu (gestion intelligente des docs avec des données d'entête suplémentaires (Accept, Accept-Language, Accept-Charset) ainsi que le pipelining au travers notamment du Keepalive (Connection). Elle offre en outre la possibilité de découper la réponse en plusieurs morceaux (Transfer-Encoding).

2.0 HTTP/2 (pas web 2.0) conserve la majorité de la syntaxe de HTTP 1.1 Cette version est issue du protocole SPDY. SPDY était un projet de protocole de remplacement de HTTP précédent lancé par Google. HTTP/2 conserve la majorité de la syntaxe de HTTP 1.1, comme les méthodes, les codes, les URI ou les headers. HTTP 2.0 économise les connexions HTTP et la charge des en-têtes. HTTP2 sur l'amélioration des transferts des contenus (compression, multiplexage de requêtes…).

Fonctionnement général du Protocole

Le protocole est entièrement défini dans les RFCs:

RFC1945: HTTP/1.0

RFC2616: HTTP/1.1

RFC 7540 : HTTP/2.0

Fonctionnement général du Protocole

Le protocole HTTP est basé sur un modèle du type requête/réponse .

Un client établit une connexion vers un serveur et lui envoie une requête sous la forme d'une méthode (GET, POST...), d'une URI (*), du numéro de version et éventuellement d'un entête et d'un corps (en fonction de la version).

Format url : protocole://ident@adr_serveur:port/chemin?donneés

***Un URI, de l'anglais Uniform Resource Identifier, soit littéralement identifiant uniforme de ressource, est un format mis en place pour le World Wide Web qui normalise la syntaxe de courtes chaînes de caractères désignant un nom ou une adresse d'une ressource (codage url RFC 1738)**

Fonctionnement général du Protocole

Le protocole HTTP est basé sur un modèle requête/réponse:

Le serveur répond par une ligne d'état, incluant la version de protocole et un message de succès ou d'erreur, suivi éventuellement d'un entête et d’un corps.

Fonctionnement général du Protocole

Le protocole HTTP est basé sur un modèle requête/réponse:

Une situation plus complexe peut apparaître lorsque un ou plusieurs intermédiaires sont présents dans la chaîne de communication (proxy, frontal-routeur, proxy transparent, firewall…)

–> HTTP/2.0 répond en partie à cette problématique

Fonctionnement général du Protocole

Sur Internet, les communications HTTP s'appuient sur le protocole de connexion TCP/IP.

Le port utilisé par défaut est le port TCP 80, d'autres ports peuvent être utilisés.

Fonctionnement général du Protocole

La pratique courante spécifie qu'une connexion doit être initiée par un client (niveau TCP/IP) avant transmission de la requête http, et refermée par le serveur après délivrance de la réponse

Le protocole est sans état (stateless protocol). La notion de session n’est donc pas directement supportée.

-> La notion de session est Implémentée via des données cachées dans le formulaire, des données ré-écrite dans l’url (taille limitée), via des données positionnées dans les entêtes (cookie, session...) mais toujours mise en œuvre par des applications particulières (scripts, PHP, JAVA) écrites au dessus de HTTP.

Fonctionnement général du Protocole

Ce type de connexion permet d'éviter la gestion des connexions TCP pénalisantes en terme de ressources machine.

Par contre cela nécessite une nouvelle connexion a chaque nouveau transfert (sauf utilisation particulière du keepalive à partir de 1.1 et autres mécanismes en 2.0)

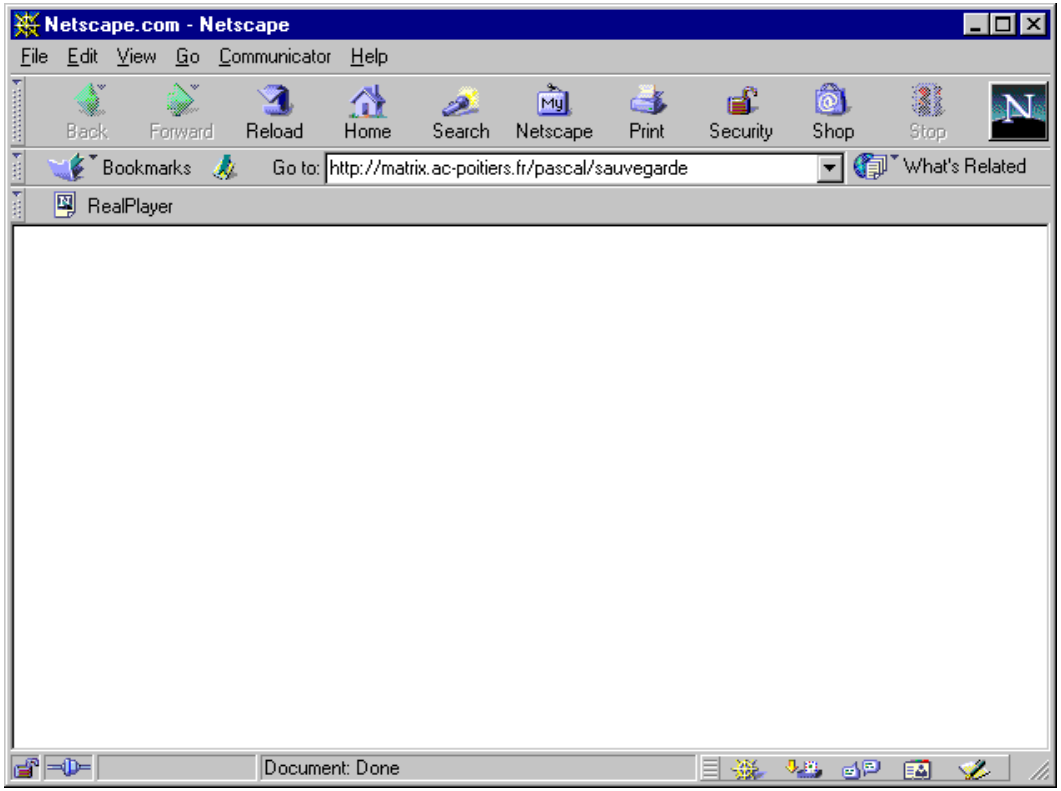
Fonctionnement général du Protocol »

Il ne faut pas confondre le protocole HTTP (conteneur HTTP 1.0 ou 1.1 : « commandes ») avec les informations transportées dans le bloc HTTP (contenu). Le protocole HTTP n'a pas évolué depuis longtemps alors que les données transportées dans le bloc HTTP s'enrichissent au fil du temps (html5, xml ...).

On peut bien entendu transporter toutes sortes de données dans le bloc http (voix, images...), il faut bien entendu que le client (navigateur) soit capable ensuite d'identifier les blocs de données et les traiter (version du moteur d'analyse du navigateur).

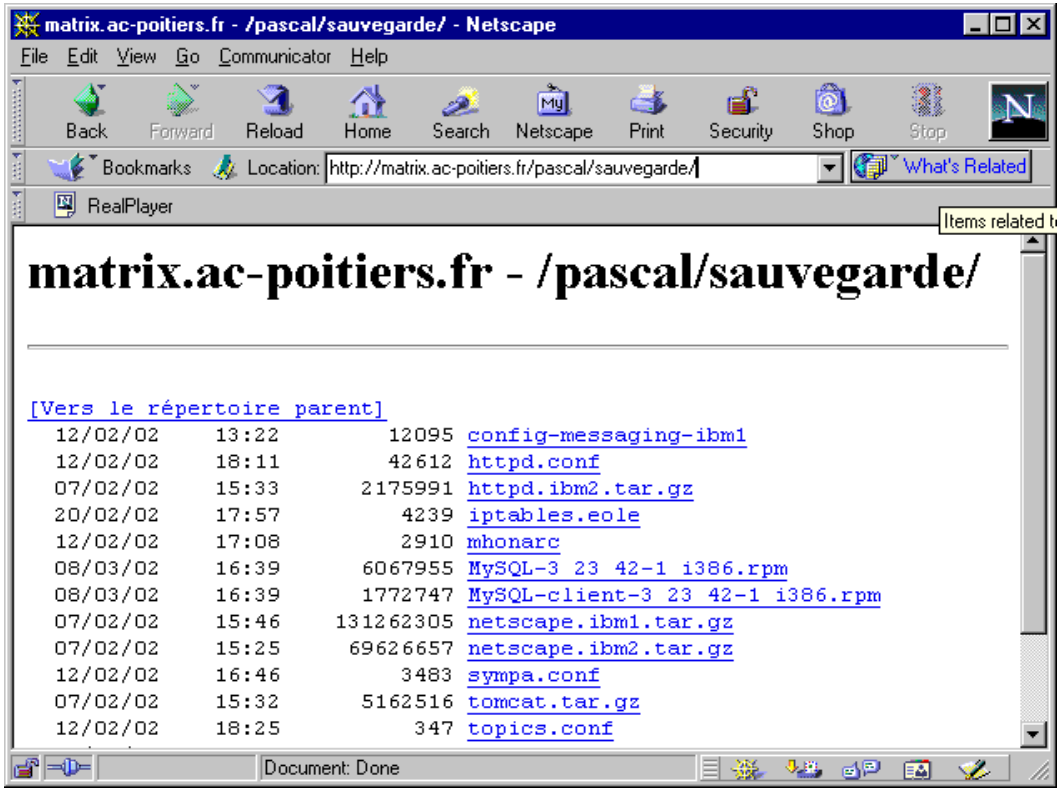
Une trame IP/TCP/HTTP

Je demande le chargement d'une URL via mon navigateur :



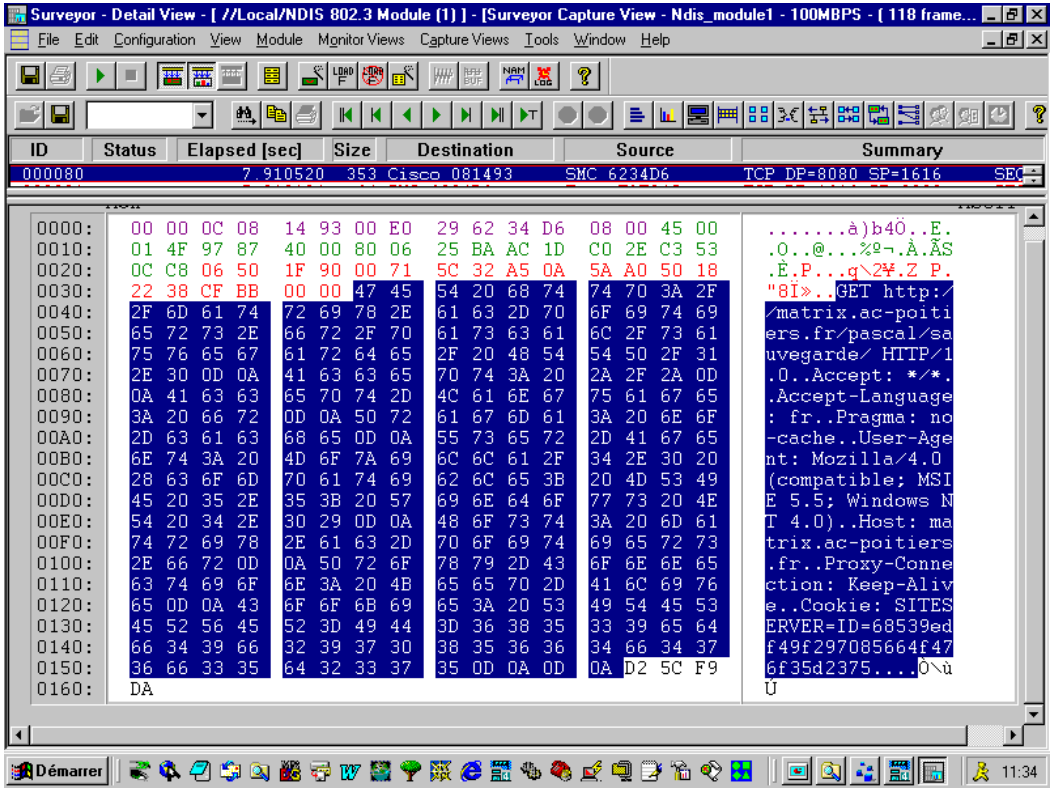
Une trame IP/TCP/HTTP

Je récupère la ressource associée en retour :



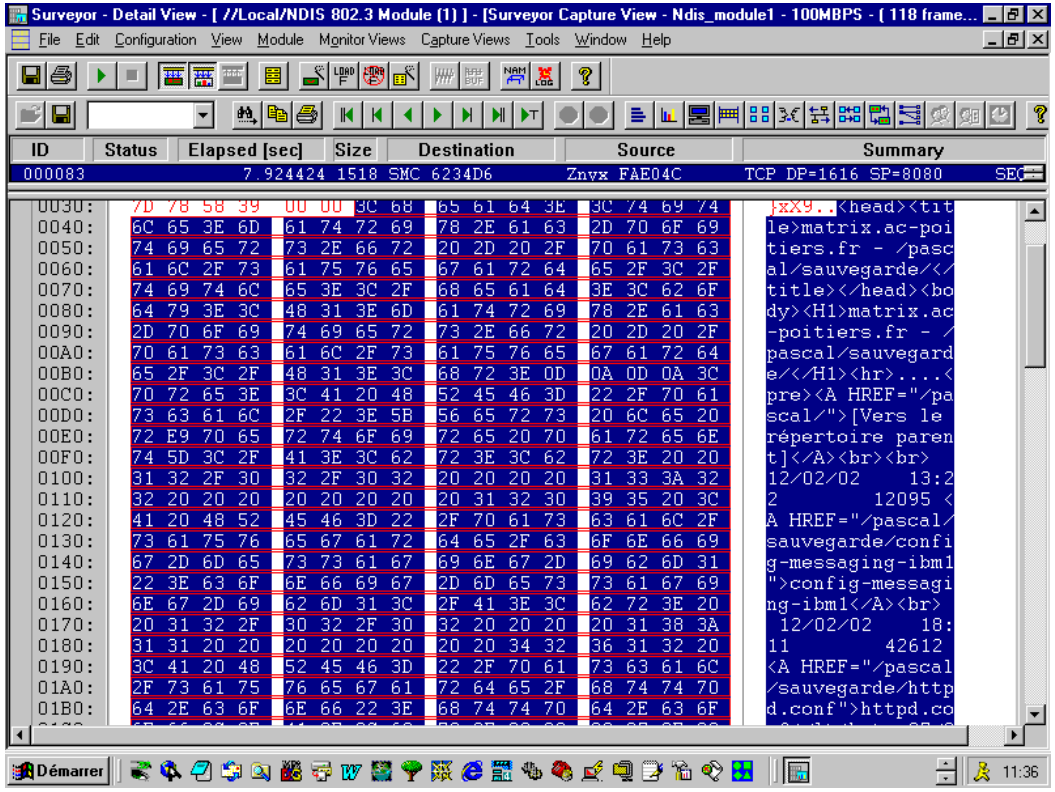
Une trame IP/TCP/HTTP

Ce qui donne au niveau des trames échangées (demande):



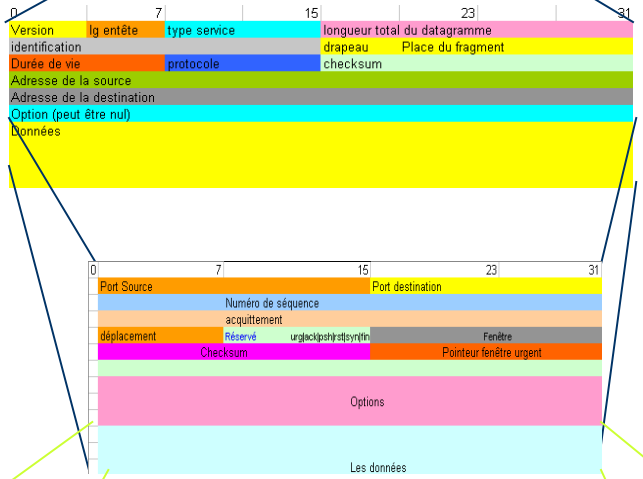
Une trame IP/TCP/HTTP

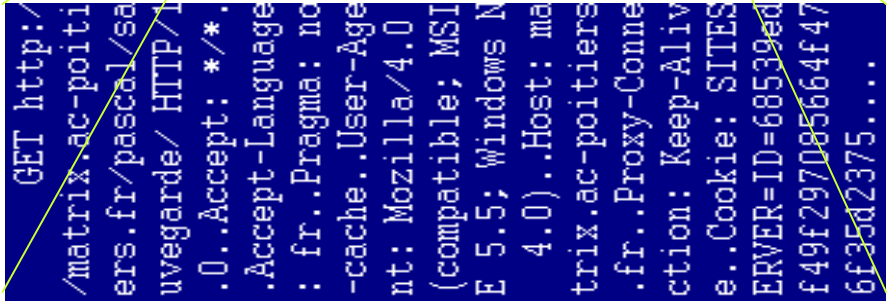
Le paquet en retour :



Trame Ethernet/IP/TCP/HTTP

Préambule	Adr destination	Adr source	Ethertype	data	Séq vérif trame

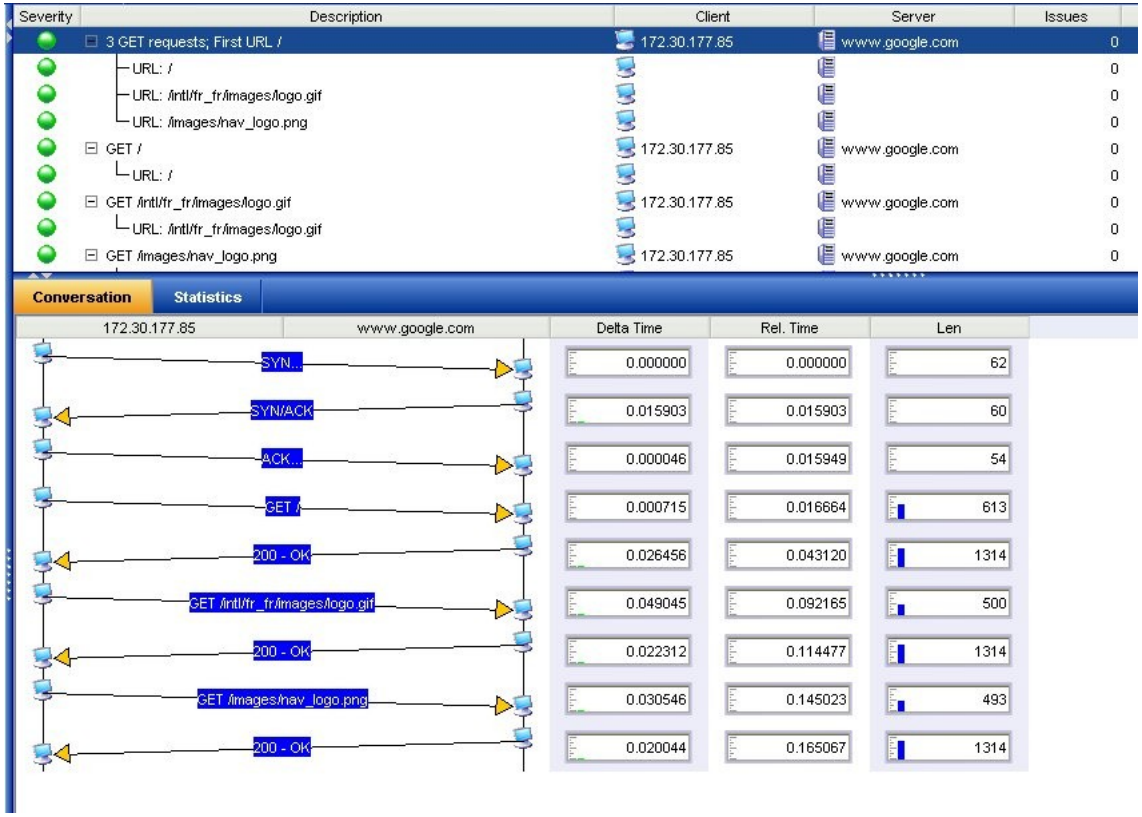




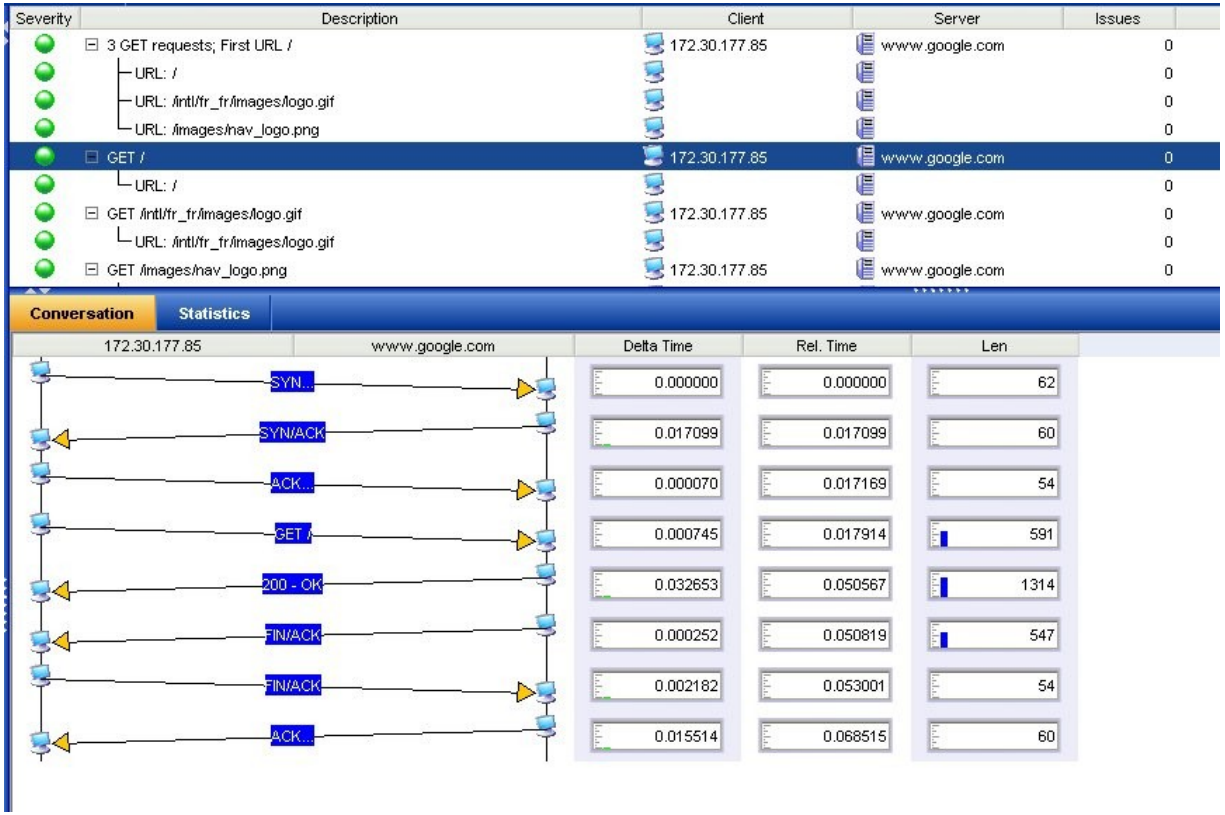
Requêtes HTTP

Bien entendu, l'analyse d'une page HTML par le navigateur va engendrer des requêtes pour l'ensemble des ressources de la page (des images par exemple). A partir de Http 1.1 le keepalive permet de générer plusieurs échanges de ressources dans une même connexion TCP/IP.

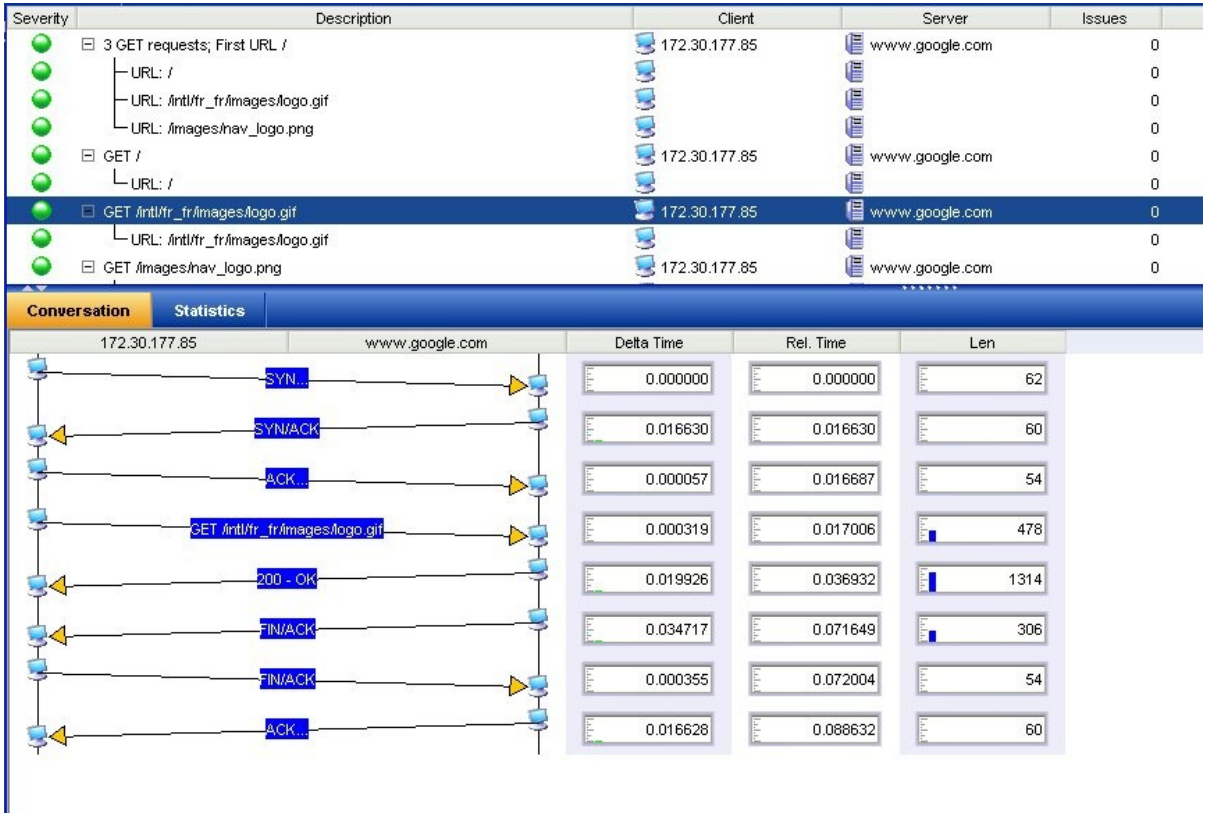
Avec KeepAlive (plusieurs requêtes/Retours HTTP dans la même connexion TCP/IP)



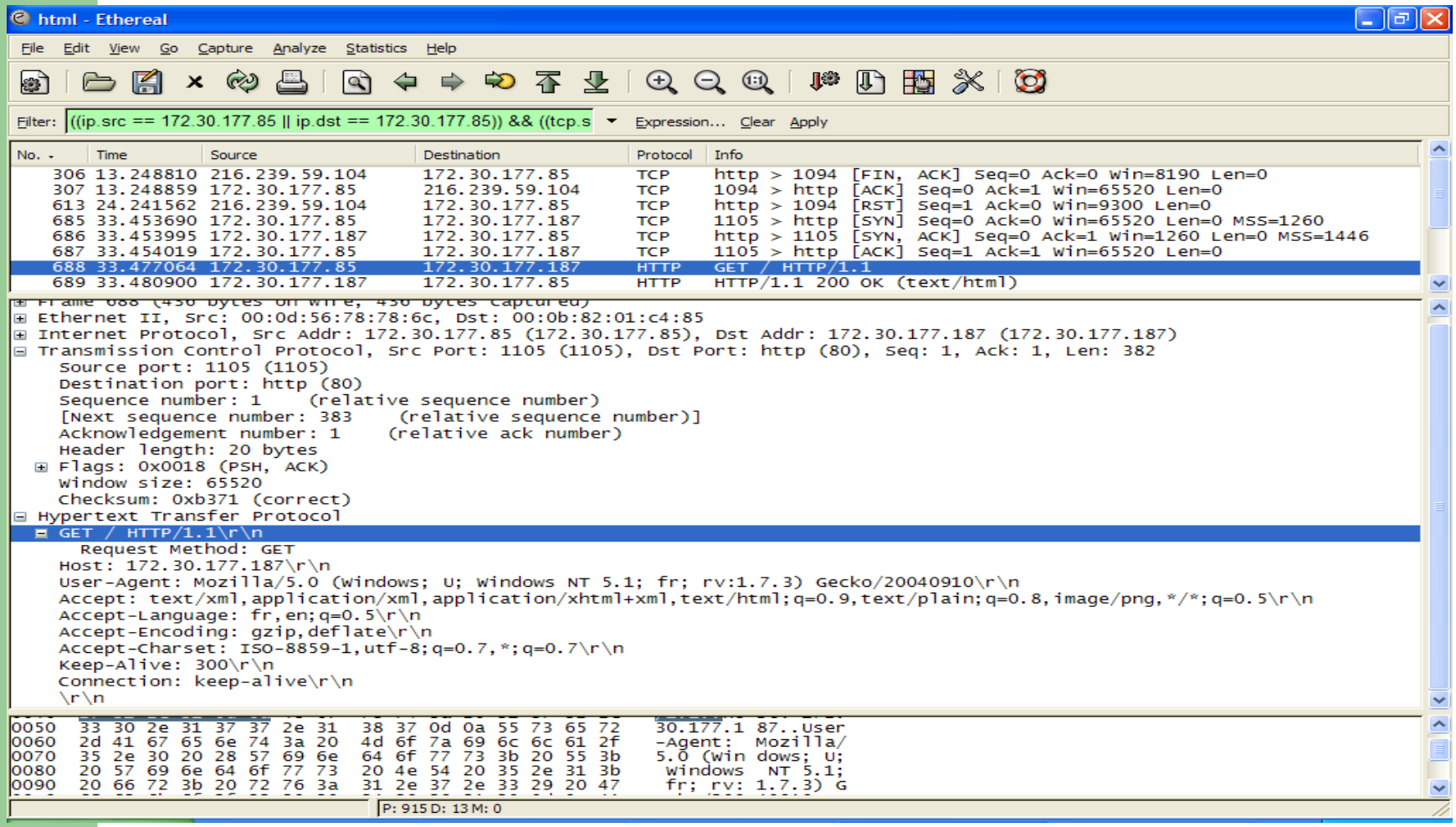
Sans KeepAlive Req 1



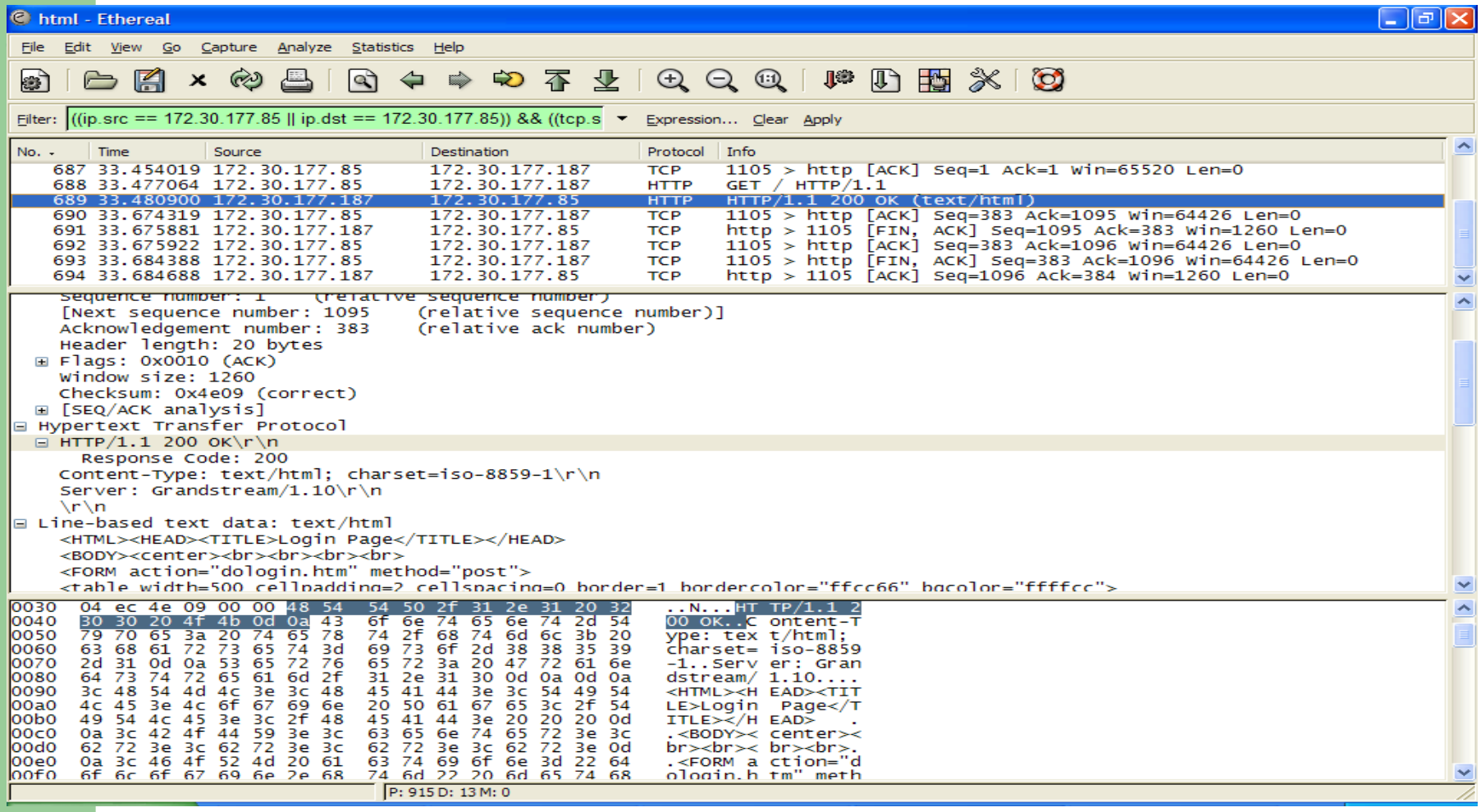
Sans KeepAlive Req 2



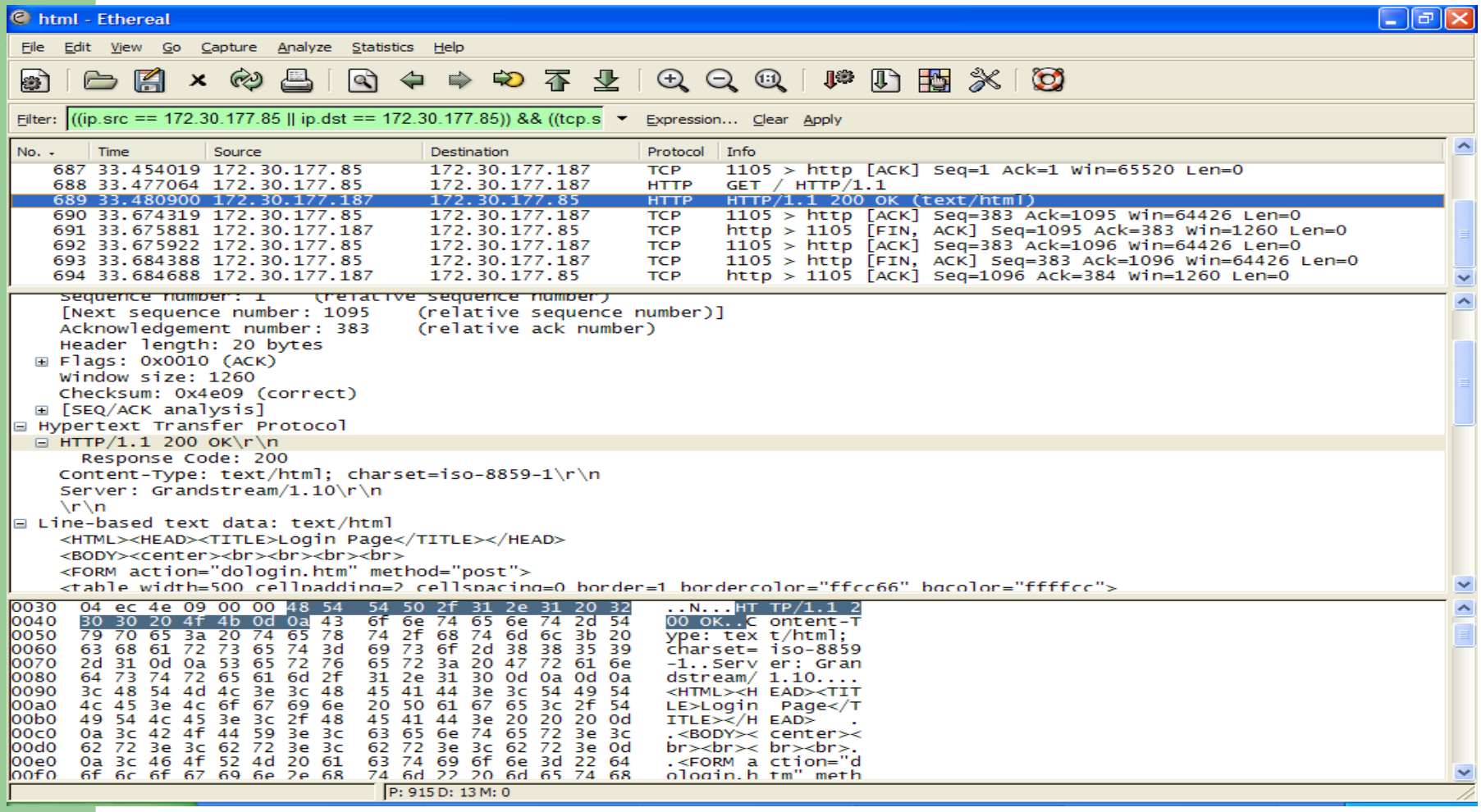
Un autre exemple...



Un autre exemple (suite)...



Un autre exemple (suite)...



Format des requêtes (client)

HTTP 0.9

une commande unique: GET http://URLI

HTTP 1.0 et HTTP 1.1

3 Parties distinctes:

La ligne de requête,

L'entête de la requête,

Le corps de la requête.

* La marque de la fin du bloc HTTP est une ligne vide. La réception de la ligne vide engendre la réponse du serveur.

Le contenu du bloc http

Les parties principales:

Pour le client et le serveur:

Les requêtes (commandes)
L’entête (*)
Le corps (*)

* En fonction de l’échange et de la version du protocole

La ligne de requête (client)

Elle respecte le format suivant:

Méthode--Protocole et URL--Version HTTP

Ex: GET http://www.monsite.com/page.html HTTP/1.0

Méthode: GET
Protocole: HTTP
URL: www.monsite.com/page.html
Version: HTTP/1.0

C’est généralement le navigateur qui va générer la commande ou des commandes du type wget Linux.

Les différentes requêtes (client)

Requête	A partir de la Version HTTP	Description
GET	HTTP/0.9	obtient un document avec ou pas de données (passage dans l'url)
HEAD	HTTP/1.0	Obtenir uniquement les infos sur la ressource (entête)
POST	HTTP /1.0	envoie du contenu au serveur dans le corps du bloc http
PUT	HTTP /1.1	demande au serveur d'enregistrer la ressource envoyée
DELETE	HTTP /1.1	permet d'effacer un fichier sur le serveur
TRACE	HTTP /1.1	Demande de renvoie de la ressource envoyée (débogage)
CONNECT	HTTP/1.1	mot réservé permettant de créer des tunnels (connexion TCP relayée) http, https, smtp…
OPTIONS	HTTP /1.1	Obtenir des options d'une ressources ou du serveur

L’entête (client)

Il permet de préciser la requête en ajoutant des directives (a partir de 1.0).

Quelques champs d’entête:

CHAMP	DESCRIPTION
Accept	Type de contenu (type MIME) accepté par le navigateur
Accept-Charset	Jeu de caractères attendu par le navigateur
Accept-Encoding	Codage de données accepté par le navigateur
Accept-	Langage attendu par le navigateur
Authorization	Identification du client
Content-Encoding	Type de codage du corps de la requête
Content-Language	Type de langage du corps de la requête
Content-Length	Type de contenu du corps de la requête
Date	Date de début de transfert des données
Forwarded	Utilisé par les machines intermédiaires entre le client et le serveur
Location	Redirection URL (serveur)
Link	Relation entre deux URL
Orig-URL	URL d’origine de la requête
Referer	URL du lien à partir duquel la requête a été effectuée
User-Agent	Informations sur le client navigateur, système d’exploitation

Le corps de la requête (client)

Le corps de la requête est généralement présent quand le client envoie des données au serveur via la méthode POST et PUT. La méthode GET peut également permettre au client (navigateur) d'envoyer des données à travers une url

(ex http://url..?var1=contenu1&var2=contenu2)

Format des réponses (serveur)

HTTP 0.9

Contient uniquement le contenu du document

HTTP 1.0 et HTTP 1.1 et 2.0

3 Parties distinctes:

La ligne de statut,

L’entête de la réponse,

Le corps de la réponse.

La ligne de statut (serveur)

Sert à préciser la manière dont s’est passé le traitement de la requête

3 parties:

Version HTTP, (HTTP/1.1)

Statut numérique de la réponse (200)

Statut texte de la réponse (OK)

La ligne de statut (serveur)

5 classes de statut:

1XX Information: non utilisé en http 1.0

2XX requête traitée avec succès (200 OK)

3XX redirection de ressource

4XX requête incorrecte (côté client)

5XX requête non satisfaite (côté serveur)

L’entête (serveur)

Généralement des directives de types génériques, mais également propre à la réponse et spécifique au contenu renvoyée.

L’entête (serveur)

Générique: Date, Pragma (comportement: no-cache…)

Spécifiant la réponse: Location (URL absolue), Server, WWW-authenticate (identification)

Spécifiant le contenu: Type mime, taille, codage, date de dernière modif, Allow (méthode acceptée sur la ressource)

Le corps (serveur)

Le résultat de la requête (peut être vide dans le cas de l’utilisation de certaines méthodes ou lors d’une erreur 404 par exemple)

Bloc Requête http

GET http://webmail.ac-amiens.fr HTTP/1.0 **(requête)**

Accept-language: fr

... ### toutes les variables http **(entête)**

cr

Corps…

Requête http via un simple telnet

telnet Ip-ou-NPQ 80 **(connexion)**

GET http://webmail.ac-amiens.fr HTTP/1.0 **(requête)**

Accept-language: fr
... ### toutes les variables http **(entête)**

cr

cr **(pas de corps)**

Requête http via un simple telnet

Ce qui donne en retour:

HTTP/1.0 200 OK (ligne de statut)

Date: Tue, 25 Feb 2003 11:11:53 GMT (entête)
Content-Type: text/html
Last-Modified: Thu, 23 Jan 2003 14:22:30 GMT
Content-Length: 5438
X-Cache: MISS from perroquet
Proxy-Connection: keep-alive

<html> (corps)
 <head>
<title>Authentification : Webmail</title>
<script>
…

Envoie Données Navigateur vers Serveur

A travers un formulaire en local, on va envoyer les données renseignées via une méthode GET ou POST (Généralement via un formulaire HTML reçu auparavant).

METHODE dans les formulaires HTML indiquent la méthode HTTP par contre les balises du formulaire permettent de présenter les éléments interactifs localement:

INPUT, TEXTEREA, SELECT,

ACTION permet d'indiquer le script (CGI, PHP…) qui va récupérer les données.

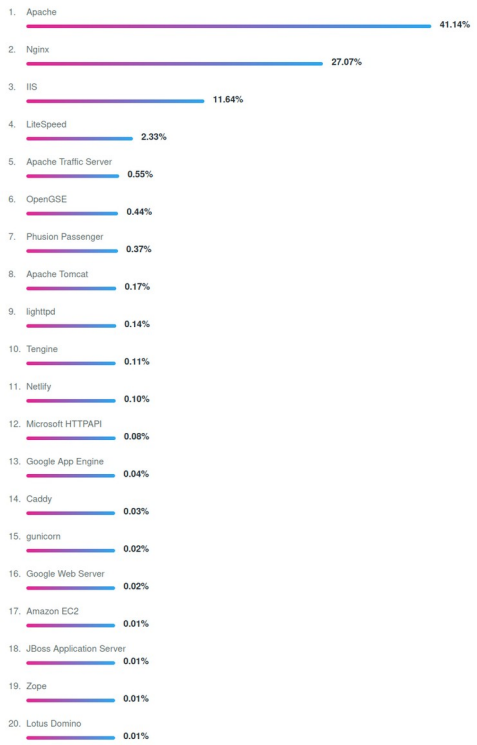
Passage des données du formulaire:

GET passage dans l'URL soit: URL?VAR1=VAL1&VAR2=VAL2…

POST dans le corps du bloc HTTP

Mise en œuvre d’un serveur ou service http

Le marché des serveurs HTTP 2021:



Mise en œuvre d’un serveur ou service http

Dans le cadre du cours nous allons mettre en œuvre Apache en tant que:

- Serveur Web standard
- FRONTAL des services HTTP (Proxy Inverse):
  Il va servir de Proxy pour tous les services de l’organisation.

Mise en œuvre d’un serveur ou service http

Apache désigne le serveur HTTP open source.

L’ASF « Apache Software Foundation » gère de nombreux projets (java, XML…) et en particulier celui du serveur http.

C’est une entité virtuelle présente uniquement sur le net.

A PAtCHy web sErver (clin d’œil à l’échange des patches).

Le serveur Apache

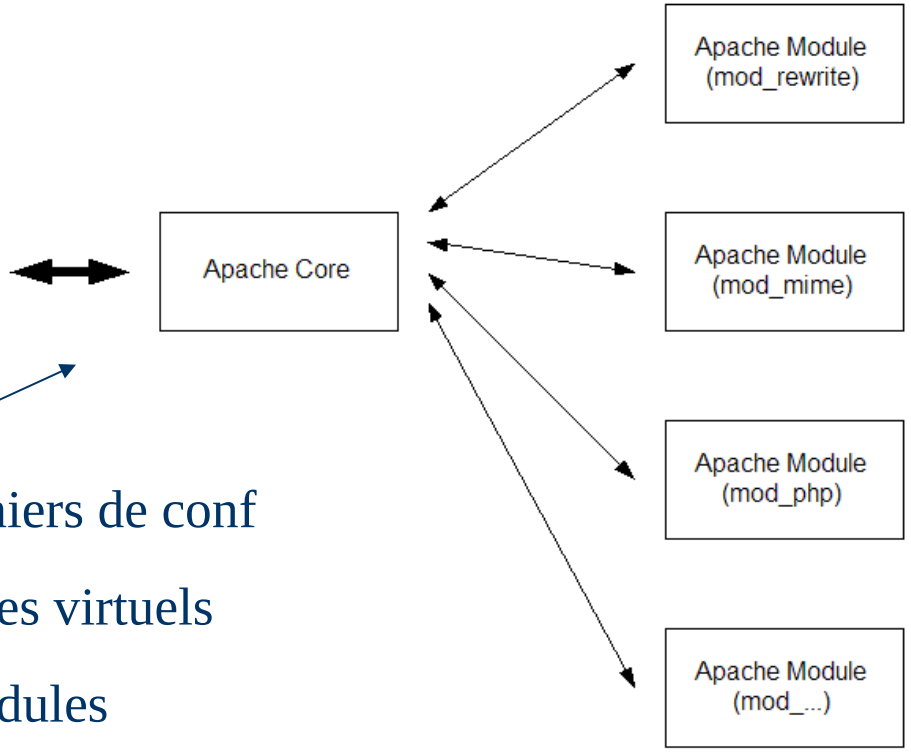
Conforme au protocole HTTP

Intègre le concept de modules http://modules.apache.org

Existe différentes sur plate-formes Linux Windows…

Se décline en plusieurs versions 1.3.X 2.0.X 2.2.X...(multi-thread, Multi-Processing Modules: MPM worker)

Le serveur Apache



- Lecture des fichiers de conf
- Gestion des hôtes virtuels
- Gestion des modules
- Communication inter-modules

Le serveur Apache

Les modules:

Module statique: intégré dans le cœur du serveur lors de la compilation

Module dynamique: lié lors de l'exécution via le support DSO du serveur (librairie dynamique type dll Microsoft)

Le serveur Apache

Installation sous linux:

Paquetage (RPM, APT …)

Source www.apache.org

Le serveur Apache

Installation via les sources:

tar zxvf apache…tar.gz

./configure –prefix=/usr/local/apache –enable-module …….

make

make install

Installation via un paquet apt:

Apt-get install apache

Le serveur Apache

Configuration:

Le fichier central est httpd.conf

Il se compose de directives générales et de directives spécifiques à une section ou à un module. La portée des directives est générale ou locale en fonction de sa position (en dehors ou dans une section).

Sections:

VirtualHost, Directory(Match), Location(Match), Files(Match), IfDefine, IfModule.

* Match permet de préciser la cible via des expressions régulières

Exemples:

Le serveur Apache: Directives et Sections

Démarrage	Paramètres généraux	Arborescence	Modules	Logique
ServerType	ServerAdmin	ServerRoot	LoadModule	IfDefine
StartServers	ServerName	DocumentRoot		
User et Group		Alias(Match)		
Listen		DirectoryIndex Directory(Match) Options		
KeepAlive				
MaxKeepAliveReguests				
KeepAlivelimeout		Files(Match)		

Le serveur Apache: Directives et Sections

ServerType: standalone ou inetd
ServerAdmin: adresse électronique de l'administration du serveur
ServerName: Nom du serveur correspondant à une IP (fichier hosts ou DNS)
ServerRoot: Répertoire des fichiers du serveur (généralement /usr/local/apache)
PidFile: fichier contenant le PID à partir du ServerRoot
StartServers: nombre de processus créés au démarrage
User: UID utilisé utilisé par le serveur httpd (généralement nobody ou apache)
Group: GID
Listen: Ip et Port d'écoute
DocumentRoot: Répertoire racine des URI
Alias: Permet d'associer un répertoire autre que DocumentRoot pour une URI
DirectoryIndex: page par defaut dans un répertoire (index.htm default.asp…)
LoadModule: chargement de modules (.so)
IfDefine: Application de directives si définit au démarrage (passage en argument)

Le serveur Apache: Directives et Sections

Exemple:
Alias permet un mappage entre une URI et un répertoire sur disque.
Directory permet d'appliquer toutes les directives dans la section sur le répertoire précisé.
FilesMatch permet d'appliquer les directives sur un fichier(s) particulier(s) exprimé via une expression régulière

Alias « /rep » « /chemin/ici/ou/la »
<Directory chemin/ici/ou/la/>
Options +FollowSymlinks
Options –Indexes
<FilesMatch « ^L »
Deny from all
</filesMatch »
</Directory>

Indexes: permettre le listage du répertoire
FollowSymlinks: suivre les liens symbolique dans ce répertoire
Deny: interdire l'accès

Le serveur Apache: Directives et Sections

Gestion du KeepAlive:
KeepAlive On: Activer le keepAlive
KeepAliveTimeOut On: temps max de l'ouverture d'une même connexion
MaxKeepAliveRequest: Nombre max de requêtes dans une même connexion
Redirection:
Redirect(Match) /labas
Redirect(Match) /Version /New_Version
Alias sur un répertoire de script (à exécuter avant l'envoie):
ScriptAlias /cgi-bin/ /usr/local/apache2/script

Le serveur Apache: Virtual hosts

Permet de gérer plusieurs sites (dits virtuels) via différents Noms ou IP.

Ip-based Virtual Hosts: le serveur écoute plusieurs adresses IP
Les résolutions de nom correspondent à des Ips différentes.

Name-based Virtual hosts: Une seule IP correspondant à plusieurs noms (variable d'entête host).

Le serveur Apache: Virtual hosts (IP-Based)

```
<VirtualHost 194.199.46.10>
# ou <VirtualHost www.site1.dom1>
ServerAdmin 1
DocumentRoot /ici/la/rep1
ServerName www.site1.dom1
ErrorLog /ici/la/log/Error-site1
TransfertLog /ici/la/log/Access-site1
</VirtualHost>

# Deuxieme site
<VirtualHost 194.199.46.11>
# ou <VirtualHost www.site2.dom2>
ServerAdmin 1
DocumentRoot /ici/la/rep2
ServerName www.site2.dom2
ErrorLog /ici/la/log/Error-site2
TransfertLog /ici/la/log/Access-site2
</VirtualHost>
```

Au niveau DNS 1 IP différente pour chacun des sites.

Au niveau VirtualHost on peut à la place de l'IP replacer le nom du site

Au niveau du DNS il faut 2 entrées

Le serveur Apache: Virtual hosts (Named-Based)

Une IP plusieurs sites:

```
NameVirtualHost 194.199.46.10
<VirtualHost 194.199.46.10>
ServerAdmin 1
DocumentRoot /ici/la/rep1
ServerName www.site1.dom1
ErrorLog /ici/la/log/Error-site1
TransfertLog /ici/la/log/Access-site1
</VirtualHost>

# Deuxieme site
<VirtualHost 194.199.46.10>
ServerAdmin 1
DocumentRoot /ici/la/rep2
ServerName www.site2.dom2
ErrorLog /ici/la/log/Error-site2
TransfertLog /ici/la/log/Access-site2
</VirtualHost>
```

Utilisation du CNAME au niveau du DNS:

1 IP pour plusieurs noms

Au niveau du DNS il faut 1 entrée et 1 alias (CNAME)

Le serveur Apache: Sécurité et Contrôle d'accès

Le contrôle d'accès:

```
# dans un conteneur <Directory> / aux IPs
<Directory /usr/local/apache/htdocs/prive>
Order Allow,Deny
Allow from 172.30.177.0 pv.ac-amiens.fr
Deny from 172.30.176.0
</Directory>
```

L'authentification:

```
<Directory /usr/local/apache/htdocs/prive>
AuthType Basic
AutName « Entrer mot de passe »
AuthUserFile /usr/local/apache/htdocs/prive/htaccess
Require user toto titi tutu
</Directory>
```

Génération d'un fichier mdp et d'un id:

```
htpasswd –c file htaccess
Htaccess –c htaccess toto
```

Le serveur Apache: Sécurité et Contrôle d'accès

Le contrôle d'accès IP ou mdp:

```
# dans un conteneur <Directory> / aux IPs
<Directory /usr/local/apache/htdocs/prive>
Order Allow,Deny
Allow from 172.30.177.0 pv.ac-amiens.fr
# Deny from all
AuthType Basic
AutName « Entrer mot de passe »
AuthUserFile /usr/local/apache/htdocs/prive/htaccess
Require user toto titi tutu
</Directory>
```

Le contrôle d'accès IP et mdp:

```
# Ajouter
Satisfy any
```

Il existe également des modules LDAP, RADIUS pour le Contrôle d'accès.

Le serveur Proxy: Fonctionnement

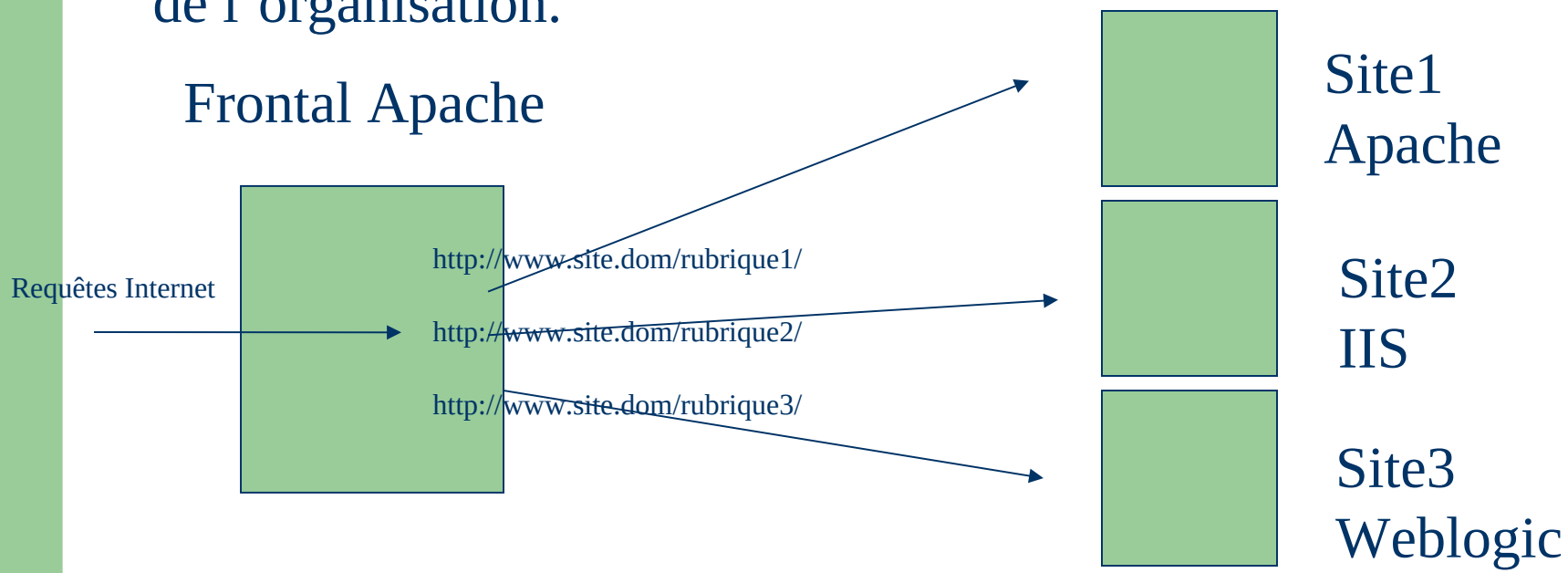
Le principe de fonctionnement basique d'un serveur proxy est assez simple : il s'agit d'un serveur "mandaté" par une application pour effectuer une requête à sa place.

Proxy-web, proxy-ftp…

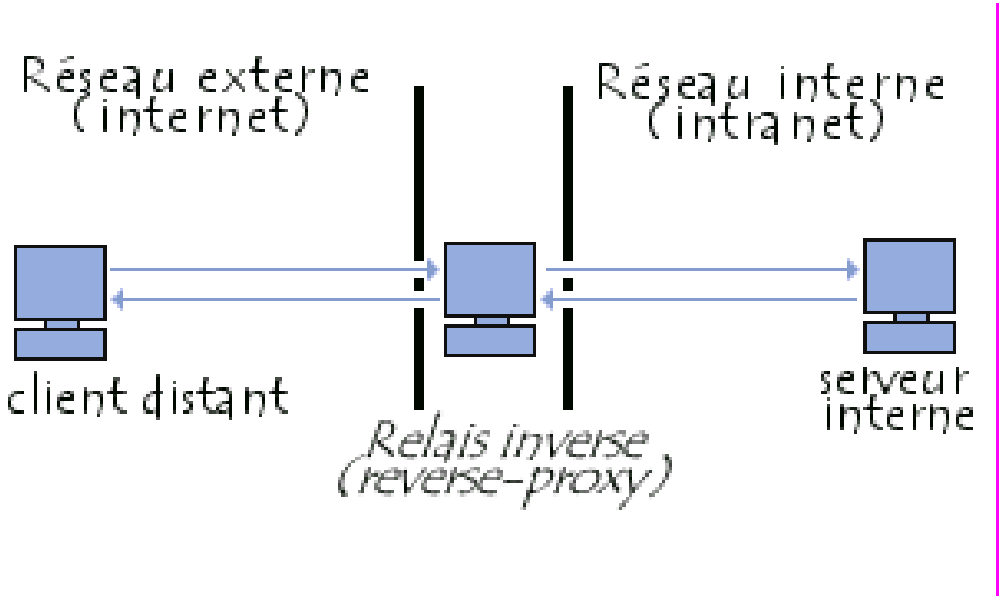
Pour le web on utilise le proxy classique (faisant généralement aussi du cache) et le reverse proxy.

Le serveur Apache en Frontal HTTP (reverse-proxy)

C’est le but de notre première étude d’apache.
Le serveur va servir de frontal HTTP pour tous les services de l’organisation.



Le serveur Apache en Frontal HTTP (reverse-proxy)



Le serveur Apache et les modules proxy

A la compilation positionner les options nécessaires:

En fonction de la version apache (1.3, 2.X)
--enable-module=proxy,
--enable-proxy,
--enable-proxy-connect,
--enable-module-http,
--enable-module-ftp

Dans la configuration:

Pour un proxy (cache) ordinaire:

ProxyRequests On

Dans le cadre de ce cours, nous réaliserons cette fonctionnalité de proxy cache avancée avec le service SQUID.

Le serveur Apache en Proxy-cache

Apache configuré en serveur proxy-cache:

<IfModule mod_proxy.c>
ProxyRequests On
NameVirtualHost 172.30.176.185:3128
<VirtualHost 172.30.176.185>
ServerName pvaniet2.in.ac-amiens.fr
cacheRoot « /var/cache » → répertoire de cache
CacheSize 1000 → 1000ko de cache
CacheGcInterval 4 → Vérifier les objets du cache tous les 4 heures
CacheMaxExpire 24 → Objets périmés au bout de 24 heures
CacheLastModifiedFactor 0.1 → Facteur pour calculer la date d'expiration / à la date dernière modification
CacheDefaultExpire → 1 délai d'expiration de l'objet du cache quand date dernière modif inconnue
</VirtualHost>
</ifModule>
ProxyRemote permet de chainer avec un autre serveur Proxy.

Le serveur Apache en Frontal HTTP (reverse-proxy)

Sur le frontal:
Dans le fichier de conf il faut les directives suivantes pour chacun des différents services à relayer:

```
ProxyPass    /monAppli
ProxyPassReverse    /monAppli
```

ProxyPassReverse permet de modifier l'entête HTTP pour une transparence complète (modification de l'entête http « location » si besoin).

* On peut placer les services à relayer sur une Zone protégée.
* On peut utiliser des ports > 1024 (ici 8065)
* On peut utiliser SSL/TLS

Le serveur Apache en Frontal HTTP (reverse-proxy)

Sur le serveur qui héberge le site:

Configuration « traditionnelle » en écoute sur le port.

On peut limiter l'accès des serveurs au frontal (ex: iptables).

→ Idéal pour la sécurité des serveurs web (apache en frontal et derrière serveurs réputés moins sécurisés (Tomcat, websphère, jboss…). Egalement pour l'authentification, le logging…

Le serveur Apache en Frontal HTTP (reverse-proxy)

Exemple de configuration:

<VirtualHost *> ServerName w4.ac-amiens.fr

ProxyPass / http://172.30.176.5/

ProxyPassReverse / http://172.30.176.5/

ProxyErrorOverride On (si besoin)

ProxyPreserveHost On (si besoin)

</VirtualHost>

Le serveur Apache en Frontal HTTP (reverse-proxy)

Depuis la version 2 le module mod_proxy inclut une possibilité de répartition de charge sur plusieurs serveurs reverses:

```
<Location /balancer-manager>
setHandler balancer-manager
Order deny, allow
Deny from All
Allow from localhost
</Location>
<Proxy balancer://balancer/ stickysession=PHPSESSIONID>
BalancerMember http://www1.dom.com
BalancerMember http://www2.dom.com
BalancerMember http://www3.dom.com
</proxy>
ProxyPass /content balancer://balancer/
```

Le serveur Proxy: Intérêts

- L'optimisation de la bande passante sur le lien Internet (proxy-cache)
- Utilisation IP privée côté clients et une IP publique côté proxy
- Le contrôle d’accès au niveau IP des postes
- Filtrage de l'accès des URI
- Authentification des utilisateurs
- Chaînage de cache (pour les proxy-cache)
- …

Le serveur Proxy-cache squid: Installation sources

Pour le relayage web (http, https) et ftp

Le site de référence:

Téléchargement des sources et installation:

```
# tar xzvf squid-xxxxx-src.tar.gz
# cd squid-xxxxx
```

Compilation:

```
# ./configure --prefix=/usr/local/squid –enable-proxy
# make
# make install
```

Le serveur Proxy squid: Configuration

Dans le fichier squid.conf

```
# port du proxy
http_port 3128

# port inter-cache
icp_port 3130

# mémoire cache
cache_mem  128 MB

# uid des process fils
cache_effective_user squid
# gid des process fils
cache_effective_group squid
```

Le serveur Proxy squid: Configuration

```
# LOGFILE PATHNAMES AND CACHE DIRECTORIES
# -----------------------------------------------------------------------

# cache_dir taille niveau arbo
cache_dir ufs /cache/squid 2000 16 256

# Pour les logs d'accès
cache_access_log /usr/local/squid/var/logs/access.log

# Pour les logs de cache
cache_log /usr/local/squid/var/logs/cache.log

# Pour les logs du store
cache_store_log /usr/local/squid/var/logs/store.log
```

Le serveur Proxy squid: Configuration

```
# OPTIONS FOR TUNING THE CACHE
# ---------------------------------------------------------------------------

#reply_body_max_size 0

#  TAG: refresh_pattern permet de configurer la durée de mise à jour du cache
#   Permet de définir la fraicheur d'une ressource cachée

#Default:
refresh_pattern         ^ftp:  1440 20%  10080
refresh_pattern         ^gopher:    1440 0%    1440
refresh_pattern  .              0       20%  4320
```

Le serveur Proxy squid: Configuration

```
#  Définitions des ACLs
acl QUERY urlpath_regex cgi-bin \?
acl all src 0.0.0.0/0.0.0.0
acl wupdate1 dstdomain .windowsupdate.microsoft.com
acl Safe_ports port 80 21 443 563 70 210 1025-65535
acl localdom dstdomain .ac-amiens.fr
acl dmz47out dst 194.199.47.0/255.255.255.128

always_direct allow localdom
always_direct allow wupdate1

# ne pas cacher
no_cache deny wupdate
no_cache deny QUERY

http_access allow manager localhost
http_access deny all
```

*Tous les éléments figurant dans la même entrée d'accès sont associés par un ET
*Tous les éléments figurant dans la même acl sont associés par un OU

Le serveur Proxy squid: Configuration

```
# ADMINISTRATIVE PARAMETERS
# ------------------------------------------------------------------------

cache_mgr reseau@ac-amiens.fr

#  TAG: logfile_rotate
logfile_rotate 0

visible_hostname proxy1.ac-amiens.fr
```

Le serveur Proxy squid: Filtrage Direct Squid

Utiliser Squid avec filtrage d'URLs:

- Directement via des ACLs
- A travers un outil annexe (SquidGuard)

Le serveur Proxy squid: Filtrage Direct Squid

```
acl horaire_restreint_semaine time MTWHF 08:30-17:30
acl horaire_restreint_samedi_matin time A 08:30-13:59
acl novell url_regex -i novell
acl mon-lan  src    172.30.176.0/255.255.240.0
acl all      src    0.0.0.0/0.0.0.0
acl site-interdit dstdomain   mauvais.fr
acl site-ok dstdomain   bon.fr

http_access allow mon-lan horaire_restreint_semaine novell
http_access allow mon-lan horaire_restreint_samedi_matin novell
http_access deny site-interdit
http_access allow mon-lan site-ok
http_access deny  all
```

Les règles http_access sont interprétées séquentiellement dans l'ordre.
Aussitôt qu'une règle match le parcours s'arrête.

Le serveur Proxy squid: Filtrage Direct Squid

Utilisation d’un fichier externe:

```
acl site-ok dstdomain "/usr/local/squid/etc/domain_ok"

http_access allow mon-lan site-ok
http_access deny  all
```

Le serveur Proxy squid: Filtrage SquidGuard

SquidGuard est un outil annexe qui va permettre le Filtrage d'URLs à partir d'une BDD (DB berkeley)
C'est squid qui va passer l' Url de la requête à squidGuard.

La directive à placer dans httpd.conf:

```
redirect_program /usr/local/squidGuard/bin/squidGuard.sh

Dans squidGuard.sh

#!/bin/sh

exec /usr/local/squidGuard/bin/squidGuard –c \
/usr/local/squidGuard/etc/squidGuard.conf
```

Le serveur Proxy squid: Filtrage SquidGuard

Installation:

→Vérifier l'installation DbBerkeley

```
tar xzvf squidGuard-1.2.0.tar.gz
cd squidGuard-1.2.0
./configure --prefix=/usr/local/squid
make
make install
```

Le serveur Proxy squid: Filtrage SquidGuard

Configuration:

Dans le fichier squidGuard.conf:

```
# Définition Log
logdir /usr/local/squidGuard/var/logs
# Répertoire pour la création des bases
dbhome /usr/local/squidGuard/var/blacklists
# Définition des Filtres
dest adult {
        domainlist adult/domains
        urllist    adult/urls
        log  refus.log
    }
```

Suite diapos suivante…

Le serveur Proxy squid: Filtrage SquidGuard

```
dest agressif {
        domainlist agressif/domains
        urllist    agressif/urls
        log  refus.log
    }
…
acl {
    default {
  pass !pub !adult !warez !violence !redirector !hacking !gambling !drogue !audio-video !agressif !aggressive  all

 redirect   http://127.0.0.1:8500/cgi-bin/filtre.cgi?clientaddr=%a&etabname=ac-amiens.fr&url=%u&clientident=%i
       }
    }
```

Le serveur Proxy squid: Filtrage SquidGuard

Si une URL match sur la base interdite une redirection est faite vers un serveur http en écoute sur l'interface loopback sur le port 8500

Une page est alors construite pour annoncer l'interdiction de l'accès.

C'est un script CGI (perl) qui construit la page d'interdiction: Script.cgi

Le serveur Proxy squid: Filtrage Dansguardian

- *Dansguardian ne fonctionne pas seulement sur une base d'Urls interdites il permet à travers de nombreux fichiers de configuration de paramétrer l'analyse en temps réels sur les contenus qui transitent (différentes méthodes sont utilisées).*

- *Comme SquidGuard, DansGuardian n'est pas un proxy et ne fait pas de mise en cache, il fonctionne de concert avec un proxy. Les requêtes internet sont dirigées vers dansguardian qui réalisera le filtrage après avoir demandé au proxy (Squid) de réaliser la requête. Il s'avère redoutablement efficace. et peut générer des faux-positifs, c'est-à-dire des pages valides interdites par erreur.*

Le serveur Proxy squid: Filtrage Dansguardian

Installation:

Projet source: http://dansguardian.org/?page=download
./configure, make, make install

Paquet (ex:Ubuntu):
Apt-get install dansguardian

Docs et Howto à l'adresse du projet.

HTTPS

C’est le protocole http encapsulé dans un flux chiffré (http sécurisé) qui utilise des méthodes de chiffrement symétriques et asymétriques.

HTTPS

Le protocole SSL (Netscape) puis TLS (*IETF Internet Engineering Task Force en 2001*)

Ces protocoles permettent de négocier les paramètres d'échanges sécurisés puis d'échanger les messages dans le mode choisit à travers un système à clé secrète.

Transport Layer Security (TLS)

Secure Socket Layer (SSL)

HTTPS

- La sécurisation des transactions par SSL est basée sur un échange de clés entre client et serveur. La transaction sécurisée par SSL se fait selon le modèle suivant :
- Le client se connecte au site sécurisé par SSL et lui demande de s'authentifier. Il envoie la liste des systèmes de chiffrement qu'il supporte
- Le serveur envoie un certificat au client (clé publique du serveur, signée par une autorité de certification), ainsi que le nom du système du chiffrement commun ayant la plus grande taille de clé.
- Le client vérifie la validité du certificat puis crée une clé secrète aléatoire, il chiffre cette clé à l'aide de la clé publique du serveur, et envoie cette clé de session.
- Le serveur déchiffre la clé de session avec sa clé privée. Le reste des échanges peut se faire à l'aide de cette clé garantissant l'intégrité et la confidentialité des données échangées.

* Avec SSL 3.0 peut éventuellement authentifier le client vis-à-vis du serveur.

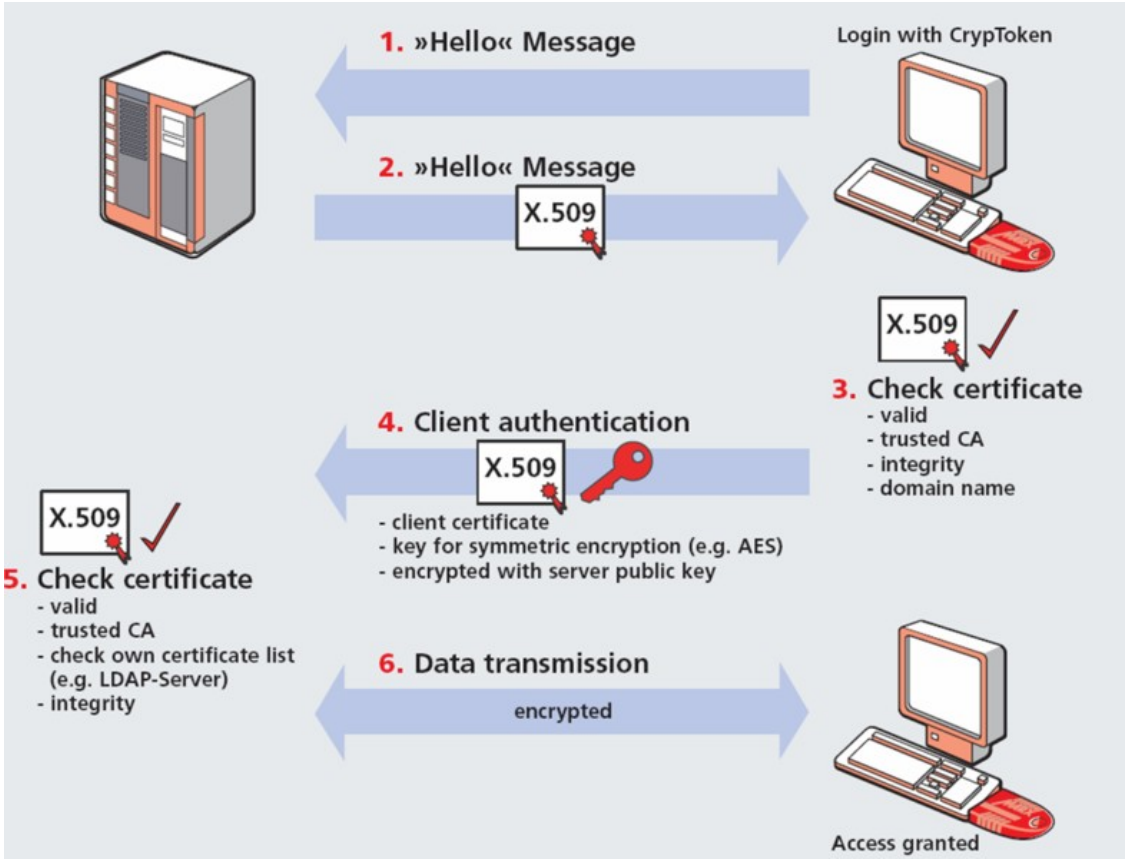
HTTPS

Que peut-on (ou doit-on) attendre du chiffrement:

- La confidentialité (le message reste privée)
- L'intégrité du message (le message est le bon)
- L'authentification (le correspondant est le bon)
- La non répudiation (ne pas nier avoir reçu un message et inversement)

* En fonction du ou des systèmes utilisés on va répondre + ou – à toutes ces attentes.
* Ce que nous allons voir dans les diapos suivantes n'est pas propre à HTTPS mais au chiffrement en général.

HTTPS


1. »Hello« Message
Login with CrypToken
2. »Hello« Message
X.509
X.509
3. Check certificate
- valid
- trusted CA
- integrity
- domain name
4. Client authentication
X.509
- client certificate
- key for symmetric encryption (e.g. AES)
- encrypted with server public key
X.509
5. Check certificate
- valid
- trusted CA
- check own certificate list
(e.g. LDAP-Server)
- integrity
6. Data transmission
encrypted
Access granted

HTTPS

Le protocole de prise de contact :

TLS va dans un premier temps à travers des paquets non chiffrés, permettre de choisir un ensemble d'attributs de sécurité pour les échanges et principalement de convenir d'une clé de chiffrement (échangée par des protocoles le permettant (DH) ou via chiffrement du client via la clé publique du serveur). Un ensemble structuré de message permettent ces échanges :

hello_request(0), client_hello(1), server_hello(2), certificate(11), server_key_exchange (12),certificate_request(13), server_hello_done(14), certificate_verify(15), client_key_exchange(16), finished(20)

HTTPS
clé secrète, clés privée publique… pourquoi, comment ?

Système à clé secrète:

Les 2 parties disposent d'une même clé. La clé permet de chiffrer et déchiffrer les messages.

Système à clés publiques:

Chacune des parties dispose d'une paire de clés privée et publique. Le chiffrement du message se fait via la clé publique uniquement déchiffrable avec la clé privée. Les clés sont donc liées a un modèle mathématique.

* Avantages, Inconvénients des deux systèmes…

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Système à clé secrète:

- La confidentialité est obtenue par le fait que **seul** les détenteurs de la clé sont susceptibles de chiffrer et déchiffrer le message.

- On garantie l'authentification (identifier mon correspondant) par challenge (on ne peut pas faire circuler la clé entre les correspondants).

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Système à clé secrète:

- L'intégrité d'un message ne peut pas reposer sur le seul fait du chiffrement (non obligatoire). Dans ce cas, il est nécessaire de produire une empreinte (code MAC ou résumé de message ou code condensé) fabriquée à partir de l’intégralité du message avec la clé secrète et généralement en y ajoutant un paramètre temporel (pour éviter qu’un message ne soit rejoué).

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Système à clé secrète:

- La non répudiation nécessite obligatoirement un tiers de confiance (le témoin). On passe obligatoirement par ce tiers qui garantie la non répudiation.

* En effet utilisant la même clé on peut accuser l'autre d'avoir créer un faux ou plus simplement de n'avoir rien reçu.

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

- Le partage et l'échange des clés secrètes constituent un nid à problèmes et c'est la raison principale qui suscita l'invention de systèmes d'échanges sécurisés des clés secrètes dont celui de DH et le fameux système à clé publique (clé publique / privée).

- **Il faut permettre l'échange de la clé en ligne** (échange public de clés secrètes).

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Une première méthode créer par Merkle a consistée à utiliser une méthode se basant sur un problème long à résoudre pour le pirate pour trouvée la clé privée partagée qui servira à crypter le message. La méthode de Merkle se basait sur un grand nombre d’association numéro de série et clé secrète de cryptage crypté et envoyé au destinataire. Le destinataire décode une des entrées (qui lui prend un certain temps) et renvoi le numéro de série à l’émetteur qui fait rapidement l’association avec la clé secrète, l’échange pouvait commencer. Le pirate doit quand à lui décrypter les entrées les unes après les autres jusqu'à découvrir l’association.

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Distribution des clés:

Alice envoie la base chiffrée à Bob. Bob connait la fonction inverse pour le déchiffrement

N° série 1 chiffré clé X / clé 1 associée chiffré clé X

N° série 2 chiffré clé Y / clé 2 associée chiffré clé Y

… (millions d'entrées)

Bob Choisit une entrée chiffrée, il déchiffre le NS et la clé. Il envoie le NS à Alice qui va rapidement connaître la clé à utiliser.

Celui qui tente de déchiffrer le message (le pirate) doit quand à lui déchiffrer toutes les entrées (ou presque) jusqu'à tomber sur le bon numéro de série donc sur la bonne clé.

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Une autre solution consiste à utiliser les mathématiques (arithmétique modulaire et les fonctions univoques) pour créer un problème plus difficile à résoudre dans un sens que dans l'autre. La méthode Diffie-Hellman-Merkle ou DH utilise cette méthode. DH peut être utilisé dans SSL et IPSec. DH permet de créer la clé secrète (de session) directement en ligne (clé symétrique).

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Chiffrement par clé publique (méthode dite à clé asymétrique)

Dans cette méthode il existe 2 clés distinctes : une publique distribuée librement (par exemple via un serveur LDAP) et une privée réservée à l'usage personnel de l'utilisateur qui les a générées (la clé priveé et publique) pour sa propre utilisation. L'idée d'une telle méthode revient à Diffie (article publié en 1976) mais RSA méthode à clé publique revient à Rivest, Shamir et Adleman.

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Chiffrement par clé publique (méthode dite à clé asymétrique)

DSA Digital Signature Algorithme ou DSS est une autre méthode mais dans ce cas le message transite en clair (ici c'est plus l'authentification qui nous intéresse que le chiffrement).

HTTPS (clé secrète, privée, publique…pourquoi, comment ?

Chiffrement par clé publique (méthode dite à clé asymétrique)

Cette méthode est consommatrice de ressources.

Elle est largement utilisée pour garantir l'authentification et ensuite échanger une clé secrète pour crypter les échanges.

HTTPS, Certification

La signature numérique :

L’utilisation de la clé privée pour le chiffrement équivaut à signer le message (la clé n’est pas partagée donc le codage via cette clé vaut signature !).

Un système à clé publique garantie :

- L’authentification (non certifier)
- L’intégrité
- La non répudiation
- Par contre seules les certificats permettent de valider une clé publique (la certifier !).

HTTPS, Certification

L’authentification des clés publiques : la certification !

Un certificat se compose de plusieurs parties :

- Du texte clair identifiant l’utilisateur et l’autorité proclamant la clé valide
- La clé publique de l’utilisateur distant (serveur, certificat de messagerie…)
- Le résumé du texte clair et la clé publique signé par la clé privée de l’autorité.

Vérification du certificat :

> Après avoir récupéré le certificat, l’application résume la partie info et la clé publique (refait le hash) puis utilise la clé publique de l’autorité (généralement contenue dans le navigateur pour les autorités racines) pour déchiffrer le résumé chiffré du certificat (chiffré par l’AC via la clé secrète) si il y a égalité entre les deux résumés le certificat est déclaré valide.
>
> On remarque que le client n’a pas besoin de clé publique pour vérifier les certificats présentés. Un certificat à une durée de vie et peut être révoqué avant la date d’expiration (la lCRL). La CRL est consultable par interrogation ou par pré-chargement (push). L’infrastructures la plus utilisée est x509 (PGP en est une autre). L’environnement d’utilisation de ces infrastructures s’appelle une PKI (Public Key Infrastructure).

http://www.delafond.org/traducmanfr/man/man1/openssl.1.html