Brève introduction à un ou deux protocoles applicatifs

Chaput Emmanuel

INP TOULOUSE ENSEEIHT

2020-2021 - Édition ”distanciation”

Notes :

Plan I

Notes :

Le DNS

Notes :

Introduction

Notes :

Qu'est-ce que le DNS ?

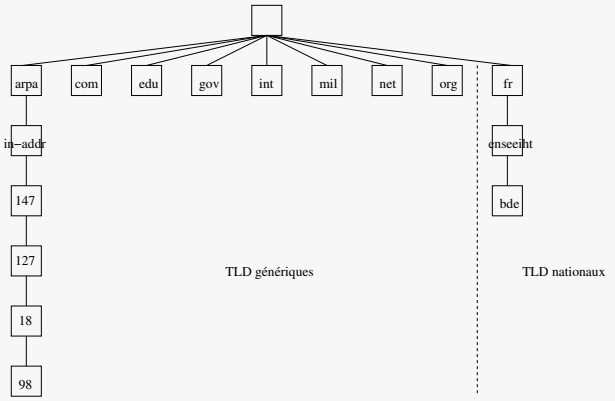
- *Domain Name System*
  - Service d'annuaire global de l'Internet
  - Correspondance entre des noms (*eg* `www.enseeiht.fr`) et des adresses IP (*eg* `193.48.203.34`)
- Fondé sur quelques concepts simples
  - Un espace de nommage arborescent
  - Une architecture client/serveur
  - Un protocole simple
- Doté de caratéristiques permettant un déploiement efficace
  - Passage à l'échelle (plus d'un milliard de noms annoncés en juillet 2015)
  - Hétérogénéité des constituants de l'Internet
- Défini depuis 1983 [**?**, **?**]
  - A subi de nombreuses évolutions [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]

Notes :

Organisation de l'espace de nommage

Notes :

Organisation de l'espace de nommage



Notes :

Notion de zone

- C'est une partie de l'espace de nommage
  - Cohérente vis-à-vis de l'arborescence (sous arbre)
  - Peut-être subdivisée en sous zones (*eg* `bde.enseeiht.fr`)
  - Gérée "indépendemment" des autres zones
- Le service de nommage de chaque zone est assuré par un ou plusieurs serveurs
  - Un serveur primaire
  - Au moins un serveur secondaire
- Résolution inverse (adresse vers nom) obtenu par une zone spécifique
  - `.arpa` avec `.in-addr.arpa` et `.ip6.arpa`

Notes :

Organisation des responsabilités

Notes :

Les différents TLDs

- Les ccTLDs (*Country Code* TLD)
  - Ce sont les domaines nationaux
  - Traditionnellement sur deux lettres (codes ISO 3166-1)
  - Délégation à une organisation spécifique (*eg* l'AFNIC pour le `.fr`)
- Les gTLDs (*Generic* TLD)
  - Les uTLDs (*Unsponsored* TLD)
    - Peuvent être obtenus par une procédure classique
    - *eg* `.com`, `.net`, `.org`, ...
  - Les sTLDs (*Sponsored* TLD)
    - Sont gérés par un *sponsor* qui définit une charte
    - *eg* `.aero`, `.museum`, ...
- Le `.arpa`
  - Pour la résolution inverse

Notes :

L'architecture client/serveur

Notes :

L'architecture client/serveur

- Le client est appelé un *resolver*
  - Utilisé par de nombreuses applications
  - Rarement utilisé directement par l'utilisateur
  - Peut mettre en place un cache
- Le serveur répond aux requêtes des clients
  - Peut répondre directement
  - Peut s'adresser lui aussi à un autre serveur
  - Peut mettre en place un cache

Notes :

Deux types de serveurs

- Serveur maître (ou primaire)
  - Il dispose de la base de données des zones qu'il couvre
  - Il répond aux requêtes des clients
  - Il transfère les bases de données aux secondaires
- Serveur esclave (ou secondaire)
  - Il ne dispose que d'une copie de la base de données
  - Il répond aux requêtes des clients
  - Il obtient la base de données et ses mises à jour depuis le maître

Notes :

Comportement d'un serveur

Lors de la réception d'une requête, un serveur peut

- Fournir "directement" la réponse
  - Après avoir effectué des requêtes auprès d'autres serveurs
- Fournir "directement" la réponse
  - En utilisant ses caches
- Fournir un élément permettant d'obtenir la réponse (*referral*)
  - L'adresse d'un serveur permettant d'obtenir la réponse
- Envoyer un message d'erreur

Un *referral* n'est pas une réponse

- C'est l'adresse d'un ou plusieurs serveurs ayant autorité sur une zone permettant de s'approcher de la réponse

Notes :

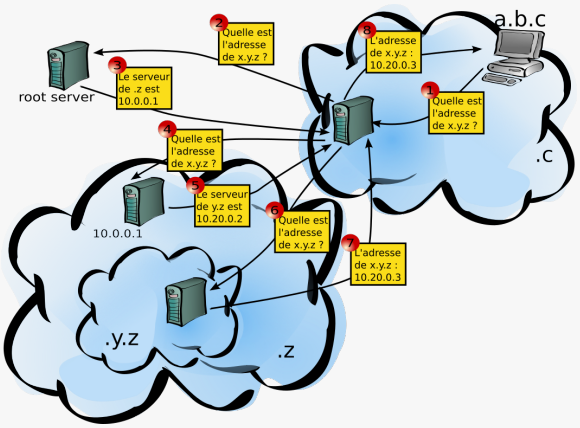
Réponse itérative ou récursive

Lorsqu'un serveur reçoit une requête de la part d'un client, il peut

- Lui donner la répons s'il la connaît
  - *eg* il a autorité sur la zone
- Lui donner l'adresse d'un serveur qui pourra lui donner la réponse
  - Traitement itératif de la requête
  - Le client progresse de serveur en serveur
- Obtenir lui même la réponse et la lui fournir
  - Traitement récursif de la réponse
  - Le serveur se comporte comme un client vis-à-vis des autres serveurs
  - Le client n'est pas directement impacté par la complexité de la requête

Notes :

Réponse itérative ou récursive



Notes :

Le client dans le monde Unix

- Utilisé au travers de fonctions de la librairie C telles que [**?**]

```
struct hostent *gethostbyname(const char *name);
int getaddrinfo(const char *node,
                const char *service,
                const struct addrinfo *hints,
                struct addrinfo **res);
```

- Paramétré par des fichiers tels que [**?**, **?**]

```
/etc/gai.conf
/etc/resolv.conf
```

- Un client en ligne de commande existe

```
$ dig www.enseeiht.fr
...
```


Notes :

Le protocole

Notes :

Les *Resource Records*

Notes :

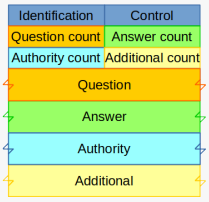
Les *Resource Records*

Description des propriétés et éléments d'une zone [**?**]. Ce sont par exemple

- *SOA Record* (*Start Of Authority*) [**?**]
  - Définition des propriétés générales de la zone
- *A Record* (*IPv4 address*) [**?**]
  - Donne l'adresse IPv4 d'un nom
- *CNAME Record* (*Canonical Name*) [**?**]
  - Donne le nom pour une adresse
- *AAAA Record* (*IPv6 address*) [**?**]
  - Donne l'adresse IPv6 d'un nom
- *NS Record* (*Name Server*) [**?**]
  - Définition du serveur d'une zone

Notes :

Format des messages



`Identification` permet de mettre en relation requête et réponse

`Control` définit le type de message (requête, réponse, récursion, ...)

`Question count` donne le nombre de questions dans le message

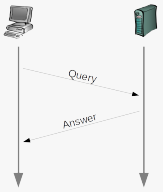
...

`Question` liste les requêtes

...

Notes :

Les échanges



- Le protocole client/serveur est particulièrement simple
- Le client envoie une requête au serveur
- Le serveur répond en intégrant la requête dans la réponse
- On peut donc transmettre plusieurs requêtes “simultanément”

Notes :

Un exemple d’échange (1)

```
$ dig www.isae.fr

; <<>> DiG 9.9.5-11ubuntu1.1-Ubuntu <<>> www.isae.fr
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29230
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.isae.fr. IN A

;; ANSWER SECTION:
www.isae.fr. 84386 IN CNAME drupalprod.isae.fr.
drupalprod.isae.fr. 84386 IN A 192.93.254.85

;; AUTHORITY SECTION:
isae.fr. 74348 IN NS supns.isae.fr.
isae.fr. 74348 IN NS filaong.isae.fr.

;; ADDITIONAL SECTION:
supns.isae.fr. 21368 IN A 192.93.254.35
filaong.isae.fr. 21368 IN A 192.93.254.38

;; Query time: 1 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Wed Jan 06 11:48:27 CET 2016
;; MSG SIZE  rcvd: 155
```


Notes :

Un exemple d’échange (2)

```
11:36:34.225066 IP (tos 0x0, ttl 64, id 26673, offset 0, flags [DF], proto UDP (17), length 68)
    fwirit-143.enseeiht.fr.15918 > ns1.enseeiht.fr.domain:
    [udp sum ok] 20154+ [1au] A? www.isae.fr. ar: .
    OPT UDPsize=4096 (40)
11:36:34.225871 IP (tos 0x0, ttl 61, id 62821, offset 0, flags [none], proto UDP (17), length 183)
    sivuca.laplace.enseeiht.fr.domain > fwirit-143.enseeiht.fr.15918:
    [udp sum ok] 20154
    q: A? www.isae.fr.
    2/2/3
    www.isae.fr. [23h38m20s] CNAME drupalprod.isae.fr.,
    drupalprod.isae.fr. [23h38m20s] A 192.93.254.85
    ns:
    isae.fr. [20h51m2s] NS supns.isae.fr.,
    isae.fr. [20h51m2s] NS filaong.isae.fr.
    ar:
    supns.isae.fr. [6h8m2s] A 192.93.254.35,
    filaong.isae.fr. [6h8m2s] A 192.93.254.38, .
    OPT UDPsize=4096 (155)
```


Notes :

Le protocole HTTP

Notes :

Introduction

Notes :

Qu'est-ce que HTTP ?

- Un protocle applicatif
  - *HyperText Transfer Protocol*
  - Initialement dédié aux système d'information hypermédia
  - Premiers travaux au CERN (par Tim Berners-Lee) fin des années 80
  - Une des clefs de voute du *Web*, devenue quasi incontournable
  - Mis à contribution pour des usages très variés (VoD, . . .)
  - Protocole sans état (*stateless*)
- Modèle client serveur
  - Classiquement, un navigateur et un "serveur Web"
  - Le client demande des ressources (fichiers HTML, . . .) au serveur
  - Un serveur peut se comporter comme un client (*proxy*) pour des raisons de performances, sécurité, . . .
- Standardisé par l'IETF
  - Protocole en mode texte et utilisant TCP
  - HTTP 1.0 [**?**]
  - HTTP 1.1 [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**]
  - HTTP 2.0 [**?**]
  - HTTP 3.0 (pas encore de RFC)

Notes :

Notion d'URI

Notes :

Notion d'URI

- *Uniform Resource Identifier*
  - Identification non ambigüe d'une ressource
  - Pour l'identifier : URN (*Name*)
  - Ou pour la localiser : URL (*Locator*)
- Syntaxe :

`scheme:[//authority]path[?query][#fragment]`

`scheme`	définit le type d'URI (les types sont définis par l'IANA)
`authority`	décrit un éventuel mécanisme d'identification et un hôte
`path`	permet de décrire un classement arborescent
`query`	permet de passer des paramètres de recherche
`fragment`	permet d'accéder à une sous partie de la ressource

- Exemples

```
http://www.enseeiht.fr/
mailto:darth.vader@death.star
tel:+33534322231
```


Notes :

Les requêtes et les réponses

Notes :

Les requêtes et les réponses

- Les ressources sont identifées par des URL
- Le client envoie une requête au serveur
  - Exemple “*Donne moi le fichier toto.html*”
  - `GET toto.html`
- Le serveur traite la demande
  - Il envoie les données
  - Il refuse si c’est impossible (code d’erreur)
- Requêtes et réponses peuvent contenir des informations complémentaires
  - Version d’HTTP
  - Description du logiciel
  - Paramètres de la conversation (format, langue, . . .)

Notes :

Notion de requête

```
$ telnet www.empire.com 80
Trying 66.66.66.66...
Connected to www.empire.com.
Escape character is '^]'.
GET /map_of_death_star
<!DOCTYPE HTML PUBLIC ``-//IETF//DTD HTML 2.0//EN''>
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /map_of_death_star was not found on this
server.</p>
<hr>
<address>Apache Server at 66.66.66.66 Port 80</address>
</body></html>
Connection closed by foreign host.
```


Notes :

Format d’une requête

```
GET / HTTP/1.1
Host: www.enseeiht.fr
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: qkfgjsqhfgjhqdsfgjh ...
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

- Une description de la requête
  - Ici une requête GET
- Des champs d’entête
  - L’hôte cible, le descriptif du client, les formats acceptés, . . .
- Éventuellement un contenu
  - Absent ici

Notes :

Les requètes

- Définies initialement par HTTP/1.0
  - GET permet d'obtenir des ressources (typiquement un fichier décrit par une URL)
  - HEAD permet d'obtenir des informations sur une ressource
  - POST permet au client de transmettre des données au serveur
- Ajouts de HTTP/1.1
  - PUT permet d'envoyer des ressources depuis le client vers le serveur
  - OPTIONS permet de connaître les capacités d'un serveur
  - DELETE permet de détruire une ressource
  - TRACE demande au serveur de renvoyer ce qu'il a reçu
  - CONNECT permet de mettre en place un tunnel IP

Notes :

La requête POST

- Permet de transférer de l'information *vers* le serveur
  - Le contenu d'un formulaire [**?**]
  - Le contenu d'un fichier (*upload*)
- Encodage des données dans les attributs de la méthode
  - Affectation de variables

```
POST /AccessDeathStarBluePrint HTTP/1.1
Host: foo.example
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

Login=vader&Password=iamyourfather
```


Notes :

Une session HTTP

Notes :

Notion de session

- HTTP est fondé sur la notion de *session*
  - Ensemble de requêtes (du client) et de réponses (du serveur)
  - Utilise une connexion TCP
  - Traditionnellement, le serveur utilise le port 80

```
$ telnet www.untruc.com 80
Trying 193.48.203.34...
Connected to www.untruc.com.
Escape character is '^]'.
GET /
<html>
<head></head>
<body>
        Ce site n'est␣pas␣encore␣en␣ligne.
␣␣␣␣␣␣␣␣</body>
Connection␣closed␣by␣foreign␣host.
$
```

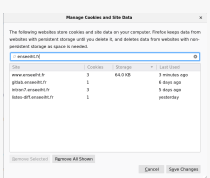

Notes :

Les cookies

Notes :

Les cookies

- Qu'est-ce qu'un *cookie* ?
  - Des informations envoyées par le serveur
  - Stoquées par le client
  - Permettre de contourner l'aspect *stateless*
- Quelle est l'utilité ?
  - Mettre en place la notion de session
    - Naviguer sur un site (historique des pages visitées)
    - Garder un "panier"
    - S'identifier
    - Publicité



Notes :

Comment les cookies sont mis en place ?

- De quoi est composé un cookie ?
  - Un nom
  - Une valeur
  - Des attributs (date de validité, domaine, . . .)
- Échangé dans l'entête des messages HTTP
  - Le serveur les envoie au client avec `Set-Cookie`

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=example.com
```

  - Le client doit les renvoyer avec `Cookie`

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

- Définis dans [**?**]
  - *HTTP State Management Mechanism*

Notes :

HTTP et les connexions TCP

Notes :

Session HTTP et connexion TCP

- Dans HTTP 1.0 une connexion TCP c'est
  - Une session HTTP
  - Une requête/une réponse
- Simple à mettre en œuvre
  - Pas de délimitation des données
- Peu efficace, même pour une page web
  - Plusieurs documents impliqués (images, . . .)
  - Souvent sur le même serveur
  - Un établissement et une fin de connexion à chaque fois
  - Slow-start à chaque fois

Notes :

Les connexions persistantes

- HTTP 1.1 introduit la notion de *connexion persistante*
  - Plusieurs couples requête/réponse dans la même connexion
- Objectif
  - Ne pas redémarrer une connexion pour chaque transfert
  - Éviter toute la lourdeur du démarrage
- Activation du mécanisme
  - Dans HTTP 1.0
    - Utilisation du paramètre `Connection: keep-alive`
  - Dans HTTP 1.1
    - Comportement par défaut

Notes :

Phénomène du *Head of Line Blocking*

- Une session trop longue bloque tout ce qui suit
  - *eg* accès à une base de données, traitements lourds, ...
- Pourtant d'autres sessions pourraient être mises en place
  - L'ordre de tranfert des différentes ressources n'est pas contraint
- HTTP 1.1 introduit la notion de *pipelining* [**?**]
  - Plusieurs requêtes peuvent être transmises sans attendre de réponse
- Le phénomène se produit même si une nouvelle connexion est utilisée pour chaque session
  - Car le nombre de connexions simultanées est limité

Notes :

HTTP/2 et HTTP/3

Notes :

Histoire

- Proposition de Google (SPDY en 2010)
  - Implanté dans Chromium
  - “Abandonné” en 2016 à la faveur de HTTP/2
- Objectif : lever les limites de HTTP
  - Une requête par connexion (on peut pipeliner)
  - Entièrement à l’initiative du client
  - Entêtes lourds et redondants
- Standardisé par l’IETF
  - Dans le groupe de travail *httpbis*
  - RFC 7540 [**?**]

Notes :

Les éléments de base de HTTP/2

- *Frame*
  - L’élément de base de communication
  - Différents types
  - Entête, données, . . .
- *Message*
  - Une suite de *Frame*s
  - Constitue un message complet
  - *eg* une requête, une réponse
- *Stream*
  - Une communication bidirectionnelle
  - Constituée d’une séquence de *message*s
  - Caractérisée par un identifiant

Notes :

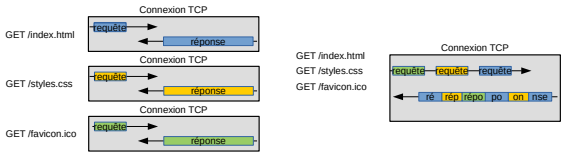
Le multiplexage

- La notion de *stream* est à la base du multiplexage
  - Les requêtes comme les réponses peuvent être transmises en parallèle
  - Inutile de “sérialiser”
  - Disparition du *head-of-line blocking*
- Une seule connexion TCP
  - En cas de perte, la connexion est “bloquée” jusqu’à la résolution
  - TCP ne délivre que dans l’ordre
  - Réapparition du *head-of-line blocking*
- Prioritisation entre les flux
  - Structure de dépendance arborescence
  - Chaque nœud est moins prioritaire que son père
  - Partage pondéré dans chaque ensemble de nœuds

Notes :

Le multiplexage

- Plusieurs requêtes transmises dans la même connexion TCP
- Sans attendre la réponse à la requête précédente
- Les *frames* constituant une réponse sont transmises lorsqu'elles sont prêtes



Notes :

Le PUSH

- Objectif
  - Transférer des informations sans demande préalable du client
  - Permettre une meilleure dynamique sur le client
  - "Échec" si les informations sont inutiles pour le client
- Exemple
  - Le client récupère le fichier `page.html`
  - Il va demander `style.css` et d'autres fichiers
  - Le serveur peut prendre l'initiative
- Techniquement
  - Le serveur envoie une *frame* de type `PUSH_PROMISE` dans un *stream* initié par le client
  - Il y envoie ensuite les ressources effectivement demandées par le client
  - Il transmet les données dans une *stream* qu'il a initié
  - Le client peut arrêter ce transfet (*frame* `RST_STREAM`)

Notes :

Vers HTTP/3

- HTTP/2 utilise une seule connexion TCP
  - En cas de problème réseau, tout est bloqué
  - TCP délivre dans l'ordre, alors qu'il n'y a pas d'ordre entre les streams
  - Finalement les connexions multiples de HTTP/1 avaient du bon
  - Quasi impossible de faire évoluer TCP
- HTTP/3 se fonde sur QUIC qui utilise UDP
  - Chaque stream sera traité indépendemment des autres
  - Pas de blocage global lié à TCP
  - Possibilité de démarrer plusieurs communications rapidement (1 voire 0 RTT)

Notes :

Le protocole QUIC

- *Quick* UDP *Internet Connections*
  - Encore à l'état de *draft* en septembre 2020 (version 31)
- Sécurité intégrée "nativement"
  - Tout est chiffré, même les informations que gérait TCP
  - HTTP utilisait TLS (ex SSL) pour constituer HTTPS
  - Meilleure protection contre les attaques "*Man in the middle*"
- Mise en place de la connexion plus rapide
  - Première connexion en 2 RTT
  - Connexion en 1 RTT si les clefs sont connues
  - Connexion en 0 RTT si les machines viennent de communiquer
- Suppression du *Head of Line Blocking*
  - Les données **d'un *stream*** sont délivrées dans l'ordre
- Multiplexage de *streams*
- Négociation de version
- ...

Notes :

Le protocole HTTP/3

- Encore à l'état de *draft* en septembre 2020 (version 31)
  - Comme son partenaire QUIC
  - Mais déjà déployé
  - Proposition de *google* de 2013
- Service équivalent à HTTP/2
  - Mais utilisant QUIC pour le multiplexage de *stream*
  - On se passe de TCP et TLS
- Une requête/réponse utilise un *stream* QUIC
  - Elles peuvent donc être multiplexées comme avec HTTP/2
  - Chaque *stream* est composé de *frames*
- Performances accrues
  - Meilleure dynamique des accés Web

Notes :

[1] D. Eastlake 3rd.
Domain Name System Security Extensions.
RFC 2535 (Proposed Standard), March 1999.
Obsoleted by RFCs 4033, 4034, 4035, updated by RFCs 2931, 3007, 3008, 3090, 3226, 3445, 3597, 3655, 3658, 3755, 3757, 3845.

[2] D. Eastlake 3rd.
Domain Name System (DNS) Case Insensitivity Clarification.
Technical Report 4343, January 2006.

[3] D. Eastlake 3rd and C. Kaufman.
Domain Name System Security Extensions.
RFC 2065 (Proposed Standard), January 1997.
Obsoleted by RFC 2535.

[4] M. Andrews.
Negative Caching of DNS Queries (DNS NCACHE).
RFC 2308 (Proposed Standard), March 1998.
Updated by RFCs 4035, 4033, 4034, 6604.

Notes :

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.
DNS Security Introduction and Requirements.
Technical Report 4033, March 2005.
Updated by RFCs 6014, 6840.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.
Protocol Modifications for the DNS Security Extensions.
Technical Report 4035, March 2005.
Updated by RFCs 4470, 6014, 6840.

[7] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose.
Resource Records for the DNS Security Extensions.
Technical Report 4034, March 2005.
Updated by RFCs 4470, 6014, 6840, 6944.

[8] A. Barth.
HTTP State Management Mechanism.
Technical Report 6265, April 2011.

[9] M. Belshe, R. Peon, and M. Thomson.
Hypertext Transfer Protocol Version 2 (HTTP/2).

Notes :

RFC 7540 (Proposed Standard), May 2015.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk.
Hypertext Transfer Protocol – HTTP/1.0.
RFC 1945, IETF, May 1996.

[11] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson.
A Means for Expressing Location Information in the Domain Name System.
RFC 1876 (Experimental), January 1996.

[12] R. Draves.
Default Address Selection for Internet Protocol version 6 (IPv6).
RFC 3484 (Proposed Standard), February 2003.
Obsoleted by RFC 6724.

[13] R. Elz and R. Bush.
Serial Number Arithmetic.
RFC 1982 (Proposed Standard), August 1996.

[14] R. Elz and R. Bush.

Notes :

Clarifications to the DNS Specification.
RFC 2181 (Proposed Standard), July 1997.
Updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452.

[15] C.F. Everhart, L.A. Mamakos, R. Ullmann, and P.V. Mockapetris.
New DNS RR Definitions.
Technical Report 1183, October 1990.
Updated by RFCs 5395, 5864, 6195, 6895.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee.
Hypertext Transfer Protocol – HTTP/1.1.
RFC 2068 (Proposed Standard), January 1997.
Obsoleted by RFC 2616.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee.
Hypertext Transfer Protocol – HTTP/1.1.
RFC 2616, IETF, June 1999.

[18] R. Fielding, Y. Lafon, and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Range Requests.

Notes :

RFC 7233 (Proposed Standard), June 2014.

[19] R. Fielding, M. Nottingham, and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Caching.
RFC 7234 (Proposed Standard), June 2014.

[20] R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Authentication.
RFC 7235 (Proposed Standard), June 2014.

[21] R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Conditional Requests.
RFC 7232 (Proposed Standard), June 2014.

[22] R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Message Syntax and Routing.
RFC 7230 (Proposed Standard), June 2014.

[23] R. Fielding and J. Reschke.
Hypertext Transfer Protocol (HTTP/1.1) : Semantics and Content.

Notes :

RFC 7231 (Proposed Standard), June 2014.

[24] R. Gilligan, S. Thomson, J. Bound, and W. Stevens.
Basic Socket Interface Extensions for IPv6.
RFC 2553 (Informational), March 1999.
Obsoleted by RFC 3493, updated by RFC 3152.

[25] F. Gont.
Security Assessment of the Internet Protocol Version 4.
Technical Report 6274, July 2011.

[26] E. Lewis.
The Role of Wildcards in the Domain Name System.
RFC 4592 (Proposed Standard), July 2006.

[27] E. Lewis and A. Hoenes.
DNS Zone Transfer Protocol (AXFR).
RFC 5936 (Proposed Standard), June 2010.

[28] B. Manning.
DNS NSAP RRs.

Notes :

Technical Report 1348, July 1992.
Obsoleted by RFC 1637.

[29] L. Masinter.
Returning Values from Forms : multipart/form-data.
RFC 7578 (Proposed Standard), July 2015.

[30] P.V. Mockapetris.
Domain names : Concepts and facilities.
Technical Report 882, IETF, November 1983.
Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[31] P.V. Mockapetris.
Domain names : Implementation specification.
Technical Report 883, IETF, November 1983.
Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[32] P.V. Mockapetris.
Domain names - concepts and facilities.
Technical Report 1034, November 1987.

Notes :

Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[33] P.V. Mockapetris.
Domain names - implementation and specification.
Technical Report 1035, November 1987.
Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[34] P.V. Mockapetris.
DNS encoding of network names and other types.
Technical Report 1101, April 1989.

[35] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley.
Network performance effects of http/1.1, css1, and png.
SIGCOMM Comput. Commun. Rev., 27(4) :155–166, October 1997.

[36] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi.

Notes :

DNS Extensions to Support IP Version 6.
RFC 3596 (Draft Standard), October 2003.

Chaput Emmanuel Brève introduction à un ou deux protocoles ap 2020-2021 - Édition "distanciation" 53 / 53

Notes :

Notes :