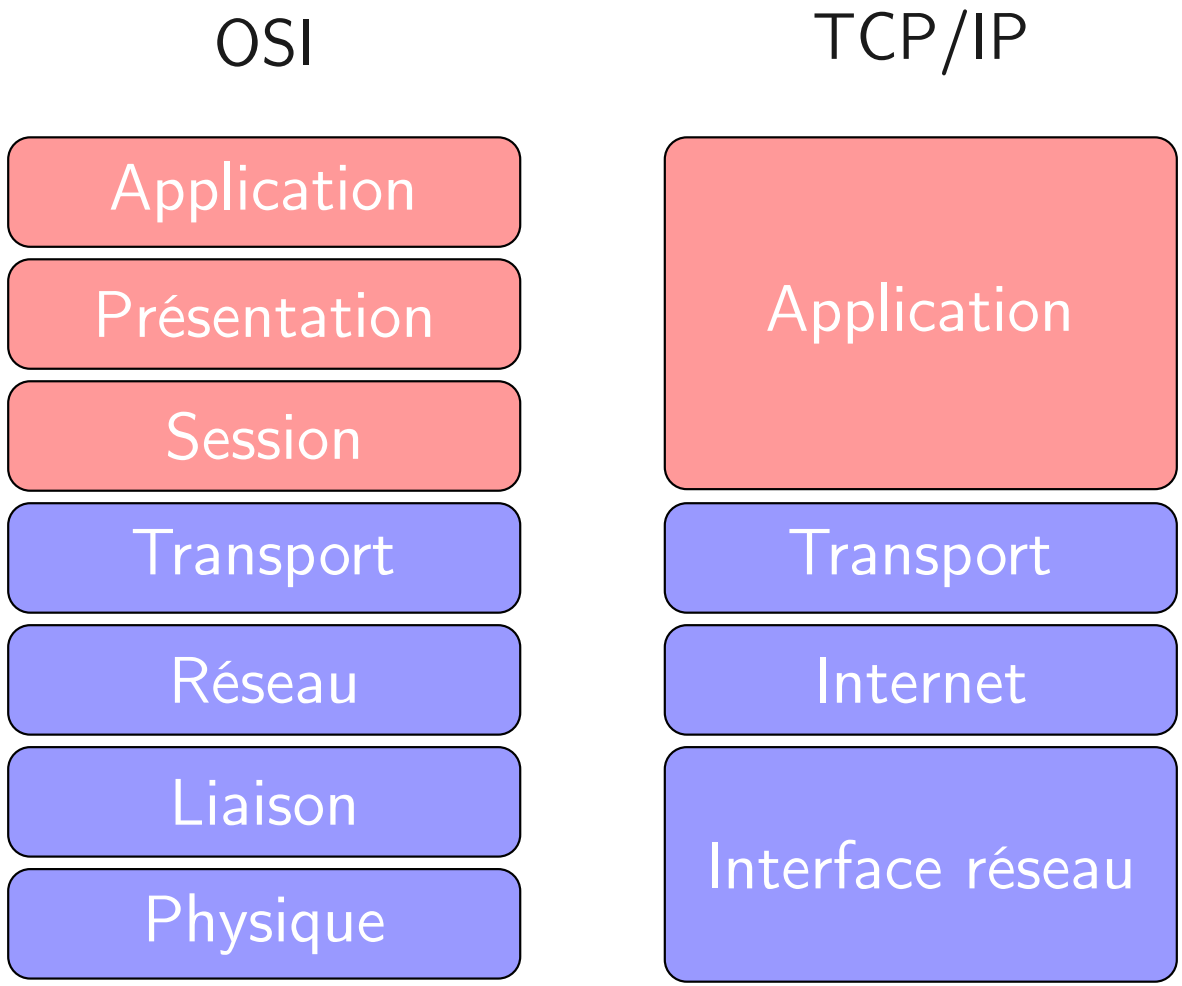
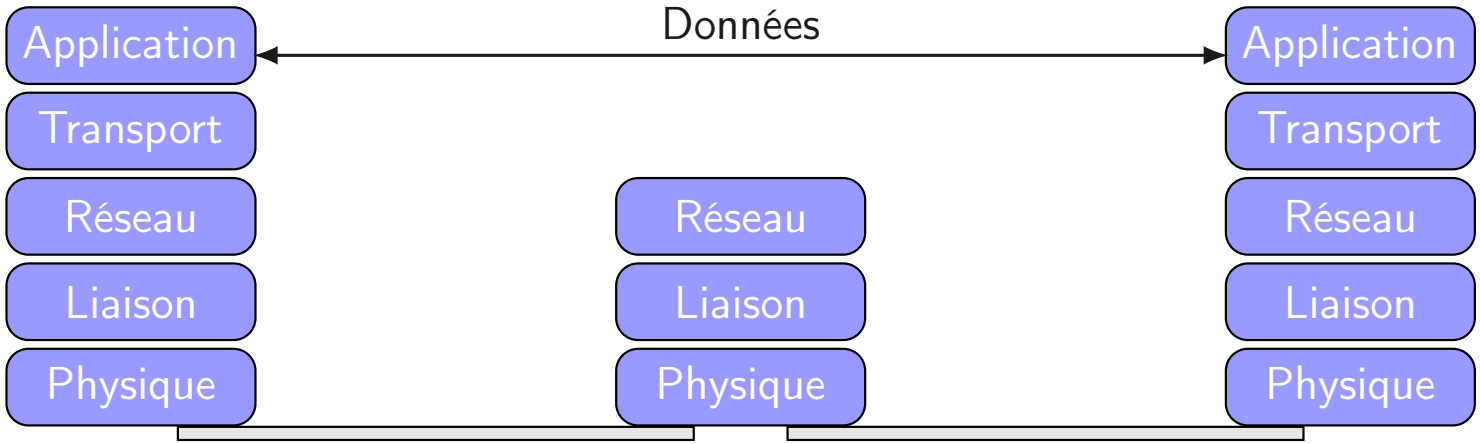
Couche Application

Sophie Robert, Université d'Orléans

Les modèles


OSI
TCP/IP
Application
Présentation
Session
Transport
Réseau
Liaison
Physique
Application
Transport
Internet
Interface réseau

Couche application (Application)



Les applications que vous connaissez avec leurs protocoles :

- Mail : SMTP
- Web : HTTP
- Chat : XMPP
- Envoi de documents : FTP
- ...

Couche application

Interactions entre hôtes dans le but de délivrer un service à l'utilisateur :

- échange de données (fichier, pages web, mail,...)
- ou d'informations sur le réseau (DNS, DHCP).
- on suppose que toute la machinerie sous-jacente est fonctionnelle : on sait envoyer un message (du texte, une chaîne d'octets,...) à une adresse IP existante donnée.

Les protocoles de la couche application décrivent ces interactions.

Application réseau

Une application réseau, c'est :

- Un ou plusieurs protocoles mais aussi...
- des formats, des normes...
- des applications logicielles, en particulier un agent utilisateur (UA user agent).

Exemple du web :

- HTTP, HTTPS
- HTML, CSS, Javascript
- Navigateur web comme UA (Opera...)
- serveur web (Apache...).

Organisation : client-serveur

Le modèle le plus courant est l'architecture client-serveur :

- le serveur est un hôte toujours connecté et disponible ;
- le client se connecte ponctuellement et envoie des requêtes au serveur.
- En général, plusieurs clients, un seul serveur.

Exemples : clients et serveurs web ou mail.

Client-serveur

- Un protocole sur ce modèle décrit les échanges entre client et serveur : synchronisation, taille et contenu des messages...
- Pour être joignable, le serveur doit en général disposer d'une IP fixe.
- Au niveau logiciel : une application serveur, une application client.

Client-serveur

- Pour des questions de capacité ou de garantie de service, plusieurs serveurs peuvent se répartir les requêtes.
- Un équilibreur de charge répartit les demandes sur les différents serveurs

Autre modèle

- Autre paradigme : P2P, les données sont distribuées et les communications ne sont pas centralisées.
- Chaque hôte peut être client ou serveur, voire les deux en parallèle, donc ponctuellement, on peut se ramener au modèle client-serveur.
- BitTorrent, Bitcoin

Design : format

L'application doit spécifier le format des données échangées :

- formats de données :
  - Textuelles : HTTP, SMTP ...
  - XML : SVG, messages XMPP ...
  - JSON (API Web modernes) ...
  - binaire / hexadécimal : FTP ...
- Délimitation des messages : séparateurs (ex : CRLF en HTTP) ou taille explicite
- des codes de retour : succès / erreur (ex : 200 OK, 404 Not Found)

Design : synchronisation

L'application doit décrire les échanges de messages :

- Mode connecté : échange de questions/réponses dans une connexion établie à l'avance. Protocole TCP de la couche transport pour échanger un flux de données. Exemples : HTTP, SMTP, FTP.

- Mode non connecté : envoi de datagrammes par le protocole UDP de la couche transport. Pas de garantie d'acheminement du datagramme, l'application doit pouvoir fonctionner avec pertes. Exemples : DNS, streaming.

En demandant un service spécifique à la couche transport (débit, sécurité, garantie de transfert), on choisit des contraintes : structure forte ou absence de garantie par exemple.

Sockets

Pour concevoir une application réseau, on utilise des sockets : application logicielle servant d'interface entre le système d'exploitation et le réseau. Une socket est la donnée :

- d'un protocole (UDP ou TCP en général)
- de l'adresse de transport de l'hôte local (IP + port)
- de l'adresse de transport de l'hôte distant (IP + port)

Une fois établie, une socket est donc identifiée par quatre données : deux adresses IP et deux ports.

Sockets

Plusieurs types de sockets selon l'usage :

- Pour le protocole TCP : socket stream.
- Pour le protocole UDP : socket datagram.
- Pour l'accès à la couche réseau : socket raw (pas de protocole de couche transport).

En TP, on travaillera avec le premier type.

Sockets : implémentation

- Différentes implémentations selon les systèmes d'exploitation : BSD sockets pour Linux.
- Des normes sont définies (POSIX : ensemble de normes pour les systèmes d'exploitation de type UNIX).
- API (Application Programming Interface) définies.
- java.net.Socket en Java, module Socket en Python.

TP Sockets

Lors des deux séances de TP à venir :

- Prise en main de l'API asyncio/stream (interface haut niveau avec les connexions réseaux).
- Programmation d'applications client ou serveur pour différents protocoles.
- Gestion de la concurrence (asyncio) pour les connexions en parallèle.

Rappel async/await

```
import asyncio
import time
async def affiche(msg, delai) :
    await asyncio.sleep(delai)
    print(msg)
async def attente() :
    await affiche('Hello', 5)
    await affiche('World', 2)
print(f"started at {time.strftime('%X')}")
asyncio.run(attente())
print(f"finished at {time.strftime('%X')}")
```

- async pour déclarer une coroutine qui peut être *attendable/waitable*.
- Les coroutines doivent être appelées avec le mot clé await indiquant qu'elles peuvent être suspendues.

Rappel async/await

```
import asyncio
import time
async def affiche(msg, delai) :
    await asyncio.sleep(delai)
    print(msg)
async def concurente() :
    task1 = asyncio.create_task(affiche('hello', 5))
    task2 = asyncio.create_task(affiche('world', 2))
    await task1
    await task2
print(f"started at {time.strftime('%X')}")
asyncio.run(concurente())
print(f"finished at {time.strftime('%X')}")
```

- *create_task* de asynchio permet d'exécuter des coroutines de manière concurrente (boucle d'événements)

Côté Serveur

```
async def multiplication_server() :
    # lancement du socket serveur
    # sercice rendu si demande de connexion
    # '0.0.0.0' en ecoute sur toutes ses adresses
    # ouvert sur le port 46464
    server = await asyncio.start_server(service,
                                        '0.0.0.0', 46464)
    addr = server.sockets[0].getsockname()
    print(f'Serving on {addr}')
    # Gestion automatique serveur en ecoute infini.
    async with server :
        await server.serve_forever()

if __name__ == '__main__' :
    # Execution de la boucle d'evenements
    asyncio.run(multiplication_server())
```

Côté Serveur

```
async def service(reader, writer) :
    # (addr,port) du socket client
    addr = writer.get_extra_info('peername')
    print(f"nouvelle demande de {addr[0]}")
    # lecture d'un message avec comme
    # separateur le retour a la ligne
    op1 = await reader.readline()
    op2 = await reader.readline()
    res = int(op1.decode())*int(op2.decode())
    msg = str(res)
    # Envoi d'un message
    writer.write(msg.encode())
    # Fermeture de la connexion
    writer.close()
    # Attente de la fin de connexion
    await writer.wait_closed()
```

Côté Client

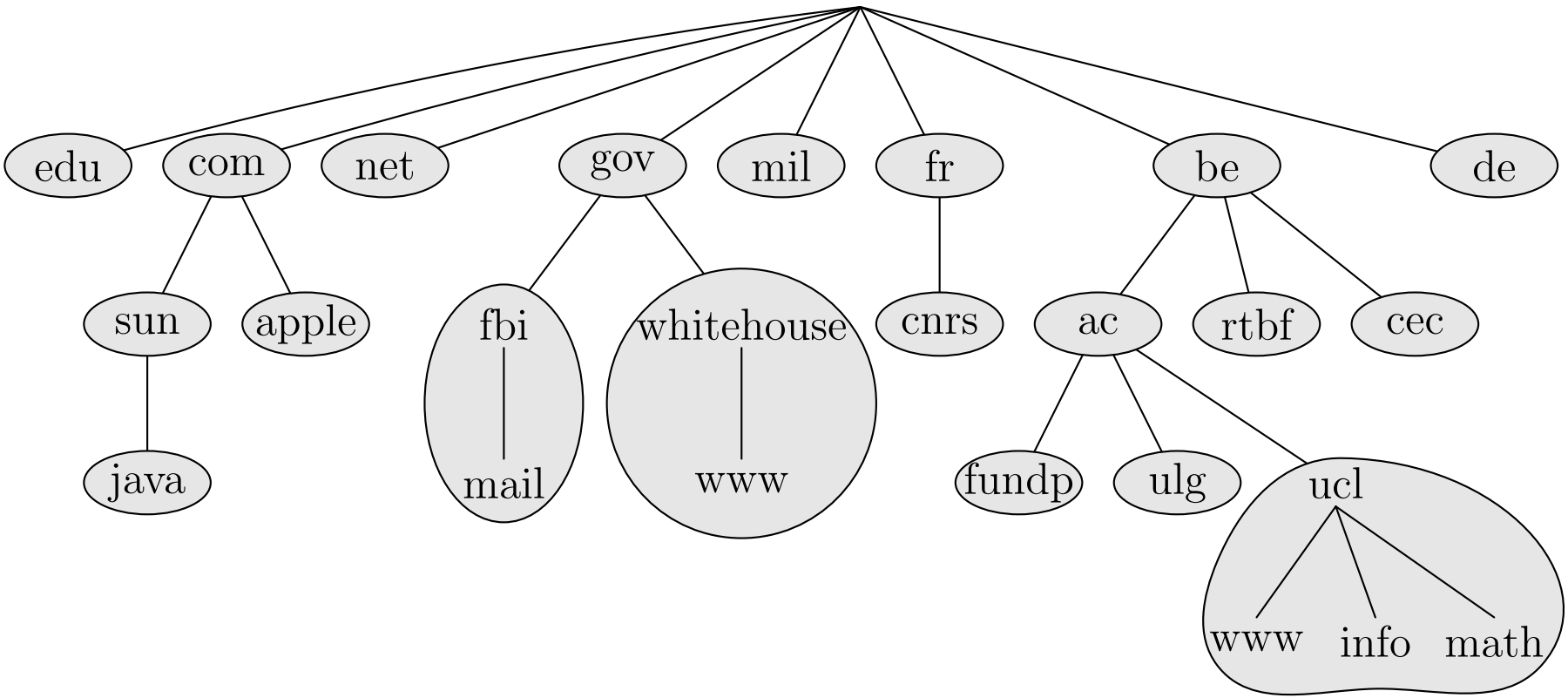
```
if __name__ == '__main__' :
    if len(argv) !=5 :
        exit(1)
    sname=argv[1]
    server=gethostbyname(sname)
    port=argv[2]
    op1 = argv[3]
    op2 = argv[4]
    print("connecting to  :", sname, server)
    asyncio.run(multiplication(server,port,op1,op2))
```

Côté Client

```
async def multiplication(server,port,op1,op2) :
    # connexion au serveur et
    #creation de deux streams
    reader, writer =
        await asyncio.open_connection(server, port)
    # string message convertit en bytes
    msg = op1+"\n"
    writer.write(msg.encode())
    msg = op2+"\n"
    writer.write(msg.encode())
    #vide le tampon et envoie
    await writer.drain()
    #lecture de la reponse
    data = await reader.read()
    print("res=",data.decode())
```

10.72613…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [SYN] Seq=0 Win=64240
10.72617…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [SYN, ACK] Seq=0 Ack=1
10.72619…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [ACK] Seq=1 Ack=1 Win=
10.72648…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [PSH, ACK] Seq=1 Ack=1
10.72649…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [ACK] Seq=1 Ack=3 Win=
10.72650…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [PSH, ACK] Seq=3 Ack=1
10.72651…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [ACK] Seq=1 Ack=5 Win=
10.72670…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [PSH, ACK] Seq=1 Ack=5
10.72672…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [ACK] Seq=5 Ack=4 Win=
10.72680…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [FIN, ACK] Seq=4 Ack=5
10.72939…	192.168.1.2	192.168.1.1	TCP	50904 → 46464 [FIN, ACK] Seq=5 Ack=5
10.72942…	192.168.1.1	192.168.1.2	TCP	46464 → 50904 [ACK] Seq=5 Ack=6 Win=

Domain Name Service - DNS

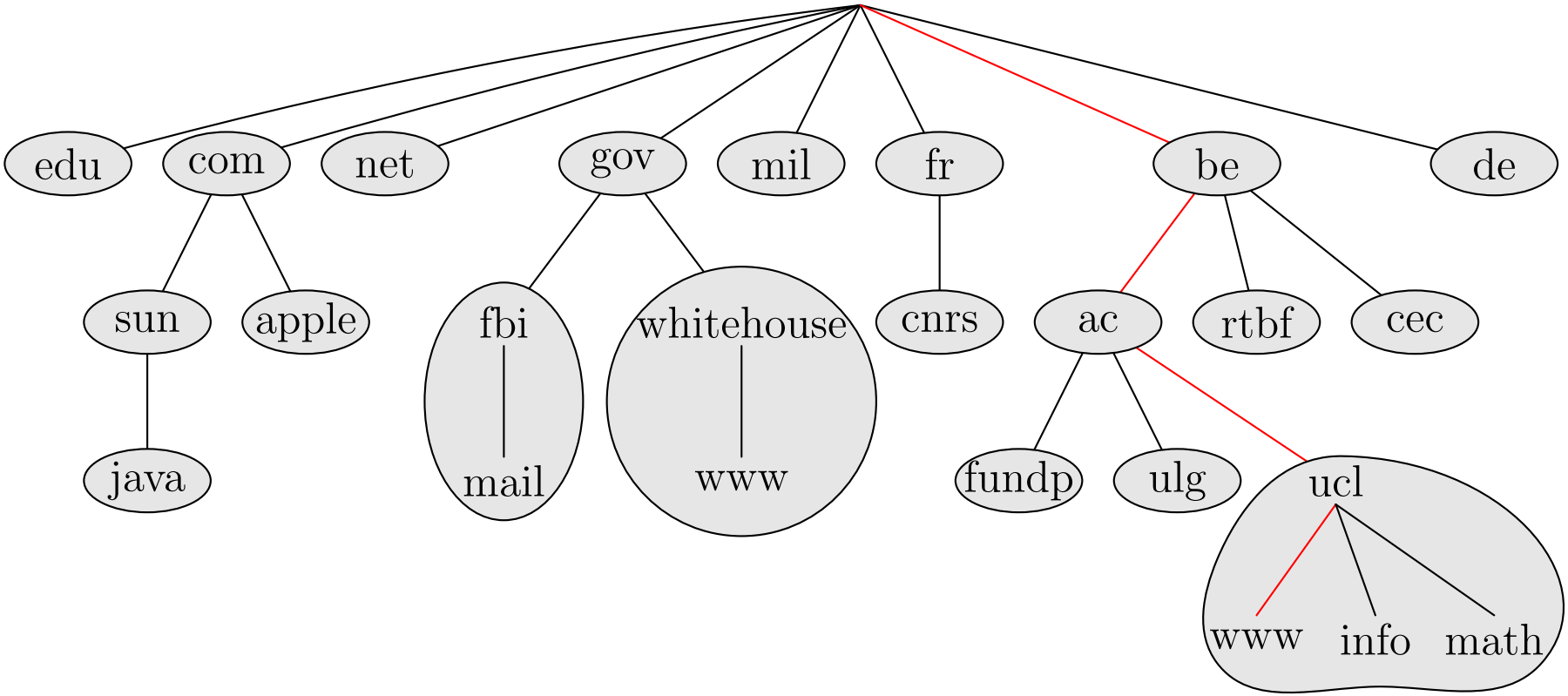


Adresses humaines

Le but du DNS est d'établir le lien entre adresses IP et noms d'hôtes ou de domaines.

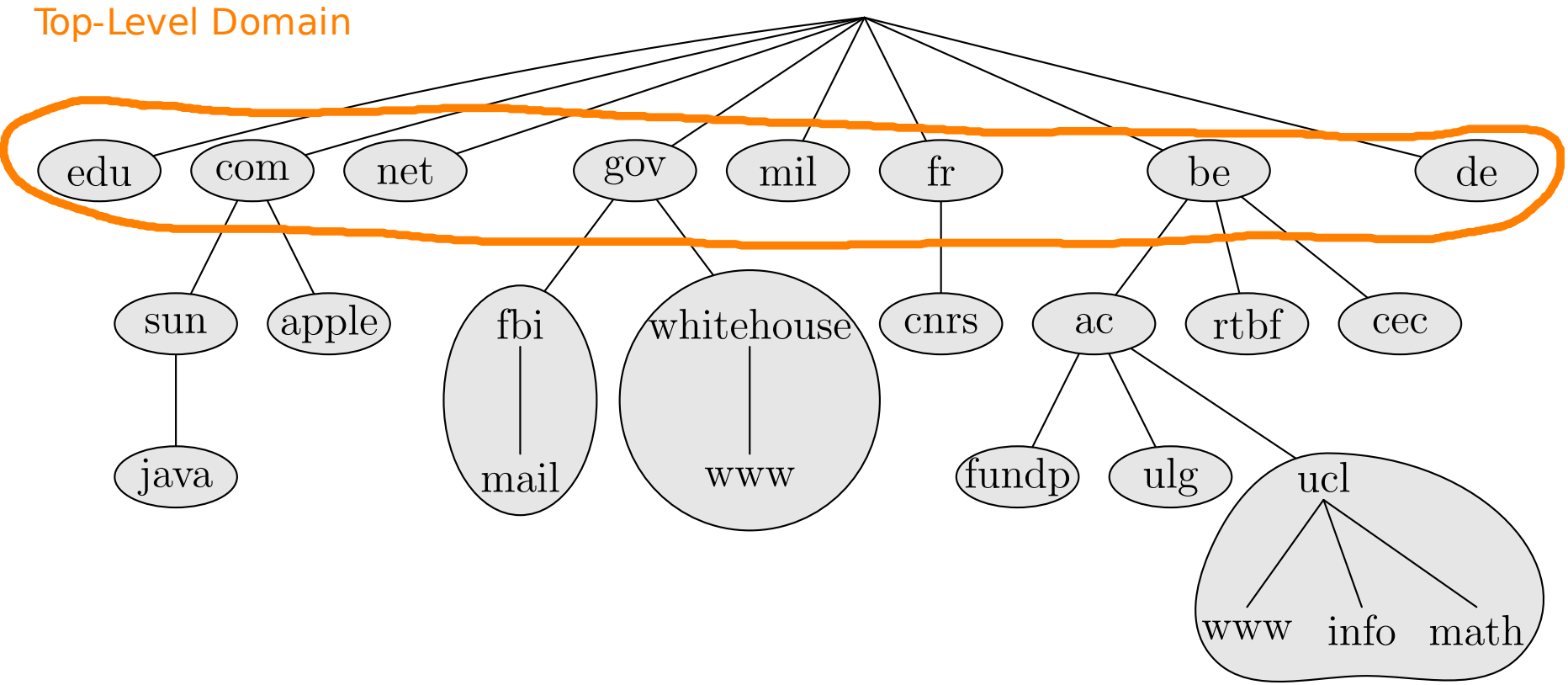
- C'est une base de données répartie entre des serveurs dédiés à cette tâche.
- Les noms sont organisés en un arbre dont chaque sous-arbre est un domaine.
- Chaque nœud a un nom, par exemple *ucl.ac.be*
- Chaque feuille correspond à un hôte (une machine), par exemple *www.ucl.ac.be*

Domain Name Service



www.ucl.ac.be

DNS et Top-Level Domain



DNS et Top-Level Domain

- gTLD generic Top-Level Domain (.com, .org, .net)
- ccTLD country code Top-Level Domain (.fr .de .us .be)
- sTLD sponsored Top-Level Domain (.gov .edu)
- nTLD new Top-Level Domain (.paris)

Internet Corporation for Assigned Names and Numbers

IP : Rappels du cours 4

Une adresse IP est de la forme x.y.z.t où x, y, z et t sont entre 0 et 255.

- Si on oublie les détails, une adresse de réseau contient un préfixe qui représente un domaine et un suffixe laissé à 0, ex : 194.167.30.0
- Une adresse d'hôte dans ce réseau spécifie le suffixe, ex : 194.167.30.130
- L'ensemble des adresses a une structure d'arbre (premier préfixe 0.0.0.0 comme racine) avec à ses nœuds les sous-réseaux et aux feuilles les hôtes.

DNS fait le lien entre les deux arborescences.

Principe

- L'ensemble du réseau est découpé en zones.
- Chaque nœud appartient exactement à une zone.
- Pour chaque zone, il existe un ou plusieurs serveurs DNS officiels ou faisant autorité.
- Ce serveur officiel gère tous les nœuds de la zone et connaît les serveurs faisant autorité sur les zones correspondant à des sous-domaines.
- Pour connaître une adresse IP, il faut suivre le chemin depuis un serveur racine qui fait autorité sur les *Top-level domains*.

Requête DNS

- On appelle résolution DNS le fait d'obtenir la correspondance entre un nom de domaine et une adresse IP.
- La résolution est initiée par une requête DNS adressée par un client à un solveur DNS : chaque FAI possède un solveur DNS et fournit cette adresse lors de la connexion du client.
  + Via la configuration DHCP
- Le solveur DNS obtient l'adresse et la fournit au client.
- Envoi de la requête en UDP sur le port 53 du solveur.

Résolutions DNS

Il existe 2 modes de résolution DNS :

- Requête récursive : Le serveur qui reçoit la requête fait la résolution et répond. Un serveur qui accepte les requêtes récursives est appelé *serveur récursif*. Ce sont les solveurs DNS des FAI en particulier.
- Requête itérative : Le serveur qui reçoit la requête répond soit avec l'information demandée s'il peut, soit avec un lien vers un serveur plus proche auquel il délègue la responsabilité. Ce sont les serveurs faisant autorité sur une zone en particulier.

Chaque serveur DNS connaît le nom et l'adresse IP des serveurs auxquels il délègue.

Résolution DNS

- Le serveur (S_1) qui a reçu la requête DNS répond directement s'il est responsable de l'adresse demandée.
- Sinon :
  - si c'est un serveur récursif, il interroge le serveur connu le plus proche.
  - si c'est un serveur itératif, il envoie en réponse un pointeur vers le serveur connu le plus proche.

Le serveur connu le plus proche étant :

→ le serveur faisant autorité sur ce domaine si l'adresse appartient à un sous domaine.

- Exemple : le client cherche `sousdom.dom.com`
- S_1 fait autorité sur `dom.com` mais pas sur `sousdom.dom.com`
- le serveur connu le plus proche est S_2, serveur faisant autorité pour `sousdom.dom.com`

→ sinon, un serveur racine.

Serveurs DNS

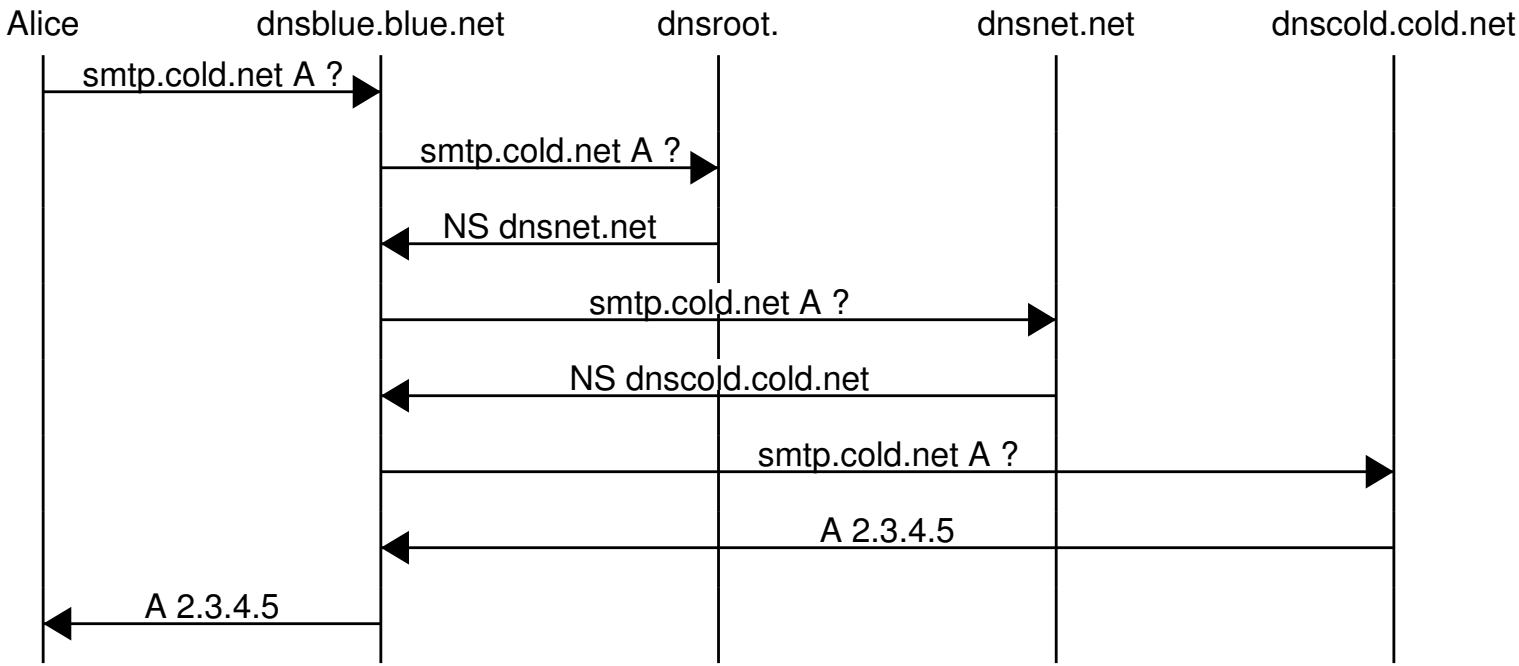
Différents types de serveurs DNS :

- Les serveurs racines qui connaissent l'adresse d'un serveur officiel pour chaque domaine de premier niveau (TLD : com, net, org...)
- Chaque serveur DNS doit connaître l'adresse d'un serveur racine pour débuter la résolution.
- Les serveurs de zones (TLD ou non) qui connaissent les adresses des hôtes de leur zone et les adresses d'un serveur officiel dans chaque sous-domaine.

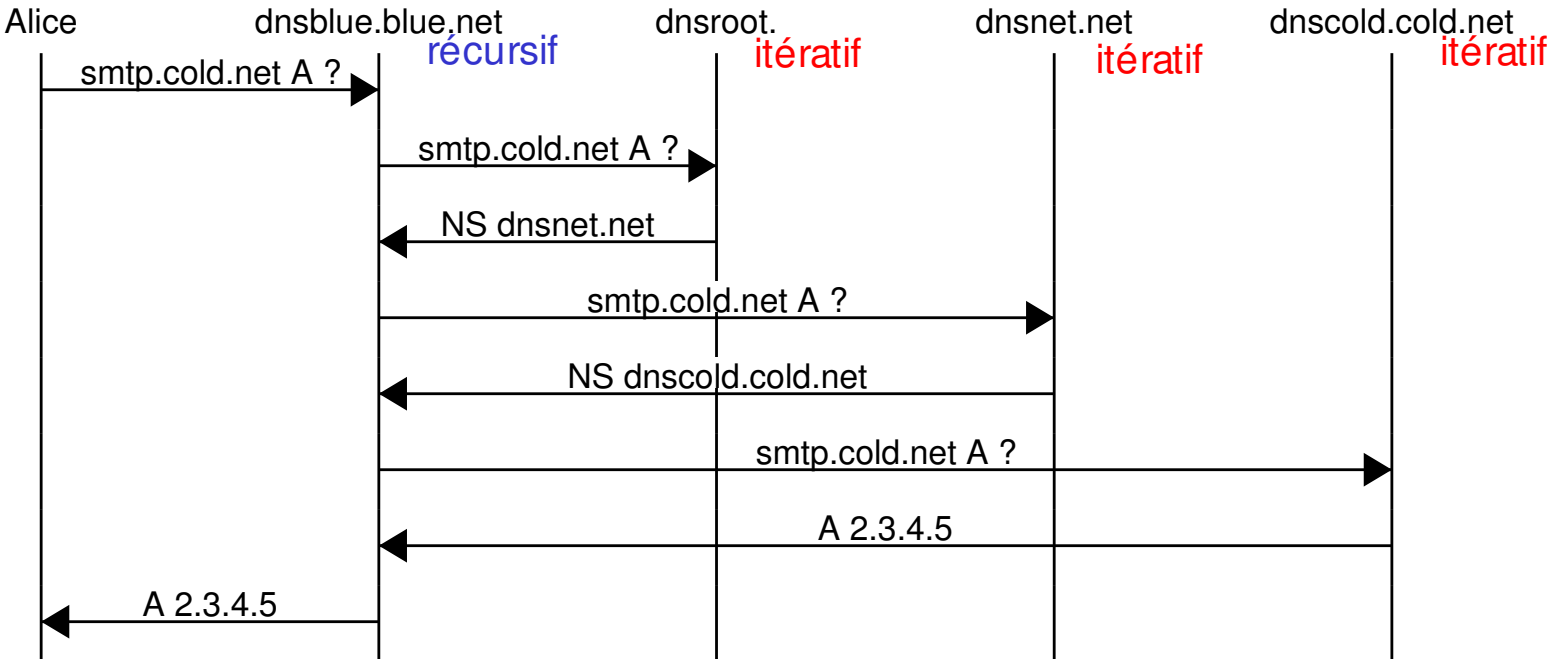
Résolution DNS

- Le client envoie une requête DNS récursive au serveur DNS **F** de son FAI.
- Ce serveur **F**, s'il ne connaît pas l'adresse demandée, envoie une requête itérative à un serveur racine **A** qui lui répond avec l'adresse d'un serveur DNS **B** auquel il délègue la responsabilité.
- Le serveur **F** poursuit en envoyant une requête itérative à **B**, etc...

Exemple


Alice
dnsblue.blue.net
dnsroot.
dnsnet.net
dnscold.cold.net
smtp.cold.net A ?
smtp.cold.net A ?
NS dnsnet.net
smtp.cold.net A ?
NS dnscold.cold.net
smtp.cold.net A ?
A 2.3.4.5
A 2.3.4.5

Exemple


Alice
dnsblue.blue.net
récursif
dnsroot.
itératif
dnsnet.net
itératif
dnscold.cold.net
itératif
smtp.cold.net A ?
smtp.cold.net A ?
NS dnsnet.net
smtp.cold.net A ?
NS dnscold.cold.net
smtp.cold.net A ?
A 2.3.4.5
A 2.3.4.5

Robustesse et temps d'exécution

Pour optimiser le fonctionnement du service :

- L'information est dédoublée : chaque nom de domaine est connu par au moins deux serveurs.
  primary name et secondary name servers.
- Chaque réponse reçue par un serveur est conservée en cache.
  Time To Live (TTL) fixé par le serveur faisant autorité.

Resource Records

Un enregistrement DNS (dans le fichier de configuration d'un serveur DNS) contient :

Nom	TTL	Classe	Type	Valeur

- Nom : Nom de domaine concerné
- TTL : Time To Live de l'enregistrement (vide si valeur par défaut)
- Classe : En général IN pour internet
- Type : Type d'enregistrement
- Valeur : Selon le type

Types d'enregistrement

- A : Adresse IPv4
- AAAA : Adresse IPv6
- MX : Mail eXchanger (avec une priorité)
- TXT : Commentaires
- NS : Name Server (nom du serveur à contacter)
- SOA : Start Of Authority (informations sur la zone DNS)
- CNAME : Canonical NAME (pour un alias)

db.exemple.com (1/3)

Pour une zone exemple.com.

```
$TTL 86400   ; Durée de vie par défaut des enregistrements (

 ; Déclaration de l'autorité sur la zone
@   IN  SOA  ns1.exemple.com. admin.exemple.com. (
          2024022102   ; Numéro de série
          7200         ; Refresh
          3600         ; Retry
          1209600      ; Expire
          86400        ; Negative cache TTL
)

 ; Serveurs DNS pour la zone
@   IN  NS   ns1.exemple.com.
@   IN  NS   ns2.exemple.com.
```

db.exemple.com (2/3)

Pour une zone exemple.com.

```
 ; Serveurs DNS pour la zone
@    IN  NS    ns1.exemple.com.
@    IN  NS    ns2.exemple.com.

 ; Adresses des serveurs DNS
ns1 IN   A     192.168.1.1
ns2 IN   A     192.168.1.2

 ; Enregistrements A (adresses IP pour des services)
www     IN   A      192.168.1.100   ; Site web principal
 ; Pour le serveur mail (adresse @exemple.com)
mail    IN   A      192.168.1.200
@       IN   MX     10 mail.exemple.com.
```

db.exemple.com

Pour une zone exemple.com.

```
 ; Enregistrement CNAME (alias)
blog    IN   CNAME www.exemple.com.

 ; === machines (sous domaines sans délégation) ===
dev     IN   A      192.168.1.150   ; machine de dev
test    IN   A      192.168.1.151   ; Serveur de test

 ; === Délégation de sous-domaine ===
sub     IN   NS     ns1.sub.exemple.com.  ;à un autre serveur
sub     IN   NS     ns2.sub.exemple.com.
ns1.sub   IN   A    203.0.113.1
ns2.sub   IN   A    203.0.113.2
```

Reverse DNS

DNS offre la possibilité de trouver le nom de domaine associé à une adresse IP :

- Création d'un domaine approprié : in-addr.arpa
- Transformation de l'adresse IP en nom de domaine dans in-addr.arpa :

  168.152.13.27 devient 27.13.152.168.in-addr.arpa

- Type supplémentaire PTR

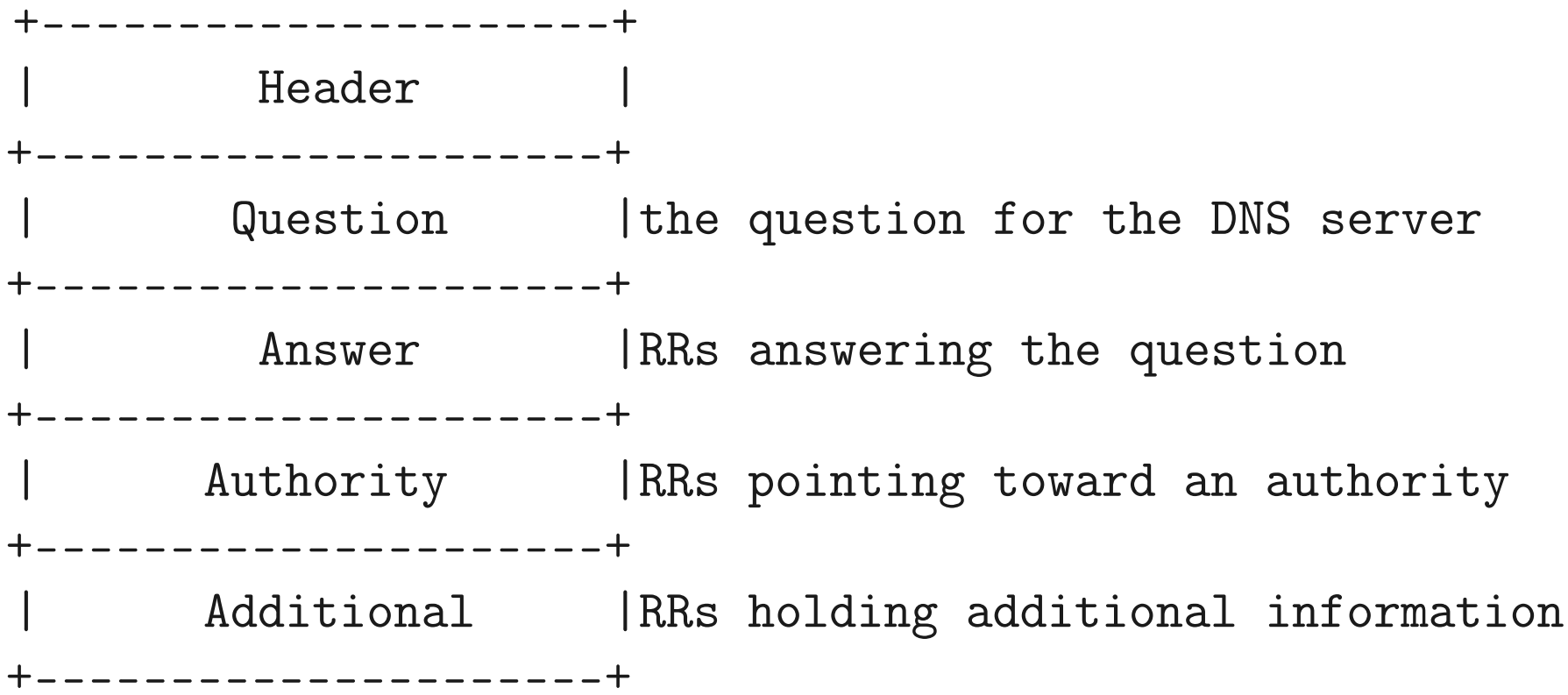
```
27.13.152.168.in-addr.arpa. IN PTR mail.exemple.com.
```

Reverse DNS : objectif

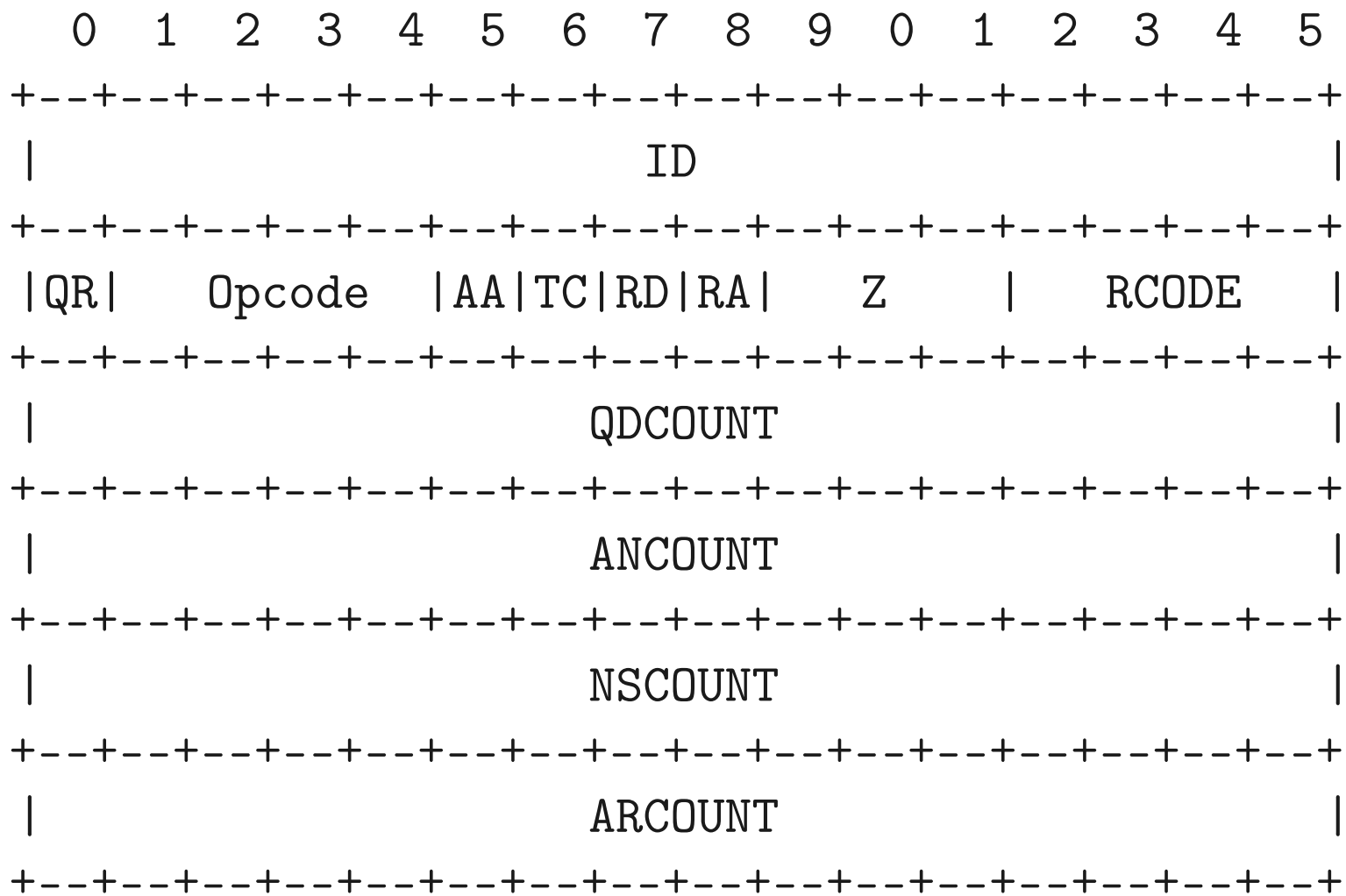
Le but principal est de permettre une vérification de certaines informations DNS :

- confirmer une réponse DNS avec une requête reverse DNS (manque de sécurité du protocole).
- lutter contre le SPAM en comparant :
  - le nom de domaine associé à l'adresse IP de l'émetteur ;
  - le nom de domaine de l'adresse mail annoncée comme expéditeur.

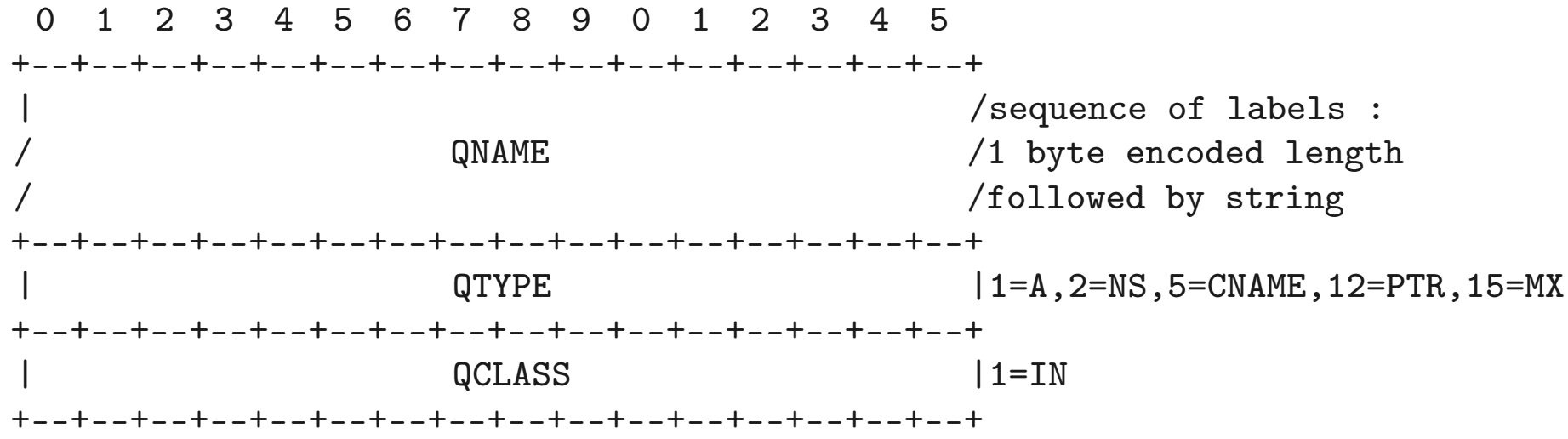
Les trames DNS : le format général



Les trames DNS : Header



Les trames DNS : Question

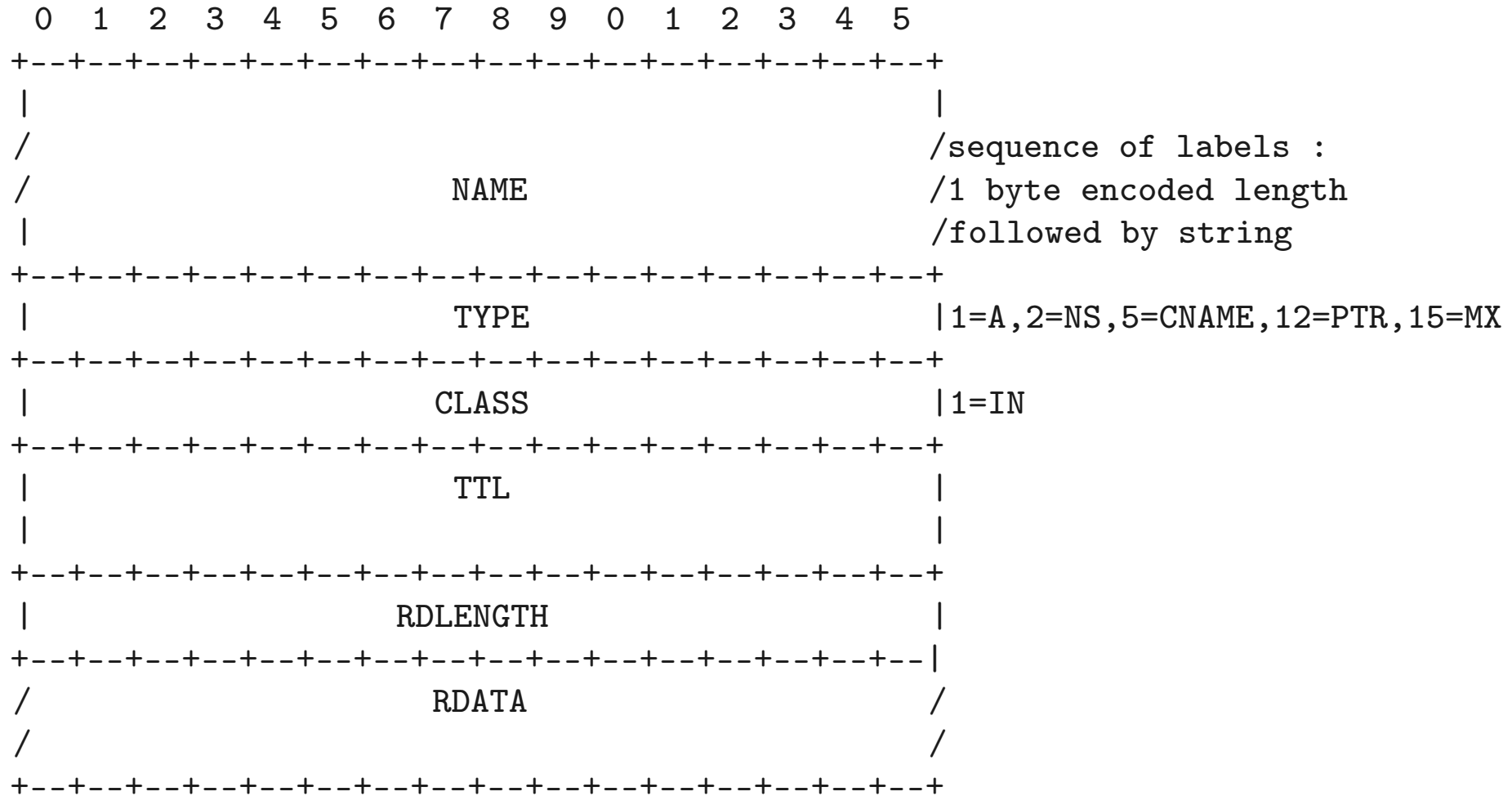


Exemple pour le QNAME

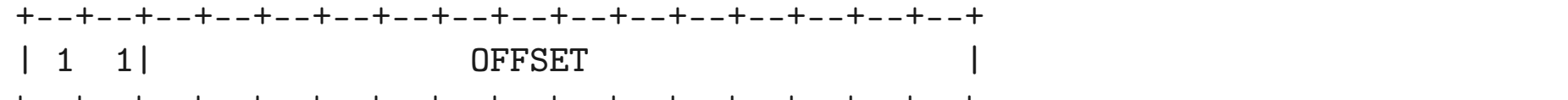
03	77
77	77
07	65
78	65
6D	70
6C	65
03	63
6F	6D
00	

03	w
w	w
07	e
x	e
m	p
l	e
03	c
o	m
00	

Les trames DNS : Réponse RR



- NAME/QNAME peut être remplacé partiellement par un pointeur :



Simple Mail Transfer Protocol

Les principes

- Envoi de messages désynchronisé.
- Diversité de formats, ASCII à l'origine, généralisation à des binaires (son, image, formats divers) avec MIME (Multipurpose Internet Mail Extensions).
- Utilisation du DNS pour trouver le serveur de destination.

Le chemin

- Mail User Agent : Client mail (alpine, Thunderbird,. . .), webmail
- Mail Submission Agent : réception/vérification du mail client
- Mail Transfer Agent : Transfert vers un autre serveur
- Mail Delivery Agent : Dépôt dans une boîte mail

Deux types de protocole :

- Pour envoyer un message SMTP, TCP port 25, RFC 821
- Pour récupérer les messages de la boîte mail POP, IMAP

MSA & MTA

On dissocie les rôles de Mail Submission Agent et MTA. Les deux reçoivent des données lors d'une session SMTP, mais :

- le MSA reçoit les mails des MUA (soumission, port 587) ;
- le MTA assure le relais des mails entre serveurs (SMTP, port 25).

Dans ce cas, MSA et MTA collaborent côté émetteur, symétriquement à MTA et MDA côté receveur.

Une suite de serveurs SMTP

MUA

↓

MSA/MTA(émetteur) SMTP

↓

...

↓

MTAi SMTP

↓

...

↓

MTA(dest) SMTP

↓

MDA POP/IMAP

↓

MUA

Le contenu

Plusieurs entêtes puis le corps après une ligne vide. Entêtes fortement recommandées :

- To : xxxx.yyyy@gmail.com
- From :yyyy.zzzz@univ-orleans.fr
- Date : Tue, 24 Feb 2025 15 :21 :45 +0200

D'autres moins :

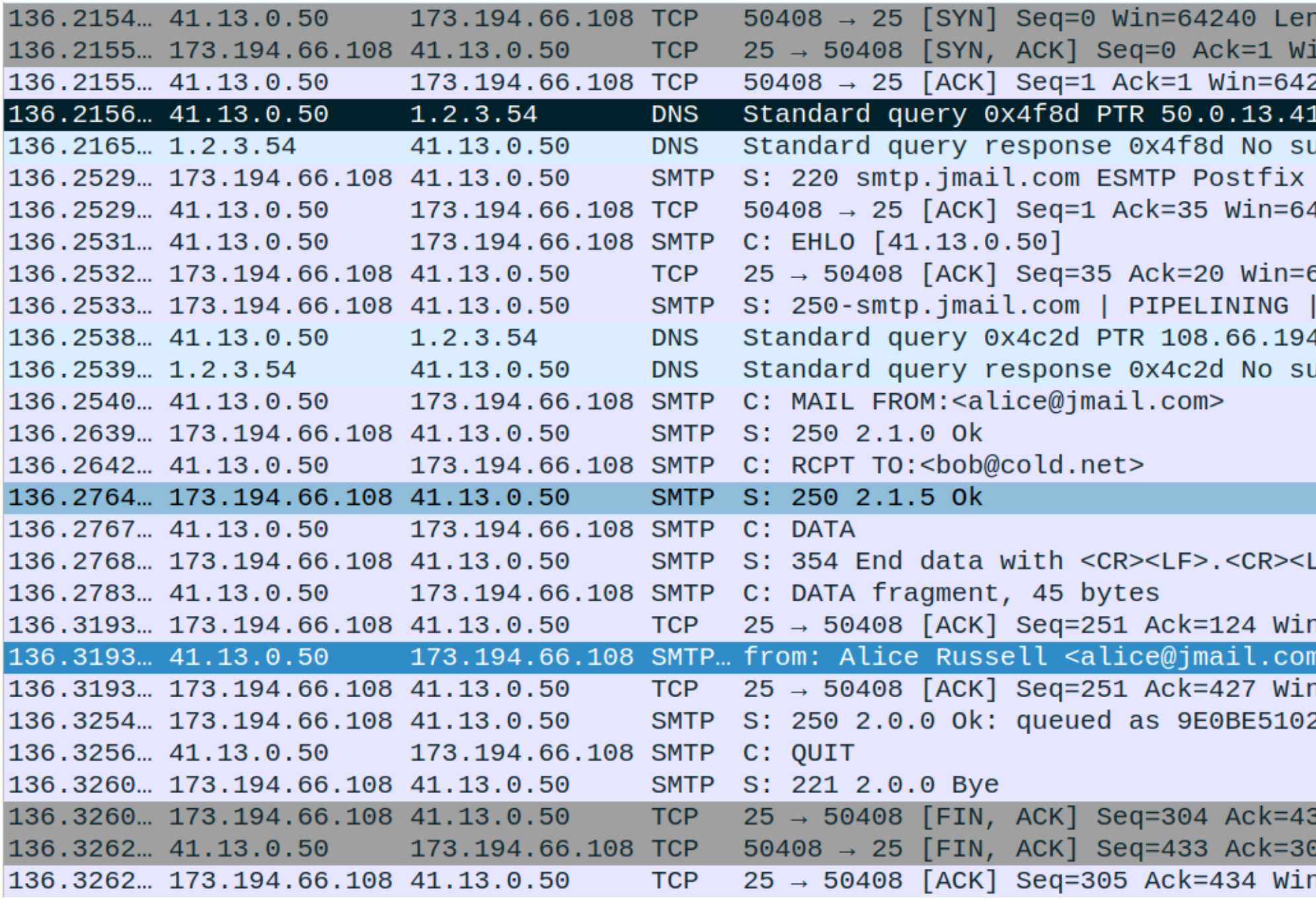
- Subject : Hola
- cc : yyyy.zzzz@univ-lorraine.fr
- MIME-Version : 1.0
- Content-Type : ext/plain ; charset=utf-8 ;

"Return-Path: <l3inge.sciences-info-owner@listes.univ-orleans.fr>
Received: from zmtain02.partage.renater.fr (LHLO
 zmtain02.partage.renater.fr) (194.254.241.20) by
 zstore-b1-042.partage.renater.fr with LMTP; Tue, 25 Feb 2025 11:20:53 +0100
 (CET)
Received: from mxb2-1.relay.renater.fr (unknown [194.254.241.249])
	by zmtain02.partage.renater.fr (Postfix) with ESMTPS id 09F09140ED2
	for <sophie.robert@univ-orleans.fr>; Tue, 25 Feb 2025 11:20:53 +0100 (CET)
Received: from mxb2-1.relay.renater.fr (localhost [127.0.0.1])
	(using TLSv1.2 with cipher ECDHE-RSA-AES256-GCM-SHA384 (256/256 bits)
	(No client certificate requested)
	by mxb2-1.relay.renater.fr (asm) with ESMTPS id ECF53603EB
	for <sophie.robert+split@univ-orleans.fr>; Tue, 25 Feb 2025 11:20:52 +0100 (CE
Received: from sara.univ-orleans.fr (sara.univ-orleans.fr [194.167.30.84])
	by mxb2-1.relay.renater.fr (asm) with ESMTP id 2551160115;
	Tue, 25 Feb 2025 11:20:00 +0100 (CET)
Received: from sara.univ-orleans.fr (localhost [127.0.0.1])
	by sara.univ-orleans.fr (Postfix) with ESMTP id 87AD31819B;
	Tue, 25 Feb 2025 11:19:05 +0100 (CET)
Received: from aurora.univ-orleans.fr (aurora.univ-orleans.fr [194.167.30.197])
	by sara.univ-orleans.fr (Postfix) with ESMTP id 2F47F1817A;
	Tue, 25 Feb 2025 11:19:05 +0100 (CET)
X-Original-To: l3inge.sciences-info@listes.univ-orleans.fr
Delivered-To: listes.univ-orleans.fr-l3inge.sciences-info@aurora.univ-orleans.fr

Le protocole

- Protocole textuel, facile à comprendre (alice@jmail.com écrit à bob@cold.net).
- Le MUA d'Alice transmet le message au serveur de courrier sortant de jmail.com, au moyen d'une session SMTP.
- Le serveur sortant résout l'enregistrement DNS de type MX pour cold.net.
- Il transmet le message au moyen d'une nouvelle session SMTP.
- Le serveur entrant de cold.net stocke le message dans la boîte mail de Bob.

Pour chaque session SMTP, le client demande une connexion TCP avec le serveur, celui-ci lance alors l'échange.



136.2154…	41.13.0.50	173.194.66.108	TCP	50408 → 25 [SYN] Seq=0 Win=64240 Len
136.2155…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [SYN, ACK] Seq=0 Ack=1 Wi
136.2155…	41.13.0.50	173.194.66.108	TCP	50408 → 25 [ACK] Seq=1 Ack=1 Win=642
136.2156…	41.13.0.50	1.2.3.54	DNS	Standard query 0x4f8d PTR 50.0.13.41
136.2165…	1.2.3.54	41.13.0.50	DNS	Standard query response 0x4f8d No su
136.2529…	173.194.66.108	41.13.0.50	SMTP	S: 220 smtp.jmail.com ESMTP Postfix
136.2529…	41.13.0.50	173.194.66.108	TCP	50408 → 25 [ACK] Seq=1 Ack=35 Win=64
136.2531…	41.13.0.50	173.194.66.108	SMTP	C: EHLO [41.13.0.50]
136.2532…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [ACK] Seq=35 Ack=20 Win=6
136.2533…	173.194.66.108	41.13.0.50	SMTP	S: 250-smtp.jmail.com | PIPELINING |
136.2538…	41.13.0.50	1.2.3.54	DNS	Standard query 0x4c2d PTR 108.66.194
136.2539…	1.2.3.54	41.13.0.50	DNS	Standard query response 0x4c2d No su
136.2540…	41.13.0.50	173.194.66.108	SMTP	C: MAIL FROM:<alice@jmail.com>
136.2639…	173.194.66.108	41.13.0.50	SMTP	S: 250 2.1.0 Ok
136.2642…	41.13.0.50	173.194.66.108	SMTP	C: RCPT TO:<bob@cold.net>
136.2764…	173.194.66.108	41.13.0.50	SMTP	S: 250 2.1.5 Ok
136.2767…	41.13.0.50	173.194.66.108	SMTP	C: DATA
136.2768…	173.194.66.108	41.13.0.50	SMTP	S: 354 End data with <CR><LF>.<CR><L
136.2783…	41.13.0.50	173.194.66.108	SMTP	C: DATA fragment, 45 bytes
136.3193…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [ACK] Seq=251 Ack=124 Win
136.3193…	41.13.0.50	173.194.66.108	SMTP…	from: Alice Russell <alice@jmail.com
136.3193…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [ACK] Seq=251 Ack=427 Win
136.3254…	173.194.66.108	41.13.0.50	SMTP	S: 250 2.0.0 Ok: queued as 9E0BE5102
136.3256…	41.13.0.50	173.194.66.108	SMTP	C: QUIT
136.3260…	173.194.66.108	41.13.0.50	SMTP	S: 221 2.0.0 Bye
136.3260…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [FIN, ACK] Seq=304 Ack=43
136.3262…	41.13.0.50	173.194.66.108	TCP	50408 → 25 [FIN, ACK] Seq=433 Ack=30
136.3262…	173.194.66.108	41.13.0.50	TCP	25 → 50408 [ACK] Seq=305 Ack=434 Win

```
220 smtp.jmail.com ESMTP Postfix
EHLO [41.13.0.50]
250-smtp.jmail.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
MAIL FROM:<alice@jmail.com>
250 2.1.0 Ok
RCPT TO:<bob@cold.net>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Date: Tue, 25 Feb 2025 15:16:01 +0000 (UTC)
From: Alice Russell <alice@jmail.com>
X-X-Sender: root@alice.blue.net
To: bob@cold.net
Message-ID: <alpine.DEB.2.21.2502251515410.50@alice.blue.net>
User-Agent: Alpine 2.21 (DEB 202 2017-01-01)
MIME-Version: 1.0
Content-Type: text/plain; format=flowed; charset=US-ASCII

Hello je suis Alice
.
250 2.0.0 Ok: queued as 9E0BE51023B3
QUIT
221 2.0.0 Bye
```

HyperText Transfer Protocol (HTTP)

Principe

Récupérer des documents hypertexte stockés sur un serveur web :

- Client : navigateur.
- Résolution du nom de domaine passé dans l'url : champ DNS de type A.
- Un format de fichier : HTML.
- Un protocole d'échanges : HTTP

Uniform Resource Locator

(RFC 3986) Identifiant unique d'une ressource contenant
le moyen d'y accéder, l'hôte et le chemin sur l'hôte :

`http://www.perdu.com`

`https://celene.univ-orleans.fr/course/view.php?id=1980`

`http://cnp3book.info.ucl.ac.be/1st/html/application/http.html`

`file:///etc/apt/sources.list`

`ftp://ftp.lip6.fr/pub/rfc/rfc/rfc1149.txt.gz`

HyperText Markup Language

```
<html>
  <head>
    <title>Vous Etes Perdu  ?</title>
  </head>
  <body>
    <h1>Perdu sur l'Internet  ?</h1>
    <h2>Pas de panique, on va vous aider</h2>
    <strong>
      <pre>    * <----- vous &ecirc ;tes ici</pre>
    </strong>
  </body>
</html>
```

HyperText Transfer Protocol

- protocole textuel, TCP, port 80.
- Actuellement HTTP 1.1, dans les RFC 7230 à 7237.
  Et HTTP 2 (RFC 7540).
-
  - Commande GET suivi du chemin du document et du protocole
  - puis Host: sur une nouvelle ligne
  - puis une ligne vide.

nc -C www.perdu.com 80

GET / HTTP/1.1
Host: www.perdu.com

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 16:01:18 GMT
Server: Apache
Last-Modified: Thu, 02 Jun 2016 06:01:08 GMT
ETag: "cc-5344555136fe9"
Accept-Ranges: bytes
Content-Length: 204
Vary: Accept-Encoding
Content-Type: text/html
<html><head><title>Vous Etes Perdu
?</title></head><body><h1>Perdu sur l'Internet
?</h1><h2>Pas de panique, on va vous
aider</h2><pre> * <—— vous êtes
ici</pre></body></html>

Droits

Merci à l'équipe historique Réseaux en L3 Nicolas Ollinger, Martin Delacourt et Mathieu Liedloff.