Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

Master 1ère année

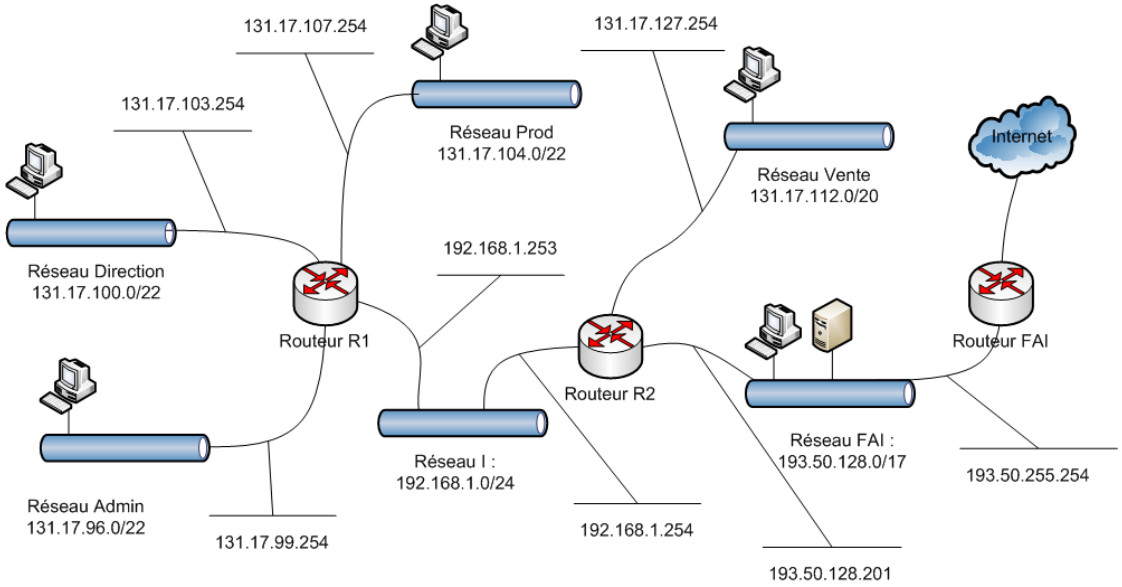
Réseaux Avancés II

Corrections — mai 2012

Durée : 2h — Documents autorisés (sauf sur support électronique)

IPv6 – (9 points)

1 – 4pts Soit le réseau d'une entreprise `131.17.96.0/19` décrit par le schéma suivant :



Cette entreprise vient d'obtenir auprès du RIPE l'adresse IPv6 suivante : `2001:660:6201::/48`.
Le responsable du système d'information vous demande de réaliser une migration du réseau vers IPv6.
Il vous fournit la description suivante :

- ⋆ il existe deux bâtiments distincts dans l'entreprise : Bâtiment_A et Bâtiment_B ;
- ⋆ les différents réseaux se répartissent :

« géographiquement » de la façon suivante :

Network	Location
Direction	Bâtiment_A
Vente	Bâtiment_B
Prod	Bâtiment_A
Admin	Bâtiment_A
Réseau I	Bâtiment_A

« par usage » de la façon suivante :

Type
Visiteur
Chargé de clientèle
Ouvrier
Direction
Admin
Infrastructure

a. Décomposez en nombre de bits, l'@IPv6 fournie, en tenant compte des « Types » & « Location ».
D'après les informations fournies, on dispose de 2 « locations » et de 6 « types ».
En utilisant la méthode proposée par le RIPE NCC et en arrondissant les besoins en multiple de 4 bits (pour un digit hexadécimal) afin de faciliter la manipulation :
- ◇ *2 locations* $\Longrightarrow$ *1 bit* ($2^1 = 2$)*, soient 4 bits (ce qui permet de prévoir les évolutions futures) ;*
- ◇ *6 types* $\Longrightarrow$ *3 bits* ($2^3 = 8$)*, soient 4 bits.*

b. De combien de sous-réseaux pourra-t-on bénéficier après cette décomposition et pour chaque combinaison « type/location » ?
Le préfixe réseau initial est en `/48`*, ce qui laisse 16 bits que l'on peut utiliser pour notre plan d'adressage.*
Il y a 4 bits pour les « locations » et 4 bits pour les « types », soient 8 bits.

Il reste donc 8 bits pour les bits « assignables », soit la possibilité de créer $2^8 = 256$ sous-réseaux pour chaque combinaison.

c. Donnez un plan d'adressage en IPv6 pour ce réseau, en favorisant la localisation.
Pour rappel, la réalisation de « subnetting » intervient pour les obligations suivantes :
- ◇ *contraintes de localisation : décomposer en réseaux différents des emplacements géographiques disjoints (physiquement, il n'est pas possible de relier toutes les machines à un même réseau local) ;*
- ◇ *contraintes d'usage : isoler des réseaux utilisés pour des usages différents.*

Dans le synoptique réseau en IPv4, localisation et usage sont mélangés du fait du faible nombre de bits disponibles pour réaliser ce « subnetting » ; l'utilisation d'IPv6 permet de rendre cette répartition explicite, comme le propose la méthodologie proposée par le RIPE NCC.
En favorisant la localisation, c-à-d en mettant les bits concernant la « location » au début afin de faciliter le travail du routeur (suivant un `/56` *le routeur peut acheminer les paquets vers le bâtiment A ou B), on obtient la répartition suivante (chaque symbole « L », « G » ou « B » désigne un chiffre hexadécimal) :*

`2001:660:6201:`	`L`	`G`	`B`	`B`	`::/64`

On va numéroter nos « Locations » et « Types », puis faire « migrer » les réseaux :

Location	Valeur
Bâtiment$_A$	1
Bâtiment$_B$	2

Type	Valeur
Visiteur	1
Chargé de clientèle	2
Ouvrier	3
Direction	4
Admin	5
Infrastructure	6

Type	Appellation originale
Visiteur	*aucune*
Chargé de clientèle	Vente
Ouvrier	Prod
Direction	Direction
Admin	Admin
Infrastructure	Réseau I

On aura pas besoin des bits « B », assignables, pour l'instant, du fait du nombre réduit de réseaux, c-à-d 5 déjà utilisés :

Location	Type	Assignable	Network
Bâtiment$_A$ (1)	Ouvrier (3)	0	`2001:660:6201:1300::/64`
Bâtiment$_A$ (1)	Admin (5)	0	`2001:660:6201:1500::/64`
Bâtiment$_A$ (1)	Direction (4)	0	`2001:660:6201:1400::/64`
Bâtiment$_A$ (1)	Infrastructure (6)	0	`2001:660:6201:1600::/64`
Bâtiment$_B$ (2)	Chargé de clientèle (2)	0	`2001:660:6201:2200::/64`

d. Comment les machines connectées dans les différents réseaux vont-elles apprendre le préfixe réseau auquel elles appartiennent ?
Par :
- ◇ *diffusion de ce préfixe depuis le routeur (« router advertisement ») ;*
- ◇ *par configuration manuelle du poste ;*
- ◇ *par DHCPv6.*

e. Est-ce que pendant la période de configuration matérielle de ce plan d'adressage (où il n'y aura plus de routage), les machines du réseau « Prod » pourront-elles continuer à communiquer entre elles et pourquoi ?
Oui, en utilisant des adresses IPv6 « link-local », obtenues par auto-configuration.

2 – 2pts Soit l'@MAC « `da:ec:6d:59:96:2e` » d'une interface réseau :

a. Donnez son @IPv6 de lien obtenue par auto-configuration ;
Ce qui donne : `fe80::d8ec:6dff:fe59:962a`.

b. Par quelle @IPv6 multicast de type « *solicitation de voisin* », cette machine pourra-t-elle être jointe (protocole « Neighbor Discovery ») ?
L'adresse de multicast pour la « solicitation de voisin » est : `ff02::1:ff59:962e`.

c. Si le préfixe global du réseau auquel appartient cette machine est le `2001:660:6201::/48` quelle sera son adresse globale ?
`2001:660:6201::d8ec:6dff:fe59:962a`

d. Lors de l'envoi d'une trame d'ethertype `0x86DD` en unicast vers cette interface, qu'elle sera l'@MAC destination de cette trame ?
L'@MAC fournie au début : `da:ec:6d:59:96:2e`.

3 – 3pts Analysez la trame suivante :

```
0000   00 10 75 1A D4 8A 00 26  BB 15 8E 87 86 DD 60 00   ..u....&......‘.
0010   00 00 00 1C 2C 40 FE 80  00 00 00 00 00 00 02 26   ....,@.........&
0020   BB FF FE 15 8E 87 FE 80  00 00 00 00 00 00 02 10   ................
0030   75 FF FE 1A D4 8A 06 00  00 00 00 00 00 00 00 50   u..............P
0040   13 89 00 00 04 E8 00 01  7F 29 50 12 20 00 65 6D   .........)P. .em
0050   00 00                                             ..
```

L'analyse donne (version condensée Scapy) :

```
<Ether  dst=00:10:75:1a:d4:8a src=00:26:bb:15:8e:87 type=0x86dd |
<IPv6  nh=Fragment Header hlim=64 src=fe80::226:bbff:fe15:8e87 dst=fe80::210:75ff:fe1a:d48a |
<IPv6ExtHdrFragment  nh=TCP |
<TCP  sport=http dport=5001 seq=1256 ack=98089 flags=SA |>>>>
```

Soit un datagramme IP, contenu dans un fragment, provenant a priori d'un serveur HTTP (port 80) en réponse d'une demande de connexion (TCP flags = 'SA') entre machines dans le même réseau local (@IPv6 en fe80::/10).

Tunnels – (2 points)

4 – 2pts Questions :

a. Quelles différences entre un tunnel SIT, « Simple Internet Transition », et un tunnel GRE, « Generic Routing Encapsulation » ?
Le tunnel GRE peut transporter tous types de protocole alors que SIT ne transporte que des datagrammes IPv6.

b. Quelles différences entre un tunnel GRE et la technologie MPLS ?
Un tunnel GRE permet d'encapsuler des datagrammes IP dans un datagramme IPv4, il ne propose ni de sécurité ni d'amélioration des performances pour les données transmises. MPLS est une technologie permettant de « contrôler » le réseau d'interconnexion : grâce à ce contrôle on obtient de meilleures performances (commutation/« switching » au lieu de routage/« relaying ») et une certaine sécurité dans la mesure où les datagrammes ne circulent qu'entre des matériels autorisés.

Routage dynamique – (2 points)

5 – 2pts Questions :

a. En étudiant la table de routage d'un routeur, peux-t-on savoir si elle a été construite à l'aide du protocole RIP ou OSPF ? Pourquoi ?
Non, les tables de routages contiendront les mêmes informations, il ne sera pas possible de savoir comment elles ont été obtenues.

b. Est-il possible de reconstruire la topologie complète du réseau grâce à **un seul arbre** obtenu par l'algorithme de Dijkstra à partir d'un des routeurs du réseau (dans un protocole de routage par « états de lien ») ? Pourquoi ?
Non ce n'est pas possible : l'arbre ne contient que les liens permettant de construire des chemins de coût minimal et donc pas nécessairement tous les liens existants.

Programmation Python avec Scapy – (7 points)

6 – 3pts Le « ping » étant interdit par une entreprise, celle-ci vous demande de simuler son fonctionnement à l'aide du protocole UDP (qui lui est *bizarrement* autorisé) et qui, à chaque réception d'un paquet UDP sur le port `55555` :

- ⋆ vérifie qu'il contient le texte « `PING UDP :` » + un « numéro d'ordre » ;
- ⋆ renvoie à l'émetteur un datagramme IP :
  - ◇ de même « id », c-à-d identifiant de datagramme ;
  - ◇ contenant un datagramme UDP :
    - ⋆ contenant le texte « `PING UDP :` » + le même « numéro d'ordre » que celui reçu.

a. Est-il nécessaire d'utiliser Scapy et pourquoi ?
Il est nécessaire d'utiliser Scapy, car il faut pouvoir « forger » un datagramme IP en choisissant son ID (comme prévu dans le protocole).

b. Écrivez le programme en Python mettant en œuvre ce serveur (pour information, l'@IP du serveur où tournera votre programme est `192.168.10.10`).

```
#!/usr/bin/python

from scapy.all import *

contenu_ping = 'PING UDP : '
adresse_serveur = '192.168.10.10'
def traiter_paquet(p):
  contenu = str(p[UDP].payload)
  if (contenu[:len(contenu_ping)] == contenu_ping):
    paquet_reponse = p[IP]
    (paquet_reponse[IP].src, paquet_reponse[IP].dst) = (paquet_reponse[IP].dst, ¶
                paquet_reponse[IP].src)
    del(paquet_reponse[IP].chksum)
    del(paquet_reponse[UDP].chksum)
    send(paquet_reponse)

sniff(filter="(dst host %s) and (udp port 55555)"%adresse_serveur, prn=traiter_paquet)
```

On fait la sélection du bon datagramme à traiter dans le filtre de la commande Scapy « `sniff` ».

7 – 4pts **Réalisation d'un serveur de « *Port Knocking* »**. Vous écrirez le programme Python utilisant Scapy et réalisant :

- ⋆ la détection de cette séquence de tentative de connexion correspondant à du « port knocking » ;
- ⋆ l'autorisation de la connexion vers l'application (le travail du « PK daemon ») :
  - ○ la séquence de tentative de connexion TCP est la suivante : `[2027, 3230, 2001, 17377]` ;
  - ○ l'application attend en TCP sur le port `16400` ;
  - ○ le serveur hébergeant l'application et votre programme possède l'@IP `192.168.1.137`.

```
#!/usr/bin/python

from scapy.all import *
import commands

sequence = [2027, 3230, 2001, 17377]
adresse_serveur = '192.168.1.137'
port_application = 16400
rang_sequence = 0

def traiter_paquet(p):
  global rang_sequence
  if (p.sprintf('%TCP.flags%') == 'S'):
    if (p[TCP].dport == sequence[rang_sequence]):
      rang_sequence += 1
      print "Knock...",rang_sequence
      if (rang_sequence == len(sequence)):
 rang_sequence = 0
 commands.getoutput('sudo iptables -A INPUT -p tcp --dport %d -m state --state NEW,
ESTABLISHED -j ACCEPT'%port_application)
sniff(filter="(dst host %s) and tcp"%adresse_serveur, prn=traiter_paquet)
```