Amérique du Nord – 2026 – sujet1

Exercice 2 (6 points)

Cet exercice porte sur l'architecture matérielle (réseau), les structures de données et la programmation orientée objet.

Voici un schéma du réseau de l'entreprise Gamerzz, qui propose à ses clients :

- une salle pour jouer à des jeux vidéos en ligne (Salle Gaming Online) ;
- une autre pour jouer en réalité virtuelle (Salle Gaming VR) ;
- un site pour réserver, organiser des évènements et gérer les classements des joueurs en réalité virtuelle.

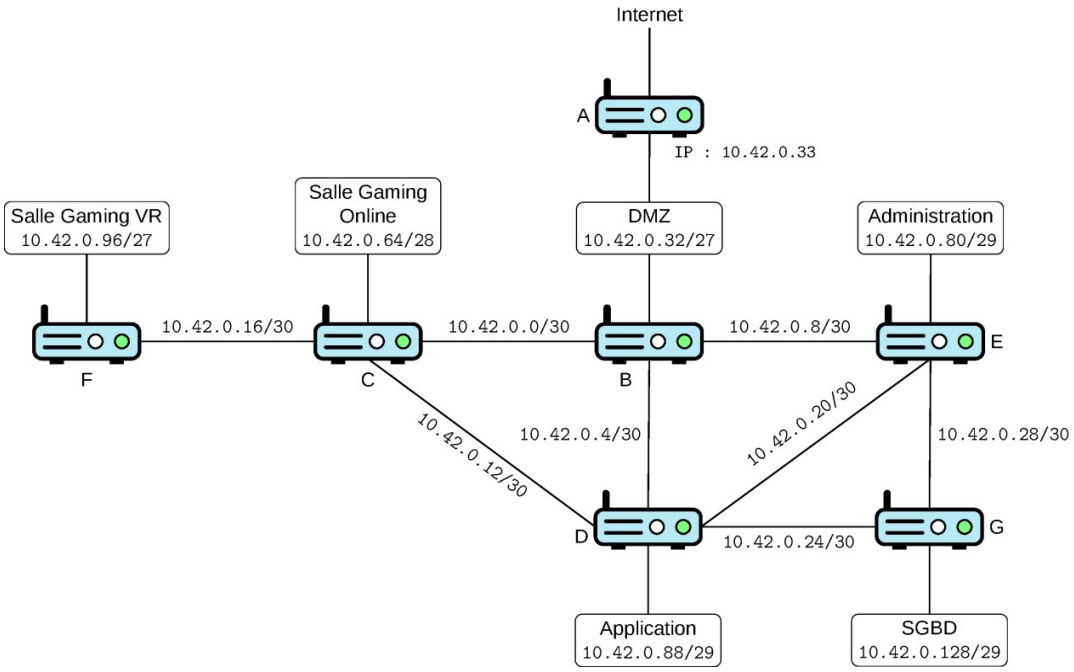


Figure 1. Réseau de l'entreprise Gamerzz

On y a fait figurer les différents sous-réseaux qui la composent en donnant leur notation CIDR.

La notation `a.b.c.d/n`, appelée notation CIDR (Classless Inter Domain Routing), signifie que les n premiers bits à gauche de l'adresse IP représentent la partie « réseau », les bits à droite qui suivent représentent la partie « machine ». L'adresse IPv4 dont tous les bits de la partie « machine » sont à 0 est appelée « adresse du réseau ». L'adresse IPv4 dont tous les bits de la partie « machine » sont à 1 est appelée « adresse de diffusion », les autres adresses peuvent être attribuées à des machines telles que des routeurs, des serveurs ou des ordinateurs.

1. Donner le nombre d'adresses IP qui peuvent être attribuées à des machines dans le réseau « Administration », ainsi qu'une adresse possible pour le routeur E.

Des tentatives de téléchargements non autorisés sont détectées depuis l'IP 10.42.0.70.

2. Indiquer dans quel réseau se situe la machine correspondante.

Les réseaux qui interconnectent les routeurs sont en /30 ce qui permet d'attribuer une adresse à exactement deux machines, les deux autres adresses étant l'adresse de réseau et l'adresse de diffusion. On choisit, sur un tel réseau, d'attribuer les adresses IP des routeurs dans le même ordre que celui des lettres qui les désignent sur la figure 1.

3. Donner les adresses IP attribuées, selon ce principe, aux routeurs C et F dans le réseau 10.42.0.16/30, ainsi que les adresses IP attribuées aux routeurs C et D dans le réseau 10.42.0.12/30.

On choisit de configurer les routeurs suivant le protocole RIP. Le protocole RIP permet de minimiser le nombre de routeurs traversés par les paquets. Voici alors la table de routage du routeur B :

Routeur B		
Réseau	Passerelle	Nombre de sauts
DMZ	connecté	0
Gaming Online	10.42.0.2	1
Internet	10.42.0.33	1
Gaming VR	10.42.0.2	2
Administration	10.42.0.10	1
Application	10.42.0.6	1
SGBD	10.42.0.6	2

4. Donner une table possible pour le routeur C, en suivant le protocole RIP.

On indique en figure 2 les débits des liaisons entre routeurs.

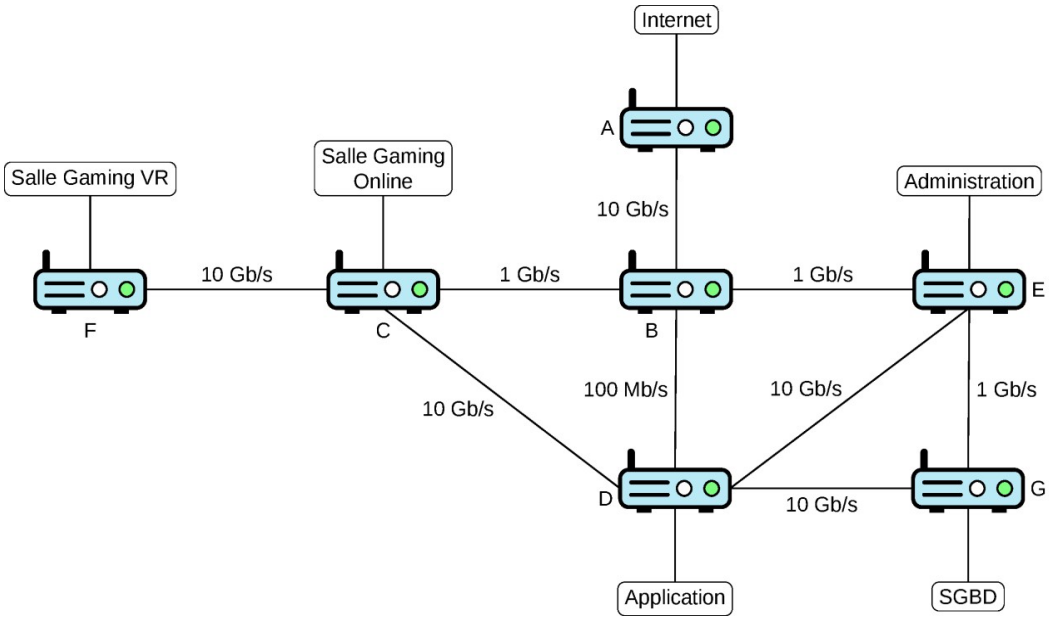


Figure 2. Débits des liaisons

On précise que :

• le coût d'une liaison est donné par $\frac{10^{10}}{d}$ où d est le débit de la liaison en bits par seconde;

• le protocole OSPF fait transiter les informations de routeur en routeur en minimisant le coût total de leur acheminement.

5. Donner, pour chaque débit présent en figure 2, le coût de la liaison.

6. Une information venue d'un client extérieur, par exemple depuis Internet, est acheminée vers le réseau « Application » pour être traitée. Donner un des ordres possibles des routeurs par lesquels cette information transite à partir du routeur A en suivant le protocole OSPF.

On s'intéresse désormais à la gestion des paquets TCP par un routeur.

7. Rappeler si les données d'un segment TCP sont contenues dans un paquet IP, ou bien si les données d'un paquet IP sont contenues dans un segment TCP.

Lorsqu'un routeur reçoit un paquet TCP, il le place dans une file d'attente avant de pouvoir le transmettre.

8. Expliquer pourquoi il est plus intéressant dans ce cas précis d'utiliser une file plutôt qu'une pile.

On modélise les files en Python à l'aide des fonctions suivantes :

- `cree_file` : renvoie une file vide ;
- `est_vide` : renvoie le booléen indiquant si la file `f` passée en paramètre est vide ;
- `enfile` : prend en paramètres une file `f` et un élément `x` et ajoute `x` dans `f` ;
- `defile` : prend en paramètre une file `f` non vide et supprime l'élément de `f` le plus anciennement ajouté et renvoie sa valeur.

On a par exemple :

```
>>> f = cree_file()
>>> est_vide(f)
True
>>> enfile(f, 0)
>>> enfile(f, 1)
>>> f
| 1 | 0 <- tête
>>> defile(f)
0
>>> f
| 1 <- tête
```

Il arrive qu'un routeur ne puisse pas transmettre l'ensemble des paquets qu'il reçoit, par exemple, si les émetteurs sont très actifs. Dans ce cas, le routeur va ignorer certains paquets.

L'une des démarches utilisées consiste à limiter la taille de la file. Si la file n'a pas atteint sa taille maximale, les paquets reçus sont enfilés. Par contre, si la taille maximale est atteinte, tous les nouveaux paquets reçus sont ignorés. Lorsqu'un paquet est transmis par le routeur, il est défilé.

Cette méthode est appelée *drop tail* en anglais. Un routeur suivant cet algorithme est donc paramétré avec une taille maximale de file `t_max` et il doit garder trace, à chaque instant, de la file `f` contenant les paquets à transmettre et de la taille `t` de celle-ci.

On considère la classe `Routeur_DROP_TAIL` dont on fournit ci-dessous une partie du code.

```
1 class Routeur_DROP_TAIL:
2     def __init__(..., ...):
3         self.f = ...
4         self.t_max = ...
5         self.t = ...
```

9. Recopier et compléter la méthode `__init__`.

La méthode `recoit` de la classe `Routeur_DROP_TAIL` prend en paramètre un paquet `p`. Cette méthode renvoie `True` si le paquet est accepté, `False` dans le cas contraire. On rappelle qu'un paquet est accepté si la taille de la file au moment de sa réception est strictement inférieure à la taille maximale. Dans ce cas le paquet est enfilé.

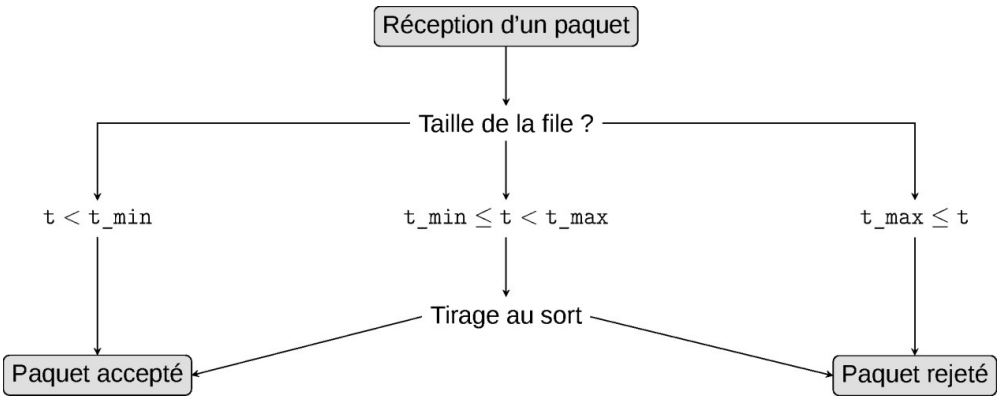
10. Recopier et compléter la méthode `recoit` proposée ci-dessous :

```
    def recoit(self, p):
        if ...:
            enfile(..., ...)
            self.t = ...
            return ...
        return ...
```

La démarche décrite ci-dessus a pour désavantage d'ignorer indifféremment tous les paquets lorsque la taille maximale de la file est atteinte. Elle peut entraîner un ralentissement général des transmissions car tous les émetteurs vont ralentir leur rythme d'envoi de paquets en même temps.

Pour palier ce problème, on se propose d'adopter la démarche suivante lors de la réception d'un paquet :

- si la taille actuelle de la file est strictement inférieure à une taille minimale `t_min`, le paquet est accepté ;
- si elle est supérieure ou égale à `t_min` mais strictement inférieure à une taille maximale `t_max`, le paquet est rejeté aléatoirement ;
- si elle est supérieure ou égale à `t_max`, le paquet est rejeté.



On modélise cette démarche par un objet de la classe `Routeur_ALEA` possédant quatre attributs :

- la file d'attente `f` manipulable par les fonctions décrites plus haut. Cette file est vide initialement ;
- la taille minimale `t_min` (nombre entier positif) ;
- la taille maximale `t_max` (nombre entier positif) ;
- la taille actuelle de la file `t` (nombre entier positif, initialement nul).

La classe `Routeur_ALEA` possède aussi une méthode `tirage_au_sort` qui renvoie un booléen au hasard. Ainsi, l'expression `self.tirage_au_sort()` renverra aléatoirement `True` ou `False`.

La méthode `recoit` de la classe `Routeur_ALEA` prend en paramètre un paquet `p`. Cette méthode met en œuvre la démarche décrite ci-dessus et renvoie `True` si le paquet est accepté, `False` dans le cas contraire.

11. Recopier et compléter la méthode `recoit` proposée ci-dessous :

```
    def recoit(self, p):
        if ... < ...:
            enfile(self.f, p)
            self.t = ...
            return ...
        elif ... <= ... < ...:
            if self.tirage_au_sort():
                enfile(self.f, p)
                self.t = ...
                return ...
        return ...
```