

---

# Serveur DHCP basique

-----

Professeurs : P. SPATHIS

-----

Étudiants : E. BRUN, E. OBLETTE, A. PETROSSI

---

## Table des Matières

1. Résumé	2
2. Introduction	2
3. Initialisation d'un serveur DHCP	2
4. Réseau avec accès à internet	3
5. Commandes Serveur	4
6. Analyse	6
7. Code	7
8. Conclusion	10

## Résumé

Ce rapport décrit la conception d'un serveur DHCP qui s'exécute sur une machine standard quel que soit son système d'exploitation, capable de répondre aux requêtes DHCP en provenance d'autres machines connectées au même réseau local. Dans ce rapport nous introduisons le fonctionnement de notre serveur puis donnons une description plus détaillée des méthodes utilisées. Nous finissons par une analyse de notre serveur afin de montrer ses forces et ses faiblesses.

## Introduction

Notre code a été réalisé en Python 3. Il initialise un serveur DHCP sur la machine d'où il est lancé, qui supporte les quatre messages : Discover, Offer, Request et Acknowledgment.

Pour arriver à cette fin, le serveur DHCP est bien entendu configurable avec la plage d'adresse IP à allouer aux clients, l'adresse réseau et masque du sous-réseau local, ainsi que l'adresse du routeur par défaut (gateway).

Nous avons tout d'abord travaillé sur une version fonctionnant en local, dite *standalone*, sans accès à internet, puis nous avons ajouté certaines capacités à ce serveur afin qu'il fonctionne également sur un réseau avec accès à internet remplaçant un serveur DHCP existant.

## Initialisation d'un serveur DHCP

Pour initialiser le serveur, une fois le "server.py" lancé, on lui donne les arguments suivants : l'adresse IP du serveur, l'adresse IP du routeur (gateway), le masque de sous-réseau, la plage d'adresses que l'on veut attribuer, le temps de lease (bail), et enfin les serveurs DNS primaires et secondaires. Dans le cas d'un test en local, on peut par exemple initialiser le serveur avec les informations suivantes :



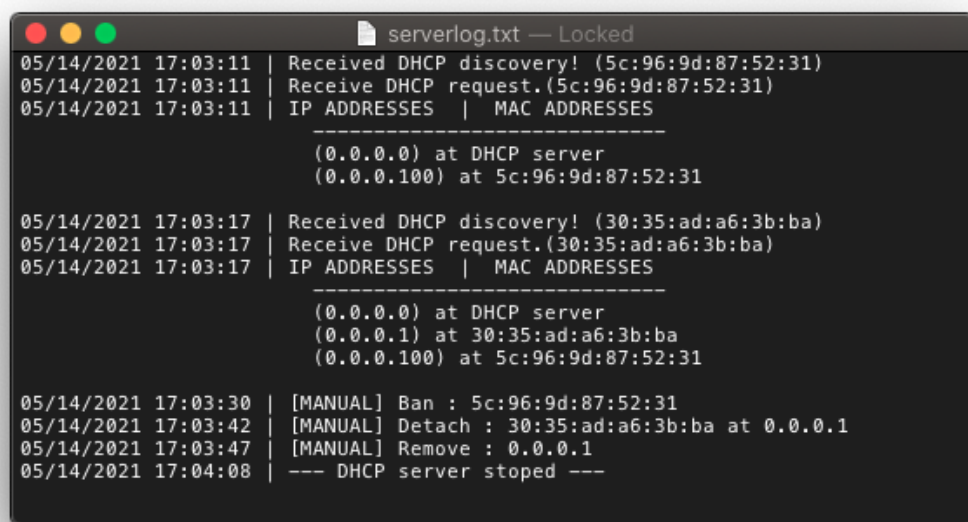
```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...
> sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777 1.2.3.4
Network: 0.0.0.0
DHCP server: 0.0.0.0
Gateway/Router: 0.0.0.0
Broadcast: 0.0.0.255
Mask: 255.255.255.0
Cidr: 24
Server info:
```

Figure 1.1 : Initialisation du serveur DHCP avec les arguments nécessaires.

Le serveur est donc déployé sur le réseau "0.0.0.0", et on peut par la suite lancer notre client.py sur le même réseau et sur la même machine afin qu'il envoie des requêtes DHCP au serveur,

auxquelles celui-ci va répondre. De cette manière on teste si l'attribution d'une adresse IP, et la gestion des requêtes du client par le serveur fonctionnent sur un réseau fictif.

En plus de cela, notre serveur stocke un journal de l'ensemble des échanges réalisés avec d'autres machines dans un fichier texte. Ce journal recense aussi les commandes manuelles réalisées en ligne de commande par l'utilisateur.



```
05/14/2021 17:03:11 | Received DHCP discovery! (5c:96:9d:87:52:31)
05/14/2021 17:03:11 | Receive DHCP request.(5c:96:9d:87:52:31)
05/14/2021 17:03:11 | IP ADDRESSES | MAC ADDRESSES
                        -----
                        (0.0.0.0) at DHCP server
                        (0.0.0.100) at 5c:96:9d:87:52:31

05/14/2021 17:03:17 | Received DHCP discovery! (30:35:ad:a6:3b:ba)
05/14/2021 17:03:17 | Receive DHCP request.(30:35:ad:a6:3b:ba)
05/14/2021 17:03:17 | IP ADDRESSES | MAC ADDRESSES
                        -----
                        (0.0.0.0) at DHCP server
                        (0.0.0.1) at 30:35:ad:a6:3b:ba
                        (0.0.0.100) at 5c:96:9d:87:52:31

05/14/2021 17:03:30 | [MANUAL] Ban : 5c:96:9d:87:52:31
05/14/2021 17:03:42 | [MANUAL] Detach : 30:35:ad:a6:3b:ba at 0.0.0.1
05/14/2021 17:03:47 | [MANUAL] Remove : 0.0.0.1
05/14/2021 17:04:08 | --- DHCP server stoped ---
```

Figure 1.2 : Journal de bord du serveur DHCP listant les évènements auxquels il a été confronté.

## Réseau avec accès à internet

La version déployée localement sur une machine et celle se déployant sur un réseau local avec accès à internet ne sont pas si différentes, une fois que nous avons conçu la version standalone il a suffi que nous ajoutions la possibilité de paramétrer le serveur avec les différentes caractéristiques du réseau local sur lequel on se trouvait.

Par exemple, si l'on teste notre serveur sur une machine ayant l'adresse "192.168.1.21", sur le réseau "192.168.1.0/24", avec comme adresse pour le routeur la "192.168.1.1", il suffisait de rentrer ses différentes informations en arguments lors du lancement de notre serveur pour que celui-ci fonctionne sur le réseau local en question avec accès à internet, en écoutant et répondant aux différentes requêtes des dispositifs connectés.

Ainsi notre serveur DHCP écrit en Python peut très bien remplacer le serveur DHCP intégré à un routeur sur un réseau local: il peut allouer dynamiquement des adresses IP en répondant de manière pertinente aux requêtes des clients sur un réseau.

```
Network: 192.168.1.0  
DHCP server: 192.168.1.21  
Gateway/Router: 192.168.1.1  
Broadcast: 192.168.1.255  
Mask: 255.255.255.0  
Cidr: 24
```

Figure 2.1 : Informations visualisées au lancement du serveur.

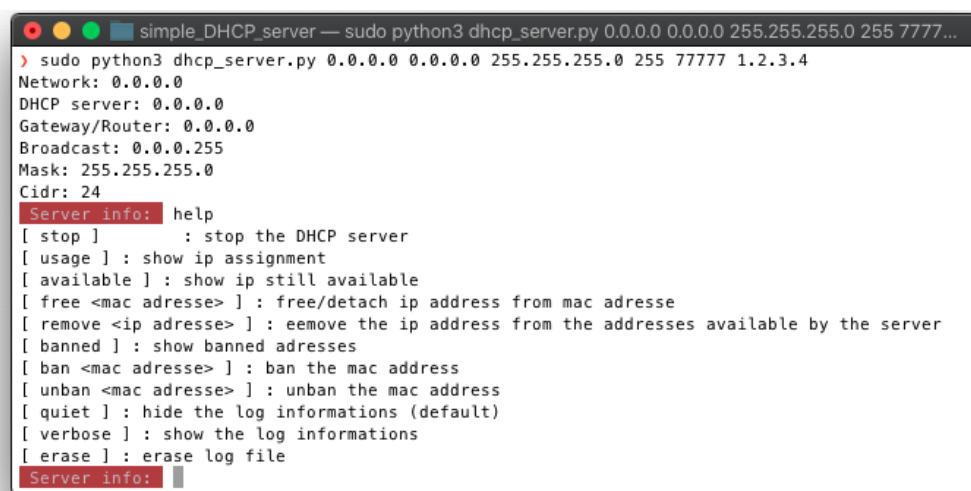
```
Server info: usage  
IP ADDRESSES | MAC ADDRESSES  
-----  
(192.168.1.1) at gateway  
(192.168.1.21) at DHCP server  
(192.168.1.51) at 5e:1d:21:21:d6:b6
```

Figure 2.2 : Adresses IP allouées aux gateway, serveur et à la machine d'adresse MAC 5e:1d:21:21:d6:b6.

## Commandes Serveur

Dans un deuxième temps nous avons ajouté des fonctionnalités supplémentaires à notre serveur, comme la possibilité de bannir une adresse MAC spécifique, ou de visualiser l'ensemble des adresses IP encore disponibles à l'attribution. Ces diverses fonctionnalités sont exécutables en ligne de commande pendant que le serveur est actif.

Ces différentes fonctionnalités ajoutées permettent un fonctionnement plus complet de notre serveur, notamment quand il s'agit de le faire fonctionner sur un réseau local avec plusieurs dispositifs connectés.



```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...  
> sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 77777 1.2.3.4  
Network: 0.0.0.0  
DHCP server: 0.0.0.0  
Gateway/Router: 0.0.0.0  
Broadcast: 0.0.0.255  
Mask: 255.255.255.0  
Cidr: 24  
Server info: help  
[ stop ] : stop the DHCP server  
[ usage ] : show ip assignment  
[ available ] : show ip still available  
[ free <mac adresse> ] : free/detach ip address from mac adresse  
[ remove <ip adresse> ] : eemove the ip address from the addresses available by the server  
[ banned ] : show banned addresses  
[ ban <mac adresse> ] : ban the mac address  
[ unban <mac adresse> ] : unban the mac address  
[ quiet ] : hide the log informations (default)  
[ verbose ] : show the log informations  
[ erase ] : erase log file  
Server info: █
```

Figure 3.1 : Commande help qui affiche une explication détaillée de toutes les commandes gérées par le serveur DHCP.

```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...
Server info: usage
IP ADDRESSES | MAC ADDRESSES
-----
(0.0.0.0) at DHCP server
(0.0.0.1) at 30:35:ad:a6:3b:ba
(0.0.0.100) at 5c:96:9d:87:52:31
```

Figure 3.2 : Commande usage qui affiche toutes les adresses IP allouées aux machines clientes.

```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...
Server info: available
IP availables : 252
(0.0.0.2)
(0.0.0.3)
(0.0.0.4)
(0.0.0.5)
(0.0.0.6)
(0.0.0.7)
(0.0.0.8)
(0.0.0.9)
(0.0.0.10)
(0.0.0.11)
(0.0.0.12)
(0.0.0.13)
```

Figure 3.3 : Commande available qui affiche les adresses IP allouables restantes.

```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...
Server info: usage
IP ADDRESSES | MAC ADDRESSES
-----
(0.0.0.0) at DHCP server
(0.0.0.1) at 30:35:ad:a6:3b:ba
(0.0.0.100) at 5c:96:9d:87:52:31

Server info: free 30:35:ad:a6:3b:ba
30:35:ad:a6:3b:ba at 0.0.0.1 detached
Server info: usage
IP ADDRESSES | MAC ADDRESSES
-----
(0.0.0.0) at DHCP server
(0.0.0.100) at 5c:96:9d:87:52:31

Server info: █
```

Figure 3.4 : Commande free qui libère une adresse IP de l'affectation à une adresse MAC, elle rend disponible l'adresse IP pour une nouvelle attribution.

```
simple_DHCP_server — sudo python3 dhcp_server.py 0.0.0.0 0.0.0.0 255.255.255.0 255 7777...

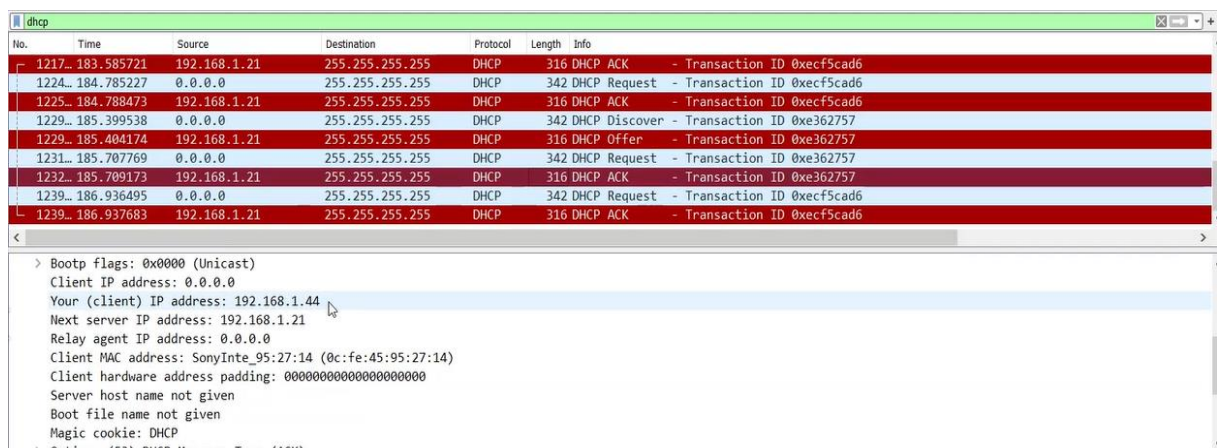
Server info: remove 0.0.0.1
0.0.0.1 removed
Server info: available
IP disponibles : 252
(0.0.0.2)
(0.0.0.3)
(0.0.0.4)
(0.0.0.5)
(0.0.0.6)
(0.0.0.7)
(0.0.0.8)
(0.0.0.9)
(0.0.0.10)
(0.0.0.11)
```

Figure 3.5 : Commande remove qui retire l'adresse IP de la base d'adresse du serveur, elle n'est donc plus attribuée et n'est plus attribuable.

## Analyse

Afin de tester notre code nous avons dans un premier temps implémenté un *client.py*, un client donc, capable de communiquer avec notre serveur, de lui demander une adresse IP, etc. Nous avons tout d'abord effectué des tests en local, sur une seule et même machine, en donnant au client une adresse MAC choisie. Puis nous avons refait des tests cette fois-ci sur un réseau local en remplaçant le serveur DHCP du routeur avec le nôtre, ce qui nous a permis de tester l'attribution dynamique d'adresses IP à plusieurs dispositifs connectés au réseau.

Pour vérifier l'échange de trames entre ces appareils et notre serveur nous avons fait usage de l'outil *Wireshark*. Ci-dessous un exemple d'échanges capturés sur Wireshark entre plusieurs dispositifs et notre serveur :



No.	Time	Source	Destination	Protocol	Length	Info
1217	183.585721	192.168.1.21	255.255.255.255	DHCP	316	DHCP ACK - Transaction ID 0xecf5cad6
1224	184.785227	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0xecf5cad6
1225	184.788473	192.168.1.21	255.255.255.255	DHCP	316	DHCP ACK - Transaction ID 0xecf5cad6
1229	185.399538	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0xe362757
1229	185.404174	192.168.1.21	255.255.255.255	DHCP	316	DHCP Offer - Transaction ID 0xe362757
1231	185.707769	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0xe362757
1232	185.709173	192.168.1.21	255.255.255.255	DHCP	316	DHCP ACK - Transaction ID 0xe362757
1239	186.936495	0.0.0.0	255.255.255.255	DHCP	342	DHCP Request - Transaction ID 0xecf5cad6
1239	186.937683	192.168.1.21	255.255.255.255	DHCP	316	DHCP ACK - Transaction ID 0xecf5cad6

> Bootp flags: 0x0000 (Unicast)  
Client IP address: 0.0.0.0  
Your (client) IP address: 192.168.1.44  
Next server IP address: 192.168.1.21  
Relay agent IP address: 0.0.0.0  
Client MAC address: SonyInte\_95:27:14 (0c:fe:45:95:27:14)  
Client hardware address padding: 00000000000000000000  
Server host name not given  
Boot file name not given  
Magic cookie: DHCP  
> Option: (53) DHCP Message Type (DHCP)

Figure 4.1 : Trames sur Wireshark

Ainsi nous avons testé notre code en Python au fur et à mesure de son écriture sur des terminaux, nous observions les résultats et en même temps nous vérifions si les échanges de trames s'effectuaient correctement grâce à *Wireshark*.

## Code

### 1. *SERVER\_DHCP* :

C'est la classe principale du projet, la classe d'où est lancé le serveur DHCP.

#### 1.1 *SERVER* :

C'est la principale fonction du projet, tout simplement parce que c'est la fonction qui lance le serveur DHCP.

#### 1.2 *START* :

Cette fonction sert au lancement du serveur ainsi qu'au bon déroulement des échanges de messages entre serveur et client.

#### 1.3 *STOP* :

Et donc celle-ci sert à arrêter le serveur.

#### 1.4 *GUI* :

Dans cette fonction nous ajoutons toutes les commandes que nous voulons disponibles sur le serveur. Par exemple *usage*, qui renvoie les adresses IP allouées et les adresses MAC auxquelles elles sont allouées, ou *available* qui renvoie toutes les adresses IP encore disponibles.

#### 1.5 *IP\_ADDR\_FORMAT* :

Cette fonction sert à récupérer l'adresse IP d'une machine connectée au réseau local du serveur DHCP puis à modifier son format de façon à ce que celle-ci s'intègre parfaitement au reste du paquet d'information envoyé.

#### 1.6 *MAC\_ADDR\_FORMAT* :

Cette fonction sert à récupérer l'adresse MAC d'une machine connectée au réseau local du serveur DHCP puis à modifier son format de façon à ce que celle-ci s'intègre parfaitement au reste du paquet d'information envoyé.

#### 1.7 *PACKET\_ANALYSER* :

Cette fonction sert tout simplement à récupérer un message venant d'un client.

#### 1.8 *SET\_OFFER* :

Cette fonction sert tout simplement à envoyer le message *offer* à un client dont le serveur a reçu un message *discover*. Le serveur lui offre donc une adresse IP de libre si le client n'a pas fait de demande spécifique.



### 1.9 **PACK\_GET :**

Cette fonction renvoie l'acknowledgment de l'allocation de l'adresse IP indiquée dans le message *request* reçu du client.

### 1.10 **INFO\_MSG :**

Cette fonction ajoute au journal tout événement auquel le serveur a été confronté, que ce soit un événement manuel ou automatique.

### 1.11 **ERROR\_MSG :**

Cette fonction renvoie tel ou tel message d'erreur selon l'erreur rencontrée par le serveur, par exemple quand plus aucune adresse IP est disponible, l'erreur 0, ça renverra : *ERROR (No more IPs available)*.

### 1.12 **CLEAR\_LOG :**

Cette fonction efface tout le contenu du journal. Active quand la commande "erase" est lancée.

## 2. **IPVECTOR :**

Dans cette classe sont toutes les fonctionnalités nécessaires pour une allocation satisfaisante d'adresses IP aux clients en réquisitionnant une au serveur DHCP, et finalement nécessaires au bon fonctionnement du serveur lui-même.

### 2.1 **INIT :**

Initialise tous les arguments nécessaires au lancement du serveur DHCP.

### 2.2 **ADD\_IP :**

Cette fonction met en place un nombre d'adresses IP disponibles équivalent au nombre passé à l'argument *range*. Elle les place dans un dictionnaire dont nous allons nous servir pour les allouer à des clients en demandant.

### 2.3 **UPDATE\_IP :**

Cette fonction met à jour la liste d'adresses IP disponibles suivant celles qui sont allouées.

### 2.4 **REMOVE\_IP :**

Cette fonction retire une adresse IP de la liste.

### 2.5 **DETACH\_IP :**

Cette fonction détache une adresse IP d'un client s'il elle est allouée.



### **2.6    *BAN\_ADDR* :**

Cette fonction bannit une adresse MAC, dont le serveur n'écouterà plus les messages.

### **2.7    *UNBAN\_ADDR* :**

Le contraire de la fonction précédente.

### **2.8    *GET\_BANNED\_ADRESSES* :**

Cette fonction renvoie la liste des adresses mises au ban.

### **2.9    *GET\_BROADCAST\_ADRESS* :**

Cette fonction renvoie l'adresse broadcast.

### **2.10   *GET\_IP* :**

Cette fonction trouve l'adresse IP précédemment allouée à un client. Si celui-ci n'en avait pas mais qu'il spécifie l'adresse à laquelle il veut être associé, cette fonction renverra cette adresse. Si nous ne nous trouvons dans aucun de ces cas, la fonction renvoie vers *get\_free*.

### **2.11   *GET\_FREE* :**

Cette fonction alloue une adresse IP disponible à un client en demandant, sans spécificité particulière.

### **2.12   *GET\_IP\_ALLOCATED* :**

Cette fonction montre quelles adresses IP ont été allouées à quels clients.

### **2.13   *GET\_IP\_AVAILABLE* :**

Cette fonction montre quelles adresses IP sont encore disponibles.

## **3.    *MAIN* :**

Dans le main nous spécifions directement les arguments à donner au serveur lors du lancement (affiché sur le terminal). Également, le fichier du journal est ouvert pour écrire tous les événements auquel est confronté notre serveur. Puis le main lance le serveur et les deux threads, le premier étant celui qui exécute les fonctions DHCP et le deuxième servant à l'affichage des informations.

## Conclusion

Au cours de ce projet nous avons beaucoup appris sur le fonctionnement d'un serveur DHCP, et sur sa conception. Nous nous sommes également amusés à ajouter diverses fonctionnalités supplémentaires à notre serveur une fois celui-ci fonctionnel, notamment activables sur commande du terminal. C'était un plaisir de travailler à plusieurs, ça a ajouté du dynamisme au développement du projet. Nous avons très bien travaillé ensemble, chacun faisant une part du travail.

Globalement nous sommes assez satisfaits de notre travail, d'une part car notre serveur DHCP est entièrement fonctionnel mais aussi puisque nous avons été en mesure de le tester sur un véritable réseau local, et d'observer des résultats pertinents. Notre serveur DHCP remplaçait très bien celui intégré à un routeur par exemple, il attribuait correctement des adresses IP de manière dynamique à des appareils connectés au réseau (iPhone, PS4, Switch).

Ce projet nous a donc beaucoup plu et intéressé. Il nous a permis de comprendre le fonctionnement du protocole DHCP de manière détaillée, et de mettre en pratique nos connaissances acquises pour créer notre propre serveur. C'est un projet qui a renforcé notre intérêt dans le domaine de l'informatique, plus spécialement des réseaux, et nous a motivé à hausser notre niveau à celui des défis de demain.