# Introduction

Les ordinateurs portables, tablettes, téléphones, objets connectés utilisent abondamment les réseaux sans fil pour maintenir le contact. La propagande marketing veut nous faire croire que c'est facile d'utilisation et intelligent. Dans la réalité, il y a souvent des couacs. Quand la connexion est perdue, l'angoisse monte et la pratique magique consiste à cliquer pour rétablir la communication. S'il vous reste des points de karma, alors les dieux vous octroient un peu de bonheur.

Parfois, ce n'est pas simple de cliquer. N'étant pas adepte des GAFAM, je me permet de me connecter sur mon ordinateur portable quand nous ne sommes pas dans la même pièce. Si sa connexion WiFi est coupée, alors je dois retourner le voir pour réactiver manuellement le WiFi. Mais est-ce l'ordinateur qui fait ce que je demande ou l'ordinateur qui m'impose ce déplacement ? C'est qui le chef ?

Alors, en bon utilisateur de GNU/Linux et de logiciels libres, je vais me débrouiller pour convaincre mon ordinateur de revenir sur le réseau quand il l'a perdu. Tout a commencé par une ligne de shell qui teste la connexion et la réactive quand il le faut. Mais, comment tester la connexion ? Un ping vers le routeur ? Mais celui-ci change quand le portable change d'endroit. J'ai donc décidé de scripter cela.

Une autre approche consisterait à corriger le dysfonctionnement et ne plus avoir de soucis. Il faut donc commencer par savoir quel est le coupable parmi : la box internet, le composant WiFi de mon ordinateur, le noyau Linux, le logiciel intelligent de gestion des interfaces réseaux. Cela sort largement du cadre de cet article et je n'ai pas forcément l'envie ou les compétences pour cela.

Ma solution permet aussi d'ouvrir la connectivité WiFi vers des usages non prévus. Par exemple, un objet connecté pourrait avoir besoin du WiFi. Si le point d'accès reboote, alors l'objet pourrait ne pas se reconnecter. Le client WiFi peut aussi se connecter sur un mauvais point d'accès. Les

points ouverts sans chiffrement permettent une connexion souvent incomplète ou imposant des identifiants ultérieurs. Un bus pourrait prédire qu'il va rentrer dans une zone d'ombre de la 6G et préparer la connexion WiFi dans cette zone. Bref, un comportement inutile pour la plupart des utilisateurs.

Nous allons donc commencer par vérifier notre matériel. Puis nous regarderons les commandes les plus basiques sous GNU/Linux de gestion des réseaux. Enfin, nous regarderons comment scripter pour adapter le comportement à nos besoins. Nous ne présenterons pas une nouvelle solution intelligente qui prévoit tout, c'est à l'utilisateur de partir des exemples et de les adapter à son besoin personnel.

# 1. Détection du matériel

Pour pouvoir utiliser le WiFi, il faut commencer par disposer d'un ordinateur équipé d'un composant spécifique. Pour vérifier si ce composant est détecté par le système, il y a plusieurs stratégies : une commande listant le matériel, l'exploration des pseudo systèmes de fichiers ou les arbres de composants (*device tree* : définition des composants non détectables). Le Raspberry 3B n'a son composant ni sur le bus PCI, ni sur le bus USB.

## 1.1 Listez vos composants

Le périphérique est souvent sur un bus PCI ou USB, les commandes sont alors lspci et lsusb spécifiquement. Pour lister les composants du bus PCI, nous utilisons la commande avec l'option verbeuse et le compte administrateur pour obtenir toutes les informations :

```
# lspci -v

...

02:00.0 Network controller: Intel Corporation Wireless 8260 (rev 3a)

        Subsystem: Intel Corporation Dual Band Wireless-AC 8260
```

Flags: bus master, fast devsel, latency 0, IRQ 140, IOMMU group 12

Memory at cc100000 (64-bit, non-prefetchable) [size=8K]

Capabilities: [c8] Power Management version 3

Capabilities: [d0] MSI: Enable+ Count=1/1 Maskable- 64bit+

Capabilities: [40] Express Endpoint, MSI 00

Capabilities: [100] Advanced Error Reporting

Capabilities: [140] Device Serial Number Z3-de-b3-ff-ff-42-30-03

Capabilities: [14c] Latency Tolerance Reporting

Capabilities: [154] L1 PM Substates

Kernel driver in use: iwlwifi

Kernel modules: iwlwifi

...

Nous obtenons, en particulier, le numéro de série (qui sera éventuellement utilisé comme adresse MAC) et le nom du pilote. Cette carte semble être reconnue par le noyau Linux actuel.

Si le composant WiFi est sur le bus USB, alors c'est la commande lsusb qu'il faut utiliser. Elle produit 108 lignes sur mon exemple, voici un extrait :

$ lsusb -v

Bus 001 Device 009: ID 0cf3:7015 Qualcomm Atheros Communications TP-Link TL-WN821N v3 / TL-WN822N v2 802.11n [Atheros AR7010+AR9287]

Device Descriptor:

...

idVendor 0x0cf3 Qualcomm Atheros Communications

idProduct 0x7015 TP-Link TL-WN821N v3 / TL-WN822N v2 802.11n [Atheros AR7010+AR9287]

iProduct 32 USB WLAN

iSerial 48 12345

...

Une autre commande explore le matériel, c'est lshw. Elle est très verbeuse et fournit beaucoup d'informations (437 lignes) dont le numéro de série du matériel.

*-network

description: Interface réseau sans fil

produit: Wireless 8260

fabriquant: Intel Corporation

identifiant matériel: 0

information bus: pci@0000:02:00.0

nom logique: wlp2s0

version: 3a

numéro de série: Z3:de:b3:42:30:03

bits: 64 bits

horloge: 33MHz

fonctionnalités: pciexpress ethernet physical wireless

configuration: broadcast=yes driver=iwlwifi ...

Enfin, la commande hw-probe fournit un résultat esthétique, présenté dans la figure 1. L'installation du paquet hw-probe rajoute 153 paquets sur mon installation minimaliste. Le résultat est disponible sur le serveur de *Hardware for Linux* **[1]**. Ils indiquent que l'anonymat de la configuration est préservé.

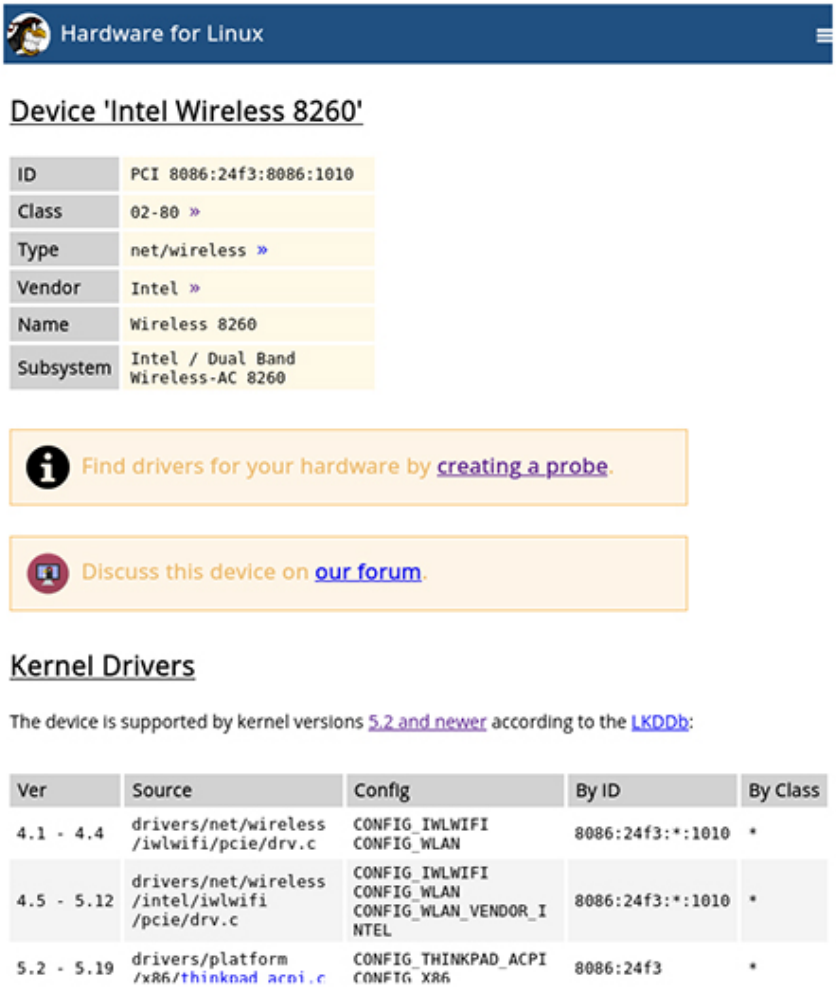


*Fig. 1 : L'interface WiFi présentée par Hardware for Linux.*

Il nous reste à mentionner les *device trees*. Dans le monde de l'embarqué, comme les Raspberry Pi, certains composants ne peuvent pas être détectés par le système. Par exemple, une diode connectée à une broche d'entrée-sortie ne pourra pas répondre pour indiquer sa présence et ses caractéristiques. Dans ce cas, un fichier *dtb* permet à Linux d'utiliser ce composant. Pour le Raspberry, les sources du noyau Linux disposent de ce fichier qui identifie l'interface WiFi.

Jusqu'ici nous avons uniquement vérifié si le composant existait. Pour un équipement USB ou PCI, il doit pouvoir répondre son identifiant même s'il

n’est pas compatible avec la version du système d’exploitation. Nous allons maintenant vérifier si nous pouvons l’utiliser avec notre noyau Linux.

## 1.2 Activation de l’interface

Linux permet d’utiliser la plupart des périphériques dont je dispose depuis longtemps. Néanmoins, il y a quelques détails qui peuvent choquer les personnes sensibles.

Tout d’abord, l’utilisation d’un matériel utilise souvent deux programmes: le pilote (*driver*) et le microgiciel (*firmware*). Le pilote est un programme qui se lie au noyau et implémente les fonctions spécifiquement pour ce matériel. Le microgiciel est un programme qui est chargé dans le périphérique et qui permet de modifier le comportement de celui-ci.

Le support d’un matériel spécifique est incorporé dans le noyau Linux à un instant donné. Il se peut alors qu’une distribution ne puisse pas directement utiliser ce matériel et qu’il faudra attendre une version ultérieure. Néanmoins, il y a des méthodes pour ajouter le support d’un matériel sur une vieille distribution, comme les *backports* pour Debian.

Lors du lancement du système ou de l’insertion du périphérique, Linux essaie de télécharger le firmware dans le composant. Le fichier syslog ou la commande dmesg permet de vérifier cela. Si le firmware n’est pas disponible, alors nous aurons un échec :

kernel: [ ] usb 1-9: new high-speed USB device number 3

kernel: [ ] usb 1-9: New USB device found, \

idVendor=0cf3, idProduct=7015

kernel: [ ] usb 1-9: New USB device strings: \

Mfr=16, Product=32, SerialNumber=48

kernel: [ ] usb 1-9: Product: USB WLAN

kernel: [ ] usb 1-9: Manufacturer: ATHEROS

kernel: [ ] usb 1-9: SerialNumber: 12345

kernel: [ ] usb 1-9: ath9k_htc: \

Firmware ath9k_htc/htc_7010-1.4.0.fw requested

kernel: [ ] usb 1-9: firmware: \

failed to load ath9k_htc/htc_7010-1.4.0.fw (-2)

kernel: [ ] usb 1-9: Direct firmware load for \

ath9k_htc/htc_7010-1.4.0.fw failed with error -2

kernel: [ ] usb 1-9: ath9k_htc: Firmware htc_7010.fw requested

kernel: [ ] usb 1-9: firmware: failed to load htc_7010.fw (-2)

kernel: [ ] usb 1-9: Direct firmware load for htc_7010.fw failed

kernel: [ ] usb 1-9: no suitable firmware found!

kernel: [ ] usb 1-9: ath9k_htc: USB layer deinitialized

kernel: [ ] usbcore: registered new interface driver ath9k_htc

Dans ce cas particulier, il faut ajouter le paquet firmware-atheros (qui n'est pas libre).

Si tout est correct, l'interface réseau apparaît :

$ ip a

5: wlxf8d111110a08: <NO-CARRIER,BROADCAST,MULTICAST,UP>...

link/ether 7a:66:9a:ec:78:de brd ff:ff:ff:ff:ff:ff ...

L'interface réseau existe, mais n'est pas configurée. Techniquement, tout

va bien. Le nom de l'interface est prévisible. Cela veut dire que cette clef USB aura le même nom d'interface quel que soit le système Linux. Si on la remplace par une clef identique, il faudra modifier ce nom.

## 2. Autopsiez votre équipement

Certes, d'après votre vendeur, votre équipement WiFi est le meilleur choix possible. Pour pouvoir choisir réellement, il faudrait disposer de données fiables, ce que je ne sais pas obtenir. Il faudrait, d'abord, pouvoir disposer de deux informations : les capacités de transmission et ce que l'interface est capable de faire du point de vue réseau.

Pour les capacités de transmission, ce qui nous intéresse, c'est la faculté d'émettre et de recevoir un signal de qualité dans le contexte d'utilisation réelle. Les perturbations sont multiples : distance, murs, WiFi des voisins, émissions perturbantes dans la même bande de fréquence (Bluetooth, micro-ondes…). C'est un élément important, mais il est difficile d'établir une méthode de test en laboratoire qui permette de prédire ce qui va se passer dans ma maison. Quoiqu'il en soit, ce n'est pas au menu de cet article.

Par contre, nous allons un peu plus regarder le point de vue des réseaux. Nous supposerons ne pas avoir de problèmes de transmission. Nous regarderons ce que la clef peut nous offrir comme connectivité réseau.

Le périphérique est donc bien reconnu par Linux et potentiellement bien géré. L'interface réseau WiFi apparaît lors de la commande ip adress. Nous pouvons demander au périphérique ce qu'il sait faire. Nous allons utiliser la commande iw.

**Les commandes obsolètes**

Les commandes pour le réseau sans fil WiFi ont évolué. Comme pour le firewall (nft remplaçant iptables, ip6tables, ebtables…) et les commandes des interfaces réseaux (ip remplaçant ifconfig, route, arp…) la commande iw rend obsolète les vieilles commandes iwconfig, iwlist…

Les vénérables administrateurs ayant appris Unix dans les années 80 ont encore les vieilles commandes dans les doigts, surtout que pour activer une interface avec ip il faut deux commandes contre une seule pour ifconfig.

Néanmoins, l'avènement du nouveau siècle a vu l'arrivée de nouvelles commandes qui offrent plus de fonctionnalités et qui sont beaucoup plus carrées que les précédentes. Les vieilles commandes souffrent d'avoir dû évoluer avec les arrivées des nouveautés et donc incorporer une nouvelle verrue à chaque fois.

La commande iw, à l'instar de ip, se décline en plusieurs sous-commandes. Nous allons commencer par lister les caractéristiques (sous-commande list). Sur trois interfaces différentes (mon portable de 5 ans, une vieille clef USB, un Raspberry Pi), nous obtenons respectivement 276, 166 et 108 lignes. Voici un extrait commenté avec les éléments qui nous intéressent pour cet article :

```
$ iw list

Wiphy phy0

        wiphy index: 0

...

        Supported Ciphers:

                * WEP40 (00-0f-ac:1)

                * WEP104 (00-0f-ac:5)

...

        Supported interface modes:

                 * IBSS

                 * managed
```

```
                * AP
                * AP/VLAN
                * monitor
                * P2P-client
                * P2P-GO
                * P2P-device
...
        Band 1:
                Capabilities: 0x11ef
                        RX LDPC
...
                Bitrates (non-HT):
                        * 1.0 Mbps
                        ...
                        * 54.0 Mbps
                Frequencies:
                        * 2412 MHz [1] (22.0 dBm)
                        ...
                        * 2472 MHz [13] (22.0 dBm)
                        * 2484 MHz [14] (disabled)
        Band 2:                 Frequencies:
```

```
                * 5180 MHz [36] (22.0 dBm) (no IR)

                ...

                * 5680 MHz [136] (22.0 dBm) (no IR, radar detection)

                ...

                * 5905 MHz [181] (disabled)

        valid interface combinations:

                * #{ managed } <= 1, #{ AP, P2P-client, P2P-GO } <= 1, #{ P2P-
device } <= 1,

                  total <= 3, #channels <= 2

...
```

Nous obtenons donc les informations suivantes :

- Wiphy phy0 : nous distinguons le composant physique (*phy0*) de l'interface réseau (*wlan0*).
- Supported Ciphers : le chiffrement disponible dans le matériel embarqué. Les protocoles chiffrés reposent sur l'utilisation d'un protocole commun entre les équipements. Les nouveaux n'implémenteront plus les algorithmes obsolètes et les anciens ne pourront pas apprendre les nouveaux. La communication sera alors interdite.
- Supported interface modes : les modes de fonctionnement de l'interface. Les principaux sont *managed* (client d'un point d'accès) et *AP* (point d'accès).
- Band 1(2) : chaque bande de fonctionnement est détaillée, nous retrouvons les débits possibles et les fréquences utilisables.
- valid interface combinations : une interface peut être utilisée (éventuellement) simultanément comme client et point d'accès. Celle-ci permet de mettre en place un répéteur. Deux canaux (fréquences) sont simultanément utilisables, vraisemblablement un

dans la bande des 2,4GHz et un dans la bande des 5GHz.

Dans les fréquences utilisables par l'interface, certaines sont marquées *disabled* (désactivées). Selon les pays, les régulations sont différentes. C'est pour cela qu'il faut indiquer à la clef le code pays. Voici trois exemples (l'administrateur semble voyager beaucoup) :

iw reg get

country CN: DFS-FCC

country 00: DFS-UNSET

country FR: DFS-ETSI

Pour le définir, il est possible d'utiliser la commande iw reg set FR, ou pour rendre l'information permanente, modifier (avec une Debian) le fichier /etc/default/crda. Il faudra peut-être installer le paquet crda.

# Set REGDOMAIN to a ISO/IEC 3166-1 alpha2 country code so that iw(8) may set

# the initial regulatory domain setting for IEEE 802.11 devices which operate

# on this system.

REGDOMAIN=FR

Nous avons donc validé le support matériel dans le noyau Linux et exploré (un peu) les possibilités du matériel. Voyons maintenant comment gérer la clef.

## 3. Activez le mode client

Nous allons présenter plusieurs méthodes pour connecter un poste client à un réseau WiFi. Dans des anciens numéros de *Linux Pratique Hors Série*, nous avons présenté comment mettre en place des points d'accès :

classique **[3]**, avec deux réseaux distincts **[4]** ou avec l'activation de WPA2 Entreprise **[5]**.

Sur une station graphique, l'installation de la suite de bureau (GNOME, KDE, Xfce, LXDE…) installe aussi un gestionnaire de connexion comme NetworkManager ou Wicd. Si c'est un ordinateur portable, alors une interface conviviale, intuitive et intelligente (voire bornée) gère harmonieusement le réseau sans fil. Une boîte de dialogue vous demande en cas de besoin d'ajouter un secret pour pouvoir utiliser une nouvelle connexion. Si le poste mobile se déplace, alors le réseau se reconnecte silencieusement au nouveau point d'accès. Mais parfois ça ne marche pas !

Nous allons donc regarder comment faire sans l'application conviviale. De plus, certains installent un ordinateur (éventuellement tout petit) sans interface graphique.

La première chose à effectuer, c'est un scan pour détecter les points d'accès voisins. La figure 2 montre un scan effectué sur mon /e/Linux, au même moment, sur une Debian, la commande iw dev wlan0 scan produit 200 lignes d'informations dont nous avons extrait :

BSS b8:27:eb:a3:de:37(on wlan0)

freq: 2447

signal: -34.00 dBm

SSID: Brelis

Supported rates: 1.0* 2.0* 5.5* 11.0* 18.0 24.0 36.0 54.0

DS Parameter set: channel 8

BSS 16:3d:d5:ec:8d:cc(on wlan0)

freq: 2412

signal: -71.00 dBm

SSID: garage

DS Parameter set: channel 1

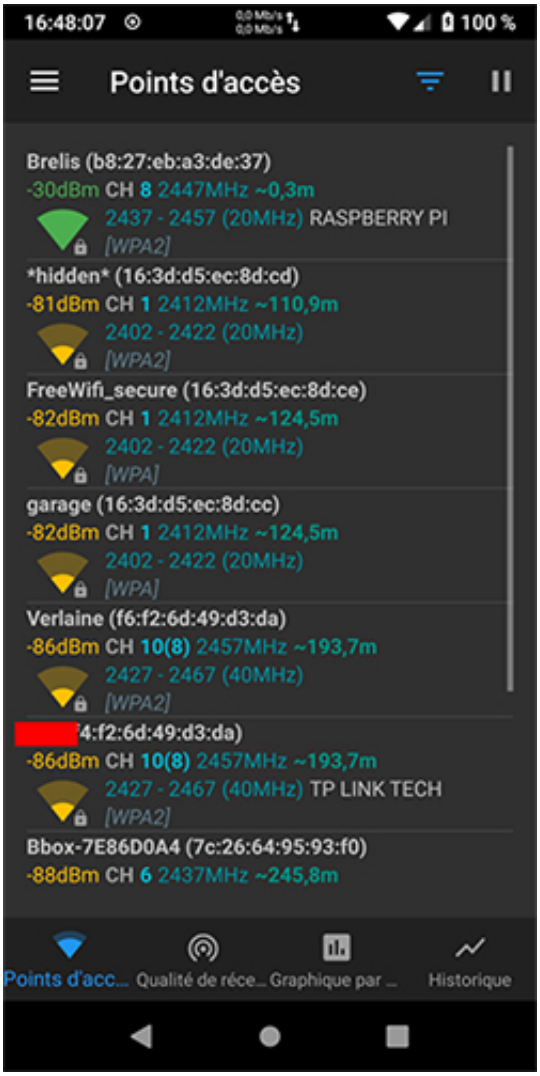


*Fig. 2 : Un scan /e/Linux.*

Les informations les plus utiles sont d'abord la puissance de réception. Plus le chiffre est bas, moins la connexion sera performante (attention au signe moins). L'autre information c'est le canal de transmission. Ici, à la campagne, les voisins ne polluent pas trop mon espace. Il y a donc uniquement mes points d'accès, l'un étant un Raspberry Pi sur mon bureau pour la rédaction de l'article.

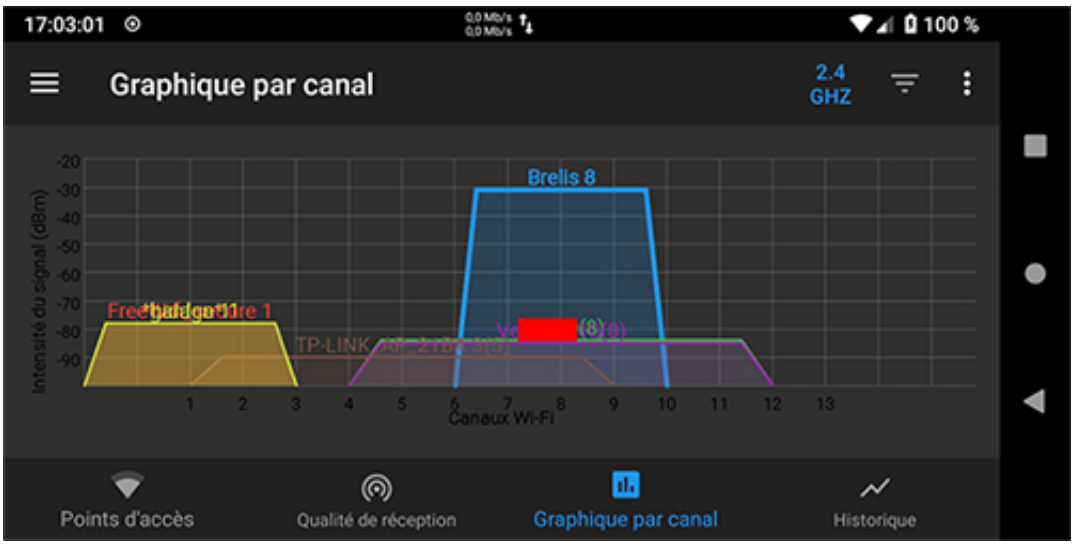


*Fig. 3 : La représentation graphique du WiFi avec /e/Linux.*

De ces informations, graphiques ou textuelles, nous pouvons choisir un point d'accès. Il faut avoir le droit de l'utiliser. Ensuite, le premier critère, c'est la puissance de réception. Il faut aussi regarder le nombre d'autres réseaux risquant de perturber notre connexion. Un analyseur de spectre pourrait mieux valider l'occupation des fréquences. Nous allons donc choisir de nous connecter au réseau Brelis. Nous allons créer un fichier wifi.conf dans le répertoire /etc/network/interfaces.d (attention au .d).

iface Wbrelis inet dhcp

  wpa-ssid Brelis

  wpa-psk SEZR12,/s@sSDF

Nous avons défini un point d'accès que nous nommons Wbrelis, avec le SSID et la phrase de passe. Pour activer le réseau, il suffit de dire que nous voulons activer l'interface WiFi sur le point d'accès désigné. Cela nous permet de définir autant de réseaux que nécessaire. Pour gagner du temps lors de l'activation, il est possible de se passer du DHCP et d'affecter les paramètres IP corrects dans ce fichier.

=> ifup wlxf8d111110a08=Wbrelis

Internet Systems Consortium DHCP Client 4.4.1

Copyright 2004-2018 Internet Systems Consortium.

All rights reserved.

For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/wlxf8d111110a08/f8:d1:11:11:0a:08

Sending on   LPF/wlxf8d111110a08/f8:d1:11:11:0a:08

Sending on   Socket/fallback

Created duid "\000\001\000\001*\262\013r\370\321\021\021\012\010".

DHCPDISCOVER on wlxf8d111110a08 to 255.255.255.255 port 67 interval 7

DHCPDISCOVER on wlxf8d111110a08 to 255.255.255.255 port 67 interval 12

DHCPOFFER of 10.19.33.131 from 10.19.33.1

DHCPREQUEST for 10.19.33.131 on wlxf8d111110a08 to 255.255.255.255 port 67

DHCPACK of 10.19.33.131 from 10.19.33.1

bound to 10.19.33.131 -- renewal in 18927 seconds.

La commande iw permet d'obtenir des détails sur la connexion qui, à cet instant, est particulièrement asymétrique (1Mb/s en réception, 54Mb/s en émission) :

=> iw wlxf8d111110a08 link

Connected to b8:27:eb:a3:de:37 (on wlxf8d111110a08)

SSID: Brelis

freq: 2447

RX: 808044 bytes (10706 packets)

TX: 14300 bytes (115 packets)

signal: -35 dBm

rx bitrate: 1.0 MBit/s

tx bitrate: 54.0 MBit/s

bss flags:  short-slot-time

dtim period:  2

beacon int:  100

Si le point d'accès n'est plus disponible, alors la même commande l'indique :

=> iw wlxf8d111110a08 link

Not connected.

Nous avons ici un moyen de détecter la perte de connexion WiFi. Mon NetworkManager perd irrégulièrement la connexion et refuse de se reconnecter sans avoir la politesse de me fournir une raison. Une autre façon de détecter la perte de réseau consiste à connaître la route par défaut et effectuer un ping dessus. En cas d'échec, on peut demander une reconnexion.

## 4. Reprenez le contrôle de votre connexion

Nous allons voir ici comment reprendre le contrôle de la connexion. Nous ne montrerons pas un programme qui serait capable de choisir la meilleure configuration. Il s'agit simplement de voir comment scripter le comportement d'une interface. Ces scripts ne sont pas aboutis, car c'est à chacun de programmer le comportement désiré. Parmi les scénarios possibles, voici quelques-uns que nous avons eus à gérer :

- Sans que l'ordinateur ne se déplace, le WiFi se déconnecte et ne se reconnecte pas automatiquement.
- L'ordinateur se connecte directement à un point d'accès sans chiffrement. Mais, il faudra ensuite entrer les identifiants dans une page web. La connexion automatique échoue donc.
- Dans le cas d'un mobile en déplacement, la position GPS permet de prédire l'entrée dans une zone autorisée. Il faut donc se connecter à un réseau dès qu'il apparaît et ne pas choisir le point d'accès qui va bientôt disparaître.

Il faut donc pouvoir programmer un comportement spécifique à partir des quelques exemples que nous présentons.

## 4.1 Coquillez !

La programmation de script se fait classiquement avec le shell (coquille). Nous allons donc présenter comment manipuler le WiFi avec le shell.

Pour commencer, il faut vérifier que nous n'avons qu'une seule interface. Pour cela, nous allons les énumérer. Nous réalisons le script suivant :

```
#! /bin/sh

NBIF=`iw dev | grep "^phy" | wc -l`

echo $NBIF interfaces WiFi
```

La commande iw dev produit le résultat suivant après ajout d'une seconde interface, nous avons compté le nombre de fois que phy apparaît en début de ligne spécifié par le caractère circonflexe :

```
iw dev

phy#1

    Interface wlxf8d1111109fe

        ifindex 6
```

```
        wdev 0x100000001
        addr f8:d1:11:11:09:fe
        type managed
        txpower 0.00 dBm
phy#0
    Interface wlxf8d111110a08
        ifindex 3
        wdev 0x1
        addr f8:d1:11:11:0a:08
        type managed
        txpower 20.00 dBm
```

Nous supposerons ensuite que nous n'avons qu'une seule interface. Sinon, il faudrait décider du comportement à adopter.

Nous allons maintenant réaliser un petit script qui choisit le meilleur point d'accès selon mes critères. J'ai donc fait la liste des points d'accès auxquels je pouvais me connecter. Je les ai classés dans un ordre croissant. D'abord, ceux qui me plaisent le moins, ensuite les meilleurs. La liste contient tous les sites sur lesquels j'ai l'habitude d'aller.

```
#! /bin/sh
# On réalise un scan, s'il échoue on abandonne
# on ne garde que les informations sur le SSID et la puissance du signal
SCANW=`iw dev wlxf8d111110a08 scan | grep -e SSID -e signal` || exit $?
# echo $SCANW
```

```
SSIDLIST="garage foo bar Brelis gee"

for ssid in `echo $SSIDLIST`;

do

   echo teste $ssid

   echo $SCANW | grep $ssid && BSTSSID=$ssid

   done

echo Best: $BSTSSID

ifdown wlxf8d111110a08

echo ifup wlxf8d111110a08=$BSTSSID

ifup wlxf8d111110a08=$BSTSSID
```

Nous commençons par un *shebang* (#! /bin/sh) qui indique que c'est un script shell.

Nous avons inclus dans le script le nom de l'interface. À chacun de voir s'il veut rajouter du code pour identifier une interface WiFi ou passer le nom de l'interface en paramètre ou laisser en dur dans le code.

Le scan commence classiquement par iw...scan. Nous ne gardons que les informations de signal (puissance) et le SSID. Le scan échoue souvent, donc nous ne continuons pas le script lors d'un échec.

La variable SSIDLIST contient les réseaux de ma maison (*garage* et *Brelis*) et ceux du travail (*foo*, *bar* et *gee*). La boucle for teste pour chaque SSID s'il a été détecté. Si je suis à la maison, les points d'accès *foo*, *bar* et *gee* ne seront pas détectés. La variable BSTSSID prendra la valeur *garage* qui sera ensuite écrasée par *Brelis*, le réseau que je souhaite utiliser.

Pour terminer, il faut déconnecter le réseau (ifdown), puis tenter de se reconnecter avec le bon réseau (le nom *Brelis* remplace le précédent

*Wbrelis*). Bien entendu, si le réseau est déjà le bon, on ne lancera pas le script. L'exécution produit le résultat suivant :

teste garage

signal: -77.00 dBm SSID: garage ...

teste foo

teste bar

teste Brelis

... signal: -35.00 dBm SSID: Brelis

teste gee

Best: Brelis

Killed old client process

Internet Systems Consortium DHCP Client 4.4.1

...

DHCPDISCOVER on wlxf8d111110a08 to 255.255.255.255 port 67 interval 9

DHCPOFFER of 10.19.33.131 from 10.19.33.1

DHCPREQUEST for 10.19.33.131 on wlxf8d111110a08 to 255.255.255.255 port 67

DHCPACK of 10.19.33.131 from 10.19.33.1

bound to 10.19.33.131 -- renewal in 16585 seconds.

Voici quelques lignes de shell qui permettent d'automatiser la gestion du WiFi. Le langage python est aussi de plus en plus utilisé pour les scripts d'administration.

## 4.2 Pythonez !

Nous allons maintenant jouer avec notre reptile. La gestion des interfaces réseaux est assez avancée, mais le support du WiFi reste plus limité (ou je n'ai pas trouvé la bonne librairie).

La librairie python pyroute2 (paquet python3-pyroute2) fournit des commandes pour gérer les interfaces réseaux. La documentation (paquet python-pyroute2-doc) fournit quelques exemples utiles.

Le programme nl80211_interfaces.py permet de lister les interfaces WiFi :

# python /.../nl80211_interfaces.py

6 phy3 wlxf8d1111107cf f8:d1:11:11:07:cf

5 phy2 wlxf8d1111103d5 f8:d1:11:11:03:d5

4 phy0 wlxf8d1111103c9 f8:d1:11:11:03:c9

Un autre programme permet de scanner les réseaux, la connexion doit être établie pour que python affiche le scan :

python /.../nl80211_scan_dump.py wlxf8d1111103c9

BSS b8:27:eb:34:90:38

TSF: 8634188309 (2:23:54.188309)

freq: 2412

capability: ESS Privacy SpectrumMgmt ShortSlotTime

signal: -47.0 dBm

last seen: 6388 ms ago

SSID: Daisuki

BSS xy:56:ef:d4:23:12

TSF: 2880161766839 (33 days, 8:02:41.766839)

freq: 2437

capability: ESS Privacy ShortPreamble ShortSlotTime

signal: -81.0 dBm

last seen: 29004 ms ago

SSID: eduroam

...

Pour la partie réseau, nous pouvons utiliser le paquet python3-netifaces. Nous pouvons l'utiliser en mode interactif pour trouver les informations qui nous intéressent. Il faut importer le module d'abord :

# python3

Python 3.9.2 (default, Feb 28 2021, 17:03:44)

[GCC 10.2.1 20210110] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import netifaces

>>> netifaces.interfaces()

['lo', 'enp0s31f6', 'kvmtap0', 'wlxf8d1111103c9', 'wlxf8d1111107cf']

Sur cet ordinateur, nous avons donc obtenu la liste de ses interfaces : la *loopback*, la carte Ethernet, l'interface virtuelle et deux interfaces WiFi. Plutôt que chercher une interface dont le nom commence par *wl*, nous pouvons utiliser l'exemple qui permet d'identifier si une carte dispose des extensions WiFi. Nous pouvons alors chercher les adresses réseau de nos cartes WiFi :

```
>>> netifaces.ifaddresses('wlxf8d1111103c9')

{17: [{'addr': 'f8:d1:11:11:03:c9', 'broadcast': 'ff:ff:ff:ff:ff:ff'}],

2: [{'addr': '10.163.1.129', 'netmask': '255.255.255.0', 'broadcast':
'10.163.1.255'}],

10: [{'addr': 'fe80::fad1:11ff:fe11:3c9%wlxf8d1111103c9', 'netmask':
'ffff:ffff:ffff:ffff::/64'}]}

>>> netifaces.ifaddresses('wlxf8d1111107cf')

{17: [{'addr': 'f8:d1:11:11:07:cf', 'broadcast': 'ff:ff:ff:ff:ff:ff'}]}
```

Une seule carte est active. Nous pouvons voir les paramètres réseau avec l'adresse IPv4. Pour continuer à évaluer le réseau, nous allons demander les routeurs qu'il connaît (*Gateways*) :

```
>>> netifaces.gateways()

{ 'default': {2: ('10.33.105.250', 'enp0s31f6')},

  2: [('10.33.105.250', 'enp0s31f6', True)]}
```

Puisque nous avons l'adresse du routeur par défaut, il est possible de vérifier s'il est joignable en lui envoyant une requête ICMP par exemple. Si le routeur n'est pas joignable, alors il faut sans doute relancer le WiFi.

## Conclusion

Nous avons présenté deux pistes de programmation en shell et Python, qui permettent d'adapter notre système pour obtenir le comportement souhaité, même si ce n'est pas la façon prévue par le vendeur.

Le WiFi est un domaine où beaucoup d'informations sont masquées. Demandez à un vendeur si sa carte WiFi permet de diffuser simultanément plusieurs réseaux et il va vous regarder bizarrement. Le composant de la clef est parfois indiqué, mais le firmware doit autoriser

l'usage désiré.

Quand un dysfonctionnement existe, il est parfois aléatoire (mon WiFi se déconnecte parfois plusieurs fois par jour, parfois il fonctionne sans broncher plusieurs semaines). Ce n'est donc pas facile à corriger.

Que ce soit pour allumer un appareil depuis le réseau ou installer un serveur à base d'une carte Raspberry dans un coin, l'équipement doit pouvoir se reconnecter sur le WiFi sans intervention humaine.

## Références

**[1]** Site officiel de *Hardware for Linux* : https://linux-hardware.org/

**[2]** A. Février, « Mettons en place un point d'accès WiFi », *Linux Pratique HS n°40*, octobre 2017, https://connect-ed-diamond-com.lama.univ-amu.fr/Linux-Pratique/lphs-040/mettons-en-place-un-point-d-acces-wifi

**[3]** A. Février, « Utilisons plusieurs réseaux Wifi », *Linux Pratique HS n°40*, octobre 2017, https://connect-ed-diamond-com.lama.univ-amu.fr/Linux-Pratique/lphs-040/utilisons-plusieurs-reseaux-wifi

**[4]** A. Février, « Installez votre Hotspot Wifi d'entreprise », *Linux Pratique HS n°43*, octobre 2018, https://connect-ed-diamond-com.lama.univ-amu.fr/Linux-Pratique/lphs-043/installez-votre-hotspot-wifi-d-entreprise