Plateforme Wifi

—

Guide du développeur

E. Chaput

—

E. Bouttier

12 septembre 2012

Résumé

Ce document est une introduction au développement sur la plateforme wifi, en particulier les drivers wifi, afin d'avoir une prise en main rapide du code. Il fait suite au document sur l'utilisation de la plateforme wifi.

Table des matières

1 Introduction

Ce document est composé de quatres parties que l'on pourrait associer deux-à-deux.

Les deux premières parties s'intéressent à la génération du système fournit dans le guide l'utilisateur. La première partie s'appuie pour cela sur une suite de scripts automatisés alors que la deuxième partie détail pas-à-pas chaque étape. Ainsi, la deuxième partie peut être survolé si on opte pour une génération automatique du système. Cependant, elle contient des informations pouvant s'avérer utiles pour personnaliser son système.

Les deux dernières parties essayent de donner quelques informations sur l'architecture des drivers wifi ATH5K et ATH9K afin de faciliter leur modification à des fins expérimentales. Ils y est étudier l'effet du carrier-sens, physique et virtuel, ainsi que du backoff sur les performances des connexions en TCP et UDP.

2 Utilisation des scripts de génération automatisé

Afin de pouvoir régénéré le système le plus rapidement possible, des scripts automatisants cette étape ont été écrit. Vous pouvez les récupérer depuis le dépot GIT accessible sur GITORIOUS. Pour cloner le dépot avec git, on utilisera la commande suivante :

```
$ git clone git://gitorious.org/plateforme-wifi/autogen.git
```

Sinon, on pourra télécharger directement une archive tar.gz de la dernière version des fichiers du dépots à l'adresse suivante :
`https://gitorious.org/plateforme-wifi/autogen/archive-tarball/master`

Pour lancer la génération, on executera simplement :

```
$ make
```

La génération du système se fait en 7 étapes, effectué par les scripts S01 à S07. Les scripts S41 et S99 ne sont pas des scripts de génération du sytème, mais des scripts qui y seront intégré. Le Makefile crée des fichiers `.stamp_*` pour marqué chaque étape comme effectué. Il peut être nécessaire d'en supprimer pour relancer une étape (ou alors exécuter directement le script associé).

3 Génération du système avec Buildroot

3.1 Description de buildroot

Buildroot est une suite de scripts sous forme de Makefile qui permet facilement de générer son propre système Linux. Il se charge de la compilation d'une chaîne de compilation croisée afin de compiler le noyau, ainsi que tous les programmes nécessaires pour avoir un système utilisable. Il permet aussi la génération du fichier d'amorçage en PXE.

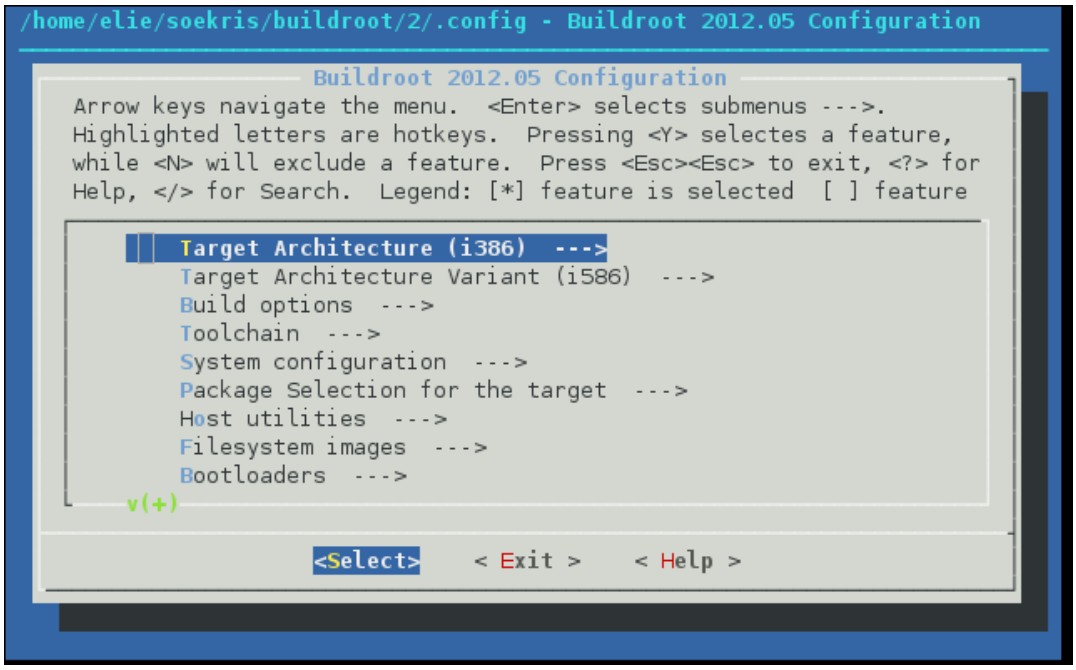
3.2 Obtenir buildroot

La dernière version de buildroot peut être trouvé sur la page `http://buildroot.uclibc.org/download.html`. Ce didactitiel a été réalisé avec la version 2012.05 de buildroot.

3.3 Configuration de buildroot

Buildroot possède une interface de configuration en curses accessible avec la commande suivante (à executer à la racine de l'archive décompressé) :

```
$ make menuconfig
```



3.3.1 Utiliser la configuration proposé

Il est possible de charger un fichier de configuration depuis le menu `Load an Alternate Configuration File`. Vous pouvez utiliser le fichier de configuration `buildroot-2012.05.config` fournit dans le dépot `autogen` (cf partie 1).

3.3.2 Utiliser une configuration personnalisé

Nous détaillerons dans cette partie les principales options qui ont permit de réaliser le système fournit en annexe.

On gardera l'architecture `i386` proposé, ainsi que la variante `i586`.

Dans le menu `Toolchain`, on pourra activer l'IPv6. Si vous souhaitez utiliser le `syslog` évolué et non celui inclue dans `busybox`, vous devez choisir `Enable large file (files > 2 GB) support`.

Dans le menu `System configuration`, on veillera à bien spécifier un baudrate de 19200 pour le port série. On pourra également personnaliser le nom d'hôte. Nous verrons ultérieurement comment définir un nom d'hôte auto-déterminé suivant la machine.

Dans le menu `Bootloader`, on choisira `syslinux` et on décochera `isolinux` pour ne garder que `pxelinux`. Cette étape est facultative si vous utilisez le fichier d'amorçage fournit en annexe.

Attention. Il se peut que la compilation échoue avec la version 4.04 de syslinux. On pourra essayer de passer à la version 4.05 en appliquand un patch disponible dans le dépot `autogen` puis en relançant la compilation :
`$ patch -p1 < syslinux-4.05.patch`

On activera la compilation du noyau dans le menu `Kernel`. Ceci est également facultatif si vous souhaité seulement généré la chaine de compilation croisé et utiliser le noyau fournit. Pour la configuration du noyau, on pourra utiliser le fichier de configuration `kernel.config` fournit en annexe. Il est également possible de commencer une nouvelle configuration en se basant sur un defconfig. On spécifira dans se cas `i386` pour son nom. Dans les deux cas, nous verrons par la suite comment modifier cette configuration.

Après l'activation de la compilation du noyau, on choisira `initramfs for initial ramdisk of linux kernel` dans le menu `Filesystem images`. On pourra également décocher `tar the root filesystem`, peut utile dans notre cas.

Dans le menu `Package Selection for the target`, on pourra sélectionner énormément de programme à compiler pour notre platforme. Busybox est un programme essentiel qui propose toutes les commandes de base (`cp, mv, udhcpc, ...`). L'option "Show packages that are also provided by busybox" permet par exemple de sélectionner le client/serveur DHCP de l'ISC. Nous sélectionnerons par exemple `bridge-utils`, `hostapd`, `iproute2`, `iptables`, `iw`, `openssh`, `tcpdump`, `wireless tools` et `wpa_supplicant`, toutes disponibles dans le menu `Networking applications`, ainsi que `vim` dans `Text editors and viewers` et `syslogd & klogd` dans `System tools`. `iperf` nécessite une chaîne de compilation croisée avec support du `C++` (plus lent).

3.4 Configuration du noyau Linux

Il est possible de personnaliser la configuration du noyau en se basant sur le defconfig précédemment sélectionné ou sur le fichier de configuration fournit. Le noyau possède également une interface de configuration en curses accessible par la commande :

```
$ make linux-menuconfig
```

On supposera dans la suite que nous sommes partie du defconfig i386. Nous allons pouvoir optimiser notre noyau en ne gardant que les drivers nécessaire à notre matériel.

On pourra jeter un coup d'oeuil dans le menu `Networking support`. Si vous comptez utiliser bridge-utils, pensez à activer le support du bridge dans le noyau (`Networkins support > Networking options > 802.1d Ethernet Bridging`).

Le menu `Device drivers` est sans doute celui où il y a le plus à faire.

- Dans le menu `Network device support > Ethernet driver support`, on pourra tous désactiver pour ne garder que le driver `DP8381x series PCI Ethernet support` disponible en cochant `National Semi-conductor devices`.
- De même, on pourra tous désactiver dans le menu `Network device support > Wireless LAN` pour ne garder que les drivers ATHEROS disponible dans le menu `Atheros Wireless Cards`.

– On vérifira que `netconsole` est bien supporté (`Network device support > Network core driver support > Network console logging support`). Cette option permet de reçevoir les messages du noyau au démarrage en UDP.

> **Remarque.** Il est préférable de compiler les drivers en dure dans le noyau pour gagner le temps de chargement (`[*]`). Cependant, pour travailler sur les drivers wifi, il est préférable de les compiler comme module (`[M]`) afin de pouvoir les décharger pour en charger d'autres.

Dans le menu `File systems`, nous pouvons décocher tous ce que nous pouvons. On pourra éventuellement laisser le NFS dans `Network File System` si besoin est.

On pourra décocher la `virtualisation`.

3.5 Personnalisation du système

3.5.1 Skeleton

Buildroot se base pour générer notre système sur un skeleton. Il s'agit d'une image du système de base avant installation des programmes compilés. Il est donc possible de personnaliser son système en modifiant le contenue du skeleton.

Pour cela, nous allons commencer par copier le skeleton fournit par buildroot disponible dans le dossier `fs/skeleton`. Nous spécifirons l'emplacement de notre copie dans le menu `System configuration` en choisissant `custom target skeleton` pour l'option `Root FS skeleton`.

> **Remarque.** Après une première compilation, le skeleton n'est plus utilisé. Vous pouvez tous de même faire des modifications en vue d'une autre compilation directement dans le dossier `output/target`.

Les scripts d'init se trouvent dans le dossier `/etc/init.d`, et sont lancé par ordre numérique. Vous pouvez donc ajouter vos propres scripts de démarrage.

Il est possible que vous souhaitiez rajouter vos propres applications dans votre système. La voie la plus simple est sans doute l'ajout d'un paquage dans buildroot. On pourra se renseigner sur la méthode à suivre depuis la documentation officielle : `http://buildroot.uclibc.org/downloads/manual/manual.html#_adding_new_packages_to_buildroot`.

3.5.2 Réseau

La configuration du réseau se fait classiquement par le fichier `/etc/network/interfaces`. Cependant, il est possible de spécifier la configuration directement dans la ligne de commande du noyau, ce qui permet de la modifier sans devoir régénérer votre noyau.

Pour obtenir une adresse IP par DHCP, il suffit de rajouter l'option suivante :

```
ip=dhcp
```

Pour configurer une IP statique, on pourra utiliser l'option suivante :

```
ip=192.168.1.1::192.168.1.254:255.255.255.0:net4826:eth0:off
```

où `192.168.1.1` est l'IP de la machine, `192.168.1.254` l'IP du routeur et `net4826` le nom d'hôte de la machine.

3.5.3 SSH

Si vous utilisez un serveur ssh, celui-ci génère au démarrage des clefs privées ce qui est très lent sur ces petites machines, et fera également apparaître des problèmes d'identification d'hôte entre deux démarrage. Il est donc préférable de les ajouter au skeleton. Vous pouvez utiliser les clefs fournit avec le guide de l'utilisateur :

```
$ cp /path/to/annexes/ssh_host_* fs/skeleton/etc/
```

Vous pouvez également télécharger les clefs par TFTP au démarrage comme c'est le cas dans le système fournit. Pour cela, vous pouvez utiliser le scripts `S41gettftpaddr` qui permet d'obtenir l'IP du serveur tftp (voir le guide de l'utilisateur), et patcher le script d'init `S50sshd` avec le patch `S50sshd.patch` pour qu'il ne génère les clefs que s'il n'a pas pu les télécharger :

```
$ patch -p1 < /path/to/annexes/S50sshd.patch
```

Remarque. Le patch modifie le fichier `package/openssh/S50sshd`. Si vous avez déjà lancé une première compilation de buildroot, copier le directement dans votre filesystem avant de relancer `make` :
`$ cp package/openssh/S50sshd output/target/etc/init.d/`

Par defaut, le serveur ssh est configurer pour refuser les connexions sans mot de passe, entre autre celle de l'utilisateur `root`. Il est donc nécessaire de modifier cette option dans le fichier `output/target/etc/sshd_config` *après une première génération de notre système* (sans quoi le fichier n'existerai pas encore). Il faut donc rajouter la ligne :

```
PermitEmptyPasswords yes
```

Le patch `sshd_config.patch` s'occupe de cette modification :

```
$ patch -p1 < /path/to/annexes/sshd_config.patch
```

Si vous souhaitez utiliser l'authentification par clef, aujouter votre clef public dans le fichier `/root/.ssh/authorized_keys` :

```
$ mkdir -p fs/skeleton/root/.ssh/authorized_keys
$ cat ~/id_rsa.pub >> fs/skeleton/root/.ssh/authorized_keys
```

Vous pouvez utiliser la clef id_soekris contenue dans le dépot `autogen` pour vous connecter au système fournit. Cependant, celle-ci étant publiquement disponible, elle peut introduire de grave problème de sécurité.

3.6 Génération du système

Pour lancer la génération de votre système précédement compilé, lancé tous simplement :

```
$ make
```

À la fin de la compilation, vous pourrez récupérer votre système dans le dossier `output/images`. Il contient généralement les fichiers `pxelinux.bin` et `bzImage` ainsi qu'un fichier `root.tar` si vous aviez demandé une archive tar du système de fichier.

Vous pouvez par la suite modifier des options dans le menuconfig et/ou modifier le système de fichier présent dans le dossier `output/target`. Relancez alors la commande `make` pour régénérer votre images. Dans le cas de la modification d'un paquage, pour que votre modification soit prise en compte, vous devrez préalablement le nettoyer :

```
$ make <package>-clean
```

Pour en savoir plus sur le fonctionnement de buildroot, on se reportera à la documentation officielle : `http://buildroot.uclibc.org/downloads/manual/manual.html#_how_buildroot_works`

4 Drivers wifi

Les douze Soekris de la plateforme wifi sont équipés de cartes `WLM200MX` munient du chipset Atheros `AR9220` et requérant le driver `ATH9K` pour fonctionner. Deux autres cartes `WLM54AG` utilisant le chipset `AR5414` et requirant le driver `ATH5K` sont également disponibles.

Anciennement, les cartes wifi Atheros étaient utilisées avec les drivers Madwifi. Ceux-ci étaient composés d'une couche `HAL`, fournit par `Atheros` et non open source, ainsi que d'une couche libre, utilisant les fonctions de la couche `HAL`. Leurs modifications étaient très limité dû à l'impossibilité d'accéder à la couche matérielle.

Depuis le noyau 2.6.25 environ, de nouveaux drivers entièrement open source, `ATH5K` et `ATH9K` sont présent directement dans les sources du noyau. Ils s'agit de ces drivers que nous étudirons, les drivers Madwifi étant dépréciés.

4.1 Récupération des sources

Les sources des drivers wifi Atheros sont disponible directement avec les sources du noyau linux, que vous pouvez récupérer sur `http://kernel.org`. Vous trouverez les drivers dans le dossier : `linux/drivers/net/wireless/ath`. Si vous avez utilisé buildroot, celui-ci a déjà récupéré les sources du noyau. Vous pouvez les récupérer dans le dossier `buildroot/output/build/linux-x.y.z/`.

4.2 Compilation des drivers

La compilation des drivers sous Linux est basé sur le Kernel Build System utilisant des Makefile un peu particulier. Le Makefile pour les drivers Atheros est présent dans le dossier `ath`. Pour lancer la compilation, exécutez la commande suivante :

```
$ make -C /path/to/linux/source/dir SUBDIRS=/path/to/module/source/dir modules
```

À partir du dossier `ath` dans buildroot, vous pouvez simplement lancer la commande :

```
$ make -C ../../../.. SUBDIRS="$(PWD)" modules
```

Vous pouvez également rajouter dans le Makefile présent dans le dossier `ath` les lignes suivantes (en fin de fichier) :

```
default:
        $(MAKE) -C /linux/source/dir SUBDIRS=/module/source/dir modules
```

Vous pouvez ainsi simplement lancer la commande `make` depuis le dossier `ath`, qui exécutera alors la cible `default`.

4.3 Modules et dépendances

Les drivers Madwifi étaient basé sur l'interface Wireless-Extention (WEXT) du noyau. Cependant, une nouvelle interface basé non plus sur `ioctl` mais sur des structures de callback, lib/mac/cfg80211, a vue le jour. Les nouveaux drivers s'appuient sur cette interface.

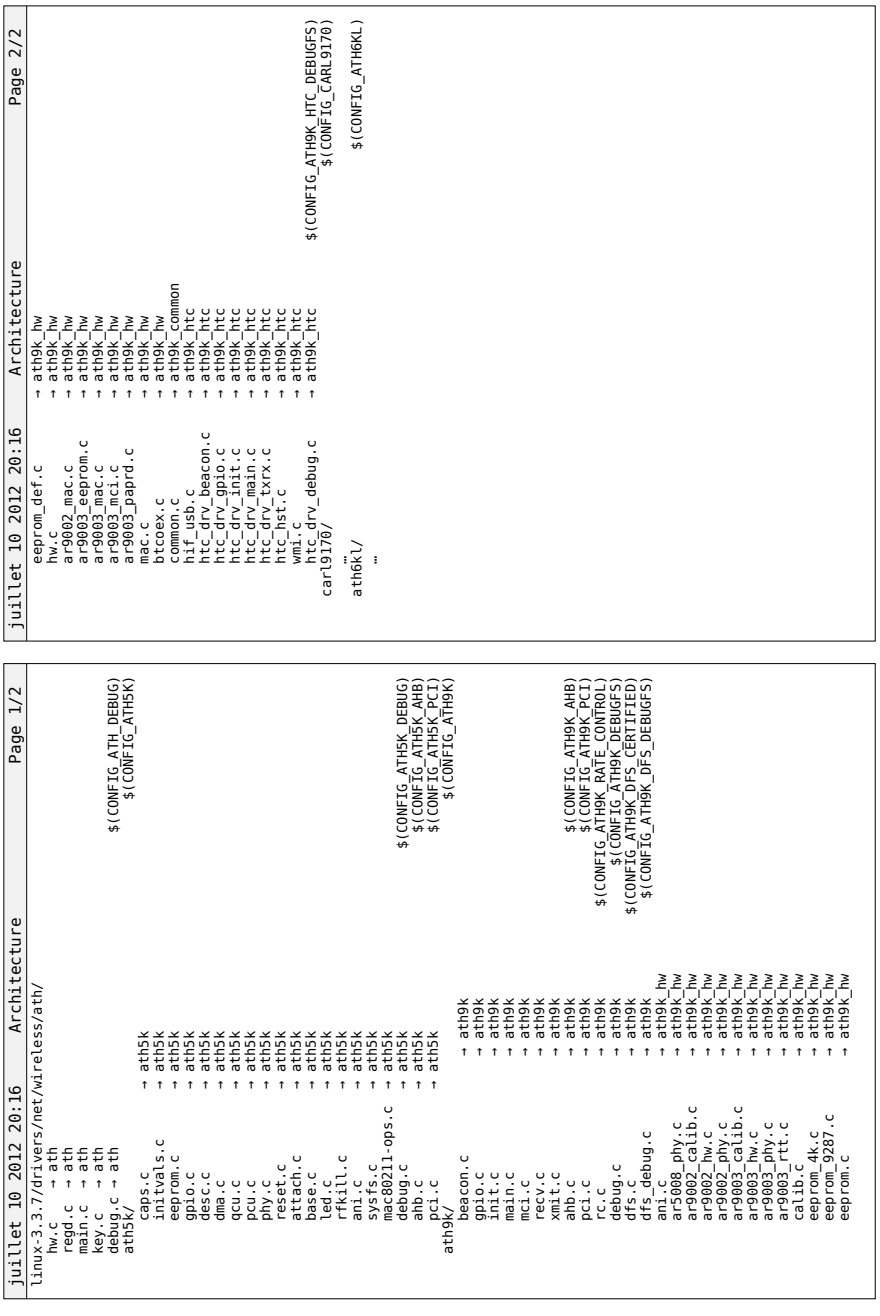
Les drivers Atheros `ATH5K` et `ATH9K` sont composé de cinq modules :

- `ath` : La partie commune aux modules `ath5k` et `ath9k` ;
- `ath5k` : le driver `ath5k` ;
- `ath9k` : le driver `ath9k` ;
- `ath9k_hw` : la partie hardware du driver `ath9k` ;
- `ath9k_common` : partie commune au driver `ath9k` et au driver `ath9k_htc` (le driver `htc` est utilisé pour deux chipset spécifique).

Après compilation des modules, il faut encore les charger en mémoire avec la commande `insmod <module.ko>`. Cependant, un ordre précis est à respecter afin de satisfaire les dépendances de chaque module. Celles-ci peuvent être obtenu à l'aide de la commande `modinfo <module.ko>`. Voici la liste des dépendances pour les modules qui nous intéressent :

- `lib80211` : ∅
- `cfg80211` : ∅
- `mac80211` : cfg80211
- `ath` : cfg80211
- `ath5k` : ath, mac80211, cfg80211
- `ath9k_hw` : ath
- `ath9k_common` : ath9k_hw, ath
- `ath9k` : ath9k_common, ath9k_hw, ath, mac80211, cfg80211

4.4 Architectures



Lorsqu'un fichier est modifié, il est utile de savoir après compilation quel module doit être

rechargé. Ce diagramme renseigne pour chaque fichier avec quel module il est linké (obtenue à partir du Makefile).

4.5 Debuggage

Des informations de débuggage sont disponible en compilant les drivers ATHEROS avec respectivement les options `CONFIG_ATH5K_DEBUG` et `CONFIG_ATH9K_DEBUGFS`.

Les informations de débuggage pour le driver `ath9k` sont disponible sous la forme d'entrées dans le debugfs. Pour y accéder, celui-ci doit d'abord être monté :

```
# mount -t debugfs none /sys/kernel/debug
```

Vous trouverez les entrées liées au driver `ath9k` dans le dossier `/sys/kernel/debug/ieee80211/phyX/ath9k/`. Les informations pour chaque entrée est disponible sur la page officielle de documentation à l'adresse : `http://linuxwireless.org/en/users/Drivers/ath9k/debug`.

L'entrée la plus importante est sans doute `debug`. En effet, celle-ci permet de modifier le filtre des messages envoyé vers syslog. Par exemple, on pourra afficher les messages de type `FATAL`, `BEACON` et `QUEUE` avec la commande `echo "0x502" > debug` (`cf` documentation pour les détails du masque).

4.5.1 Queues

Les trames, data, beacon ou autre, sont placées dans des queues. Celles-ci possèdent plusieurs paramètres, stoqué dans une structure `ath9k_tx_queue_info` (définie dans `mac.h`). Les fonctions `ath_txq_setup` et `ath_txq_update` (présentent dans `xmit.c`) permettent de les modifier. Afin de connaître les paramètres actuels des différentes queues, il est possible de rajouter une entrée `tx_queue_info` dans le système de fichiers debugfs à l'aide du patch `debugfs_tx_queue_info.patch` :

```
ath/ath9k$ patch -p1 < debugfs_tx_queue_info.patch
```

4.6 Tests avec débit

Afin d'étudier l'influence des modifications effectuées, il est intéressant de faire des tests de débit à l'aide d'`iperf`. Celui-ci fonctionne en mode client/serveur et effectue un test de bande passante du client vers le serveur.

Pour lancer le serveur sur un soekris, lancez simplement `iperf -s`. Pour connecter un client au serveur, lancez `iperf -c A.B.C.D`. Vous pouvez spécifier la durée d'un test avec l'option `-t` (par défaut, 10 secondes).

Par défaut, les tests sont effectué en TCP. Utilisez l'option `-u` pour effectuer un test en UDP (le serveur également doit être lancé avec cette option). Vous pouvez spécifier le débit de la transmission avec l'option `-b`, par exemple : `iperf -c A.B.C.D -u -b 100M`.

Il est intéressant de coupler l'utilisation d'`iperf` avec l'observation du fichier `/sys/kernel/debug/ieee80211/phyX/ath9k/xmit` (ainsi que `recv`).

4.7 Paramètres de module

Il est possible de passer des paramètres à un module à son chargement. Pour cela, il suffit de les passer en argument à la commande `insermod` :

```
# insmod module.ko param=value
```

On trouve facilement de nombreuse information à ce sujet sur internet, par exemple `http://www.makelinux.net/books/lkd2/ch16lev1sec6`.

Voici tous de même un exemple pour se faire une idée : rajoutez les lignes suivantes dans les premières lignes du fichier `ath/ath9k/init.c`.

```
char * modparam_name = "unnamed";
module_param_named(name, modparam_name, charp, 0644);
MODULE_PARM_DESC(name, "Test parameter");
```

Ceci a pour effet de déclarer une variable `modparam_name` donc la valeur pourra être spécifiée en utilisant la commande `insmod ath9k.ko name=monkey`.

Il est également possible de contrôler la valeur de cette variable, et de la modifier, et ce alors que le module est chargé, grâce au système de fichier `/sys`. En effet, les paramètres des modules apparaissent dans le dossiers `/sys/modules/MODULE_NAME/parameters/`. Les permissions spécifiées dans la macro `module_param_named` s'appliquent au fichier présent dans ce dossier. Ceci constitue un moyen simple et rapide de mettre en place une entrée de débuggage dans le système de fichier au moyen de peu de ligne de code.

4.8 Envoyer des messages dans les log

Il est intéressant de pouvoir afficher des messages de débuggage dans notre driver (par exemple la valeur des paramètres reçu en argument). Pour cela, on se tournera vers la fonction `printk`. Elle s'utilise simplement :

```
printk(KERN_INFO "Value of my int : %d\n", my_int);
```

Les autres niveaux de priorité sont disponible dans la documentation de `printk`.

Si vous avez activé l'option `CONFIG_ATH_DEBUG` à la compilation, il est possible d'utiliser la macro `ath_dbg` dans le code source. Celle-ci a l'avantage de permettre de filtrer les messages de debug suivant leur catégorie. On la préfèrera donc à `printk`. Elle s'utiles de la manière suivante :

```
ath_dbg(common, QUEUE, "Format printf avec des %s\n", "variable");
```

Le deuxième argument renseigne la catégorie du message. La liste des options disponible peut être consultée ici :
`http://linuxwireless.org/en/users/Drivers/ath9k/debug`.

Il est possible de modifier cette macro pour afficher également le nom du fichier, la ligne et la fonction d'où l'appelle à `ath_dbg` a été effectué. On pourra pour cela appliquer le patch `ath_dbg.patch` dans le dossier `ath` :

```
$ patch -p1 < /path/to/ath_dbg.patch
```

On fera bien attention après recompilation à recharger *tous* les modules. En effet, certains pourraient continuer d'utiliser l'ancien prototype et créer ainsi des `kernel panic`.

4.9 Détails des fichiers

- `init.c` : Fonctions d'initialisations. On y trouve les points d'entrée et de sortie du module (déclaré par les macros `module_init` et `module_exit`). L'initialisation du module consiste en la déclaration auprès du système de la gestion de périphérique sur des bus AHB et PCI. La suite de l'initialisation n'a lieu qu'après un callback du système, informant le module de la présence d'un périphérique qu'il a déclaré gérer. Le hardware du périphérique est alors initialisé (enregistrement de différentes valeurs dans les registres) et le périphérique réseau est déclaré auprès du système. Cette déclaration se fait au travers de la surcouche `lib/mac/cfg80211` spécifique au drivers wifi.
- `pci.c` : Gére l'enregistrement auprès du système comme driver PCI. Ceci consiste à indiquer au système les périphériques géré par le driver en fournissant leur vendor-id et product-id. Si le système détecte la présence d'un périphérique concerné, le driver est prévenu par une fonction de callback.
- `ahb.c` : Équivalent de la partie PCI mais pour le bus AHB. Il s'agit d'un bus minimaliste, de type DMA (Direct Memory Access), que l'on trouve généralement dans les systèmes de type SoC (System On a Chip). Les démarches sont très semblables à la partie PCI (enregistrement comme gestionnaire d'une liste de périphériques, callback du système lors de la présence d'un de ceux-ci).
- `ani.h` : On y parle de CCK (Complementary Code Keying), d'OFDM (Orthogonal Frequency Division Multiplexing) et de MRC (Maximal Ratio Combining) (Système d'optimisation des performances par usage de deux antennes distinctes).
- `arXXXX` : Les fichiers préfixés par `ar` sont spécifique à certain chipset. Dans notre cas, nous nous intéresserons aux fichiers préfixé par `ar9002`, en charge de notre chipset `ar9220`.
- `btcoex.c` : Utile seulement pour les cartes wifi qui sont également des cartes bluetooth. On y gère alors la coexistance avec le bluetooth. Ceci ne nous intéresse pas dans le cadre de notre firmware.
- `beacon.c` : Gère l'envoit des trames balises. Ce fichier est intéressant car il modifie les paramètres de la queue d'envoit (nottament contention window).
- `mci.c` : Gestion de l'état du lien, sleeping mode, puissance du signal, ...
- `phy.h` : Quelques `define` lié à la couche physique (puissance, gain, ...)
- `rc.c` : Rate Control
- `reg.h` : Longue liste de `define` d'adresse de registre. Difficilement exploitable ... Cependant, on peut arrivé à deviner certaine définition comme étant l'adresse d'un registre, suivit de la définition des bits de ce registre (par exemple, la définition du registre `AR_D_BGL_IFS_MISC` suivit du bit `AR_D_GBL_IFS_MISC_IGNORE_BACKOFF`. Ces information peuvent par la suite être utilisées avec les macros `REG_SET_BITS` et `REG_CLR_BIT`. Celles-ci prennent en argument un pointeur sur une structure `ath_hw` (habituellement `ah`, ou `sc->sc_ah` où `sc` est un pointeur sur une structure `ath_softc`), un registre et un bit.
- `recv.c` : Fonctions de reception des trames wifi.
- `xmit.c` : Fonctions d'envoit des trames wifi.
- `ath9k.h` : Header for the ath9k.ko driver core *only* – hw code nor any other driver should rely on this file or its contents.
- `calib.c` : Calibration : Ajuste le NF (Noise Floor) en fonction du bruit.

- `common.c` : Quelques fonctions, peut utiles, commune à `ath9k` et `ath9k_htc`. Ce fichier constitue à lui seul le module `ath9k_common`. Cependant, ses points d'entrée et de sortie sont vide, ses fonctions sont simplement exportées.
- `dfs.c` : Compilé et linké au module `ath9k.ko` ssi `CONFIG_ATH9K_DFS_CERTIFIED` est définie. L'acronyme de ce module signifie Dynamic Frequency Change. L'objectif est de ne pas brouiller les radars en changant dynamiquement de fréquence lorsqu'un pulse radar est détecté sur la fréquence actuelle.
- `debug.c` : Les fonctions de débuggage sont activé ssi la variable `CONFIG_ATH9K_DEBUGFS` est définie. Les possibilités énoncé dans la partie sur le débuggage sont fournit par ce fichier.
- `dfs_debug.c` : Les fonctions de débuggage liées à la partie DFS, linké ssi la variable `CONFIG_ATH9K_DFS_DEBUGFS` est définie.
- `eeprom` : Gestion de l'eeprom. Celle-ci contient plusieurs information telles que le code du pays (utile au régulateur qui interdit certain channel suivant la réglementation en vigueur) ou l'adresse MAC.
- `gpio.c` : Gère :
  - Les LED ;
  - Les boutons rfkill (radio frequence kill - désactivation matériel du wifi) ;
  - La coexistance avec le bluetooth (btoex).
- `htc` : ath9k_htc provides hardware support for Atheros AR9001 and AR9002 family hardware.
- `hif_usb.c` : En lien avec le module `htc`.
- `hw.c` : Fonction hardware. Le fichier est divisé en plusieurs section :
  - Helper Functions
  - Chip Revisions
  - HW Attach, Detach, Init Routines
  - INI
  - Reset and Channel Switching Routines
  - Power Management (Chipset)
  - Beacon Handling
  - HW Capabilities
  - GPIO / RFKILL / Antenne
  - General Operation
  - HTC
- `hw-ops.h` : Courtes fonctions `inline` de paramétrage.
- `wmi.c` : Wireless Module Interface

4.10 Séquence de démarrage

Le fichier `ath9k_init.c` contient un résumé des principales fonctions appelées lors du chargement du module.

4.11 Structures de données courtantes

La structure la plus courament véhiculé est la structure `ath_softc`, appelé couramment `sc`. Elle est définie dans le fichier `ath9k.h`. Beaucoup d'autre structure utilisé sont stoqué

dans la structure `ath_softc`. Par exemple, la structure `ath_hw`, courament appelé `ah`, s'obtient par `sc->sc_ah`.

Une structure très intéressante est la structure `ieee80211_ops`. Le driver `ath9k` en définie une dans le fichier `main.c` à la ligne 2433. Celle-ci contient toutes les fonctions de callback qu'il est enregistré auprès de la sous-couche `lib/cfg/mac80211`. Par exemple, le champ `.tx` pointe sur la fonction `ath_tx`, qui est la fonction du driver qui est appelé lorsqu'une trame est à transmettre. Vous pouvez obtenir les informations sur chaque champ de cette structure dans la documentation de la sous-couche `802.11` consultable en ligne : `linuxwireless.org/80211books/`.

Cette structure est utilisé en argument de la fonction `ieee80211_alloc_hw` dans la fonction `ath_pci_probe`, appelé lorsqu'un périphérique pci est géré par le driver :

```
ieee80211_alloc_hw(sizeof(struct ath_softc), &ath9k_ops);
```

5 Désactivation du backoff et carrier-sens

5.1 Macros

Vous avez 3 macros à votre disposition pour désactiver le backoff et le carrier-sens :

```
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH); // phy carrier-sens
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS); // virtual carrier-sens
REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Il peut être intéressant de consulter dans le fichier `ath9k/reg.h` les autres bits des registres `AR_D_GBL_IFS_MISC` et `AR_DIAG_SW` pour en étudier l'influence (par exemple, `AR_D_GBL_IFS_MISC_TURBO_MODE`).

5.2 Tests

Ces tests ont été effectués en mode ad-hoc à l'aide de trois stations : l'une exécutant `iperf` en mode serveur et les deux autres exécutants `iperf` en mode client (iperf test la bande passante du client vers le serveur). Le premier client était munie du drivers wifi `ath9k` non modifié tandis que le deuxième était munie premièrement du driver non modifié puis par la suite de plusieurs versions du driver modifié. Pour chaque version du driver, le débit a était testé en TCP et UDP, tout d'abord seul, puis avec le client au driver non modifié en concurrence. Les tests ont été fait sur une durée de une minute. Pour les tests en TCP, il est donné le débit moyen tandis que pour les tests en UDP, il est donné le débit d'émission (maximal de la carte réseau, avec les paramètres du driver wifi utilisé), le débit de réception et le pourcentage de paquets perdus. Pour les tests en concurrences, il est indiqué sur la première ligne les résultats pour le driver non modifié.

5.2.1 Test 1

Les deux clients sont munies du driver non modifié.

Client seul	
TCP UDP	11.2 Mb/s 12.1 Mb/s 12.1 Mb/s 0%

Clients en concurrence		
	TCP	UDP
TCP	5.77 Mb/s 5.67 Mb/s	3.82Mb/s 12.5 Mb/s 12.3Mb/s 1.7%
UDP	12.5 Mb/s 12.3Mb/s 1.7% 3.82Mb/s	11.2 Mb/s 8.45 Mb/s 24% 11.8 Mb/s 8.64 Mb/s 26%

5.2.2 Test 2

Le deuxième driver a été modifié par l'ajout des trois lignes suivantes dans la fonction `ath_txq_schedule` (présente dans `xmit.c`).

```
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH);
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS);
REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Client seul	
TCP UDP	3.81 Mb/s 11.9 Mb/s 11.9 Mb/s 0%

Clients en concurrence		
	TCP	UDP
TCP	7.22 Mb/s 3.82 Mb/s	452 Kb/s 12.8 Mb/s 12.8Mb/s 0.11%
UDP	9.46 Mb/s 3.30 Mb/s 65% 960 Kb/s	1.62 Mb/s 531 Kb/s 67% 12.4 Mb/s 12.4 Mb/s 0.03%

5.2.3 Test 3

Il s'agit du même test que le test 2, mais cette fois-ci les deux clients possèdent un driver modifié.

Clients en concurrence		
	TCP	UDP
TCP	1.02 Mb/s 1.52 Mb/s	34.3 Kb/s 10.2 Mb/s 9.57Mb/s 5.5%
UDP	10.2 Mb/s 9.57 Mb/s 5.5% 34.3 Kb/s	1.80 Mb/s 1.25 Mb/s 25% 6.92 Mb/s 49.2 Kb/s 99%

5.2.4 Test 4

Le driver du premier client n'est pas modifié alors que le driver du deuxième client a été modifié par l'ajout des lignes suivantes dans la fonction `ath_txq_schedule` (présente dans `xmit.c`) :

```
//REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH);
//REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS);
REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Client seul	
TCP UDP	955 Kb/s 11.9 Mb/s 11.9 Mb/s 0%

Clients en concurrence		
	TCP	UDP
TCP	5.80 Mb/s 1.24 Mb/s	3.22 Mb/s 11.9 Mb/s 11.9 Mb/s 0.28%
UDP	11.8 Mb/s 10.8 Mb/s 7.4% 1.02 Mb/s	4.49 Mb/s 4.45 Mb/s 0.62% 12.1 Mb/s 12.0 Mb/s 0.16%

5.2.5 Test 5

Le driver du premier client n'est pas modifié alors que le driver du deuxième client a été modifié par l'ajout des lignes suivantes dans la fonction `ath_txq_schedule` (présente dans `xmit.c`) :

```
//REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH);
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS);
//REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Client seul	
TCP UDP	10.8 Mb/s 11.6 Mb/s 11.6 Mb/s 0%

Clients en concurrence		
	TCP	UDP
TCP	5.52 Mb/s 5.43 Mb/s	3.72Mb/s 12.0 Mb/s 11.9 Mb/s 1.1%
UDP	11.8 Mb/s 11.1 5.2Mb/s % 3.95 Mb/s	11.8 Mb/s 6.60 Mb/s 44% 11.8 Mb/s 6.24 Mb/s 47%

5.2.6 Test 6

Le driver du premier client n'est pas modifié alors que le driver du deuxième client a été modifié par l'ajout des lignes suivantes dans la fonction `ath_txq_schedule` (présente dans `xmit.c`) :

```
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH);
//REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS);
//REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Client seul	
TCP	5.51 Mb/s
UDP	Mb/s Mb/s %

Clients en concurrence		
	TCP	UDP
TCP	Mb/s Mb/s	Mb/s Mb/s Mb/s %
UDP	Mb/s Mb/s % Mb/s	Mb/s Mb/s % Mb/s Mb/s %

5.2.7 Test 7

Le driver du premier client n'est pas modifié alors que le driver du deuxième client a été modifié par l'ajout des lignes suivantes dans la fonction `ath_txq_schedule` (présente dans `xmit.c`) :

```
REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_FORCE_CH_IDLE_HIGH);
//REG_SET_BIT(ah, AR_DIAG_SW, AR_DIAG_IGNORE_VIRT_CS);
//REG_SET_BIT(ah, AR_D_GBL_IFS_MISC, AR_D_GBL_IFS_MISC_IGNORE_BACKOFF);
```

Client seul	
TCP	5.51 Mb/s
UDP	11.8 Mb/s 11.8 Mb/s 0%

Clients en concurrence		
	TCP	UDP
TCP	6.79 Mb/s 3.67 Mb/s	1.43 Mb/s 12.1 Mb/s 12.0 Mb/s 0.5%
UDP	8.39 Mb/s 3.03 Mb/s 63% 2.13 Mb/s	1.92 Mb/s 1.90 Mb/s 1.1% 9.94 Mb/s 9.91 Mb/s 0.2%

5.3 Remarques divers

– Si les résultats sont incohérants, même avec seulement certains clients, tester de redémarrer le serveur `iperf` ...