Un routeur WIFI sous Linux

Pierre Ficheux (pierre.ficheux@openwide.fr)

Mars 2006

Résumé

Cet article décrit la mise en oeuvre d'un routeur WIFI sous Linux et utilisant une architecture compatible x86 (VIA C3). Le projet fut démarré en 2003 ce qui explique les choix techniques qui peuvent aujourd'hui paraître quelque peu désuets (noyau 2.4.20, pas de Busybox, etc.). Il est bien évident qu'une architecture plus récente conduirait au même résultat sans pour cela changer réellement la démonstration.

La projet est totalement viable puisque qu'il a conduit à un système utilisé dans un environnement domestique 24h/24H depuis 3 ans sans aucun redémarrage, mise à part les coupures de courant ou quelques évolutions du logiciel.

L'idée de départ

De nombreux utilisateurs expérimentés de Linux ont mis en place des fonctions de routage sur des architectures x86. Plusieurs projets sont dédiés depuis longtemps à de telles fonctions. Nous pouvons citer LRP (Linux Router Project) sur http://www.linuxrouter.org. L'idée du projet était non seulement de mettre un place un routeur WIFI domestique mais également de mettre en pratique les concepts décrits dans diverses publications: utilisation de PC industriel « réduit », construction de distribution embarquée, utilisation de mémoires flash spécifiques (Disk On Chip), « fiabilisation » d'un système en utilisant un *root filesystem* compressé et en lecture seule.

Mis à part le coté expérimental et pédagogique, une telle entreprise peut paraître aujourd'hui un peu vaine lorsque l'on sait que des produits comme LINKSYS fournissent des routeurs complets (sous Linux) très bien conçus pour moins de 100 €. Cependant le choix d'un produit maison peut s'avérer très intéressant dans le cas ou le projet demande des fonctions ou évolutions spécifiques qui ne sont pas forcément accessibles au travers d'un produit commercial.

La plate-forme cible

Le choix s'est porté sur une architecture compatible x86 (VIA C3) au format Mini-ITX. La plate-forme a l'avantage de disposer de plusieurs composants matériels très intéressants. Elle est décrite sous la référence STC-NET sur le site de la société SILINK, soit http://www.silink.fr.

- 3 interfaces ethernet
- Support pour disque IDE 2.5 pouces et 3.5 pouces
- Support Compact Flash IDE
- Support Disk on Chip ou DoC (M-Systems)
- Démarrage possible sur clé USB
- En option, contrôleur WIFI ATMEL sur bus USB (supporté par Linux)
- Alimentation 12V
- Dimensions réduites (boitier 22 cm x 16.5 cm x 4.9 cm)
- Fonctionnement silencieux

La figure ci-dessous représente la plate-forme cible.

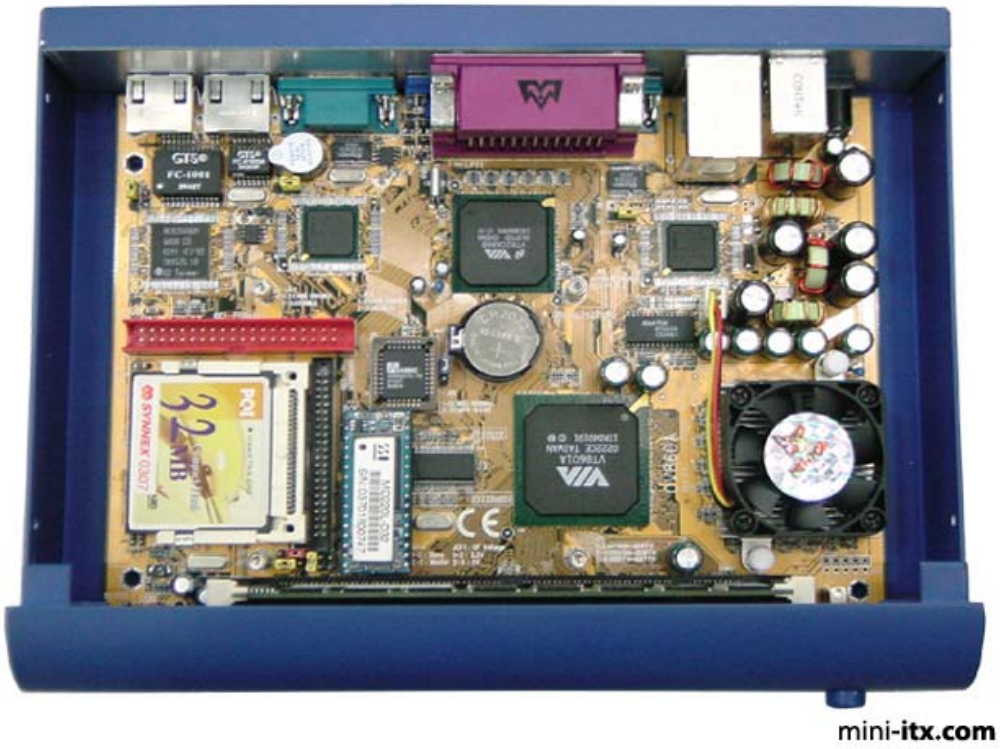

Figure 1: plate-forme cible STC-NET

Créer la distribution Linux embarqué

Le sujet est traité de manière approfondie dans les publications citées dans la bibliographie. En fait diverses solutions sont possibles.

1. Réduire une distribution classique (type Red Hat ou Mandrake)
2. Utiliser une distribution réduite existante, commerciale ou non
3. Construire la distribution à partir des composants (approche Linux From Scratch ou Busybox)

La première solution est complexe et ne donne en général pas de résultats satisfaisants car les distributions classiques ne sont pas conçues pour être réduites dans des proportions acceptables pour le sujet qui nous intéresse (soit une empreinte mémoire de quelques Mo). La deuxième est relativement simple à mettre en oeuvre et sera adaptée dans le cas de l'utilisation ponctuelle de Linux pour une application embarquée.

Dans les publications précédentes, nous avions choisi d'utiliser la troisième solution, soit de construire une distribution à partir de composants fondamentaux (noyau Linux, GLIBC, etc.). Cette approche nécessite un investissement intellectuel non négligeable mais elle a l'avantage d'être très pédagogique et d'assurer une parfaite maîtrise du résultat. De plus, une connaissance correcte de la structure du système Linux permet de mettre en place assez facilement un système de démonstration fournissant un noyau adapté et un environnement minimal fonctionnel (système minimal démarrant un interpréteur de commande type `bash`). A l'heure actuelle, l'utilisation de *Busybox* (http://www.busybox.net) est certainement la solution la plus rapide pour arriver à un système fonctionnel. Cependant, à l'époque du lancement du projet, Busybox n'était pas au niveau de performance d'aujourd'hui et nous avions alors choisi de construire la distribution à partir de composants standards. Les lecteurs intéressés par le sujet pourront se reporter à l'article « Busybox in a nutshell » paru dans Linux Magazine hors série numéro 24.

Mémoire de masse et système de fichiers principal

Dans un système Linux - réduit ou non - la mémoire de masse a deux fonctionnalités principales:

1. Héberger le système de fichiers principal (ou *root filesystem*) de Linux. L'existence de ce système de fichiers est indispensable au bon fonctionnement du système.
2. Le plus souvent permettre le démarrage du système en accueillant un programme spécialisé (comme LILO ou GRUB) dans le premier secteur du périphérique concerné.

Dans la majorité des cas, la mémoire de masse est constituée d'un disque dur, accessible à travers les fichiers spéciaux (ou *block devices*) `/dev/hdX` pour les disques IDE ou `/dev/sdX` pour les disques SCSI. La variable *X* pourra prendre les valeur a, b, c, d ou plus suivant la position du disque sur le bus concerné. Certains autres périphériques comme le DoC (choisi pour ce projet) utiliseront des pilotes différents mais qui conserveront leurs caractéristiques de périphériques en mode bloc et seront donc utilisables de la même manière pour les opérations habituelles sur les mémoires de masse.

- Le *partitionnement* avec l'outil `fdisk`
- Le *formatage* avec l'outil `mke2fs` (en cas d'utilisation d'un système de fichiers de type EXT3)
- Le *montage* avec `mount`

Concernant le type de système de fichiers, le choix d'un système de fichiers journalisé de type EXT3 est fortement conseillé sachant qu'un système embarqué est le plus souvent arrêté sur une coupure d'alimentation. Concernant l'intégrité des données, il y a deux niveaux à considérer :

- l'intégrité d'écriture d'un bloc : en cas de coupure, un bloc peut-il être partiellement écrit ou non ?
- l'intégrité d'écriture d'un fichier : en cas de coupure, un fichier peut-il être partiellement écrit ou non ?

Les mémoires Compact Flash ne garantissent pas le premier point mais par contre il est garanti dans le cas des DoC. Le deuxième point est intégralement dépendant du type de système de fichiers.

Cependant, le choix de tel ou tel système ne garantira pas la sécurité des données à 100%. Si le système nécessite une garantie absolue de fiabilité, il sera nécessaire de mettre en place une procédure matérielle de traitement de la coupure. Dans la suite de l'article, nous donnerons cependant un méthode permettant de minimiser les risques de perte de données en utilisant un système de fichiers principal monté en lecture seule.

Les mémoires Disk On Chip (DoC)

Le DoC est une mémoire FLASH intelligente (contenant un BIOS) développée par la société M-Systems (http://www.m-sys.com). Le figure suivante présente l'aspect du DoC.

Figure 2: Le Disk On Chip de M-Systems

Du fait de l'existence de ce BIOS, les DoC ont une durée de vie plus élevée que les CompactFlash ou les DoM IDE (Disk On Module), mais ils sont également plus onéreux. De même, ils sont plus difficile à trouver et sont de ce fait réservés à des applications professionnelles.

Utilisation du DoC sous Linux

Les DoC utilisent une interface propriétaire (ni IDE, ni SCSI) et leur utilisation nécessite une configuration spéciale du noyau Linux. Il y a deux méthodes possibles:

1. Utiliser le pilote propriétaire fourni par M-Systems (un *patch* du noyau)
2. Utiliser le pilote MTD du noyau Linux

Il faut noter que l'utilisation du pilote propriétaire interdit en toute rigueur la construction d'un noyau incluant le pilote en statique car ce pilote contient des parties non GPL fournies uniquement en binaire (il faut alors passer par un *initrd*). De ce fait, dans la suite du document nous utiliserons uniquement le pilote MTD. L'utilisation de ce pilote est décrite en détails au chapitre 7 de l'ouvrage *Linux Embarqué 2ème édition* mais nous allons rappeler brièvement la procédure.

La validation du support MTD (pour *Memory Technology Devices*) s'effectue dans le menu correspondant de la configuration du noyau Linux comme décrit sur les figures suivantes.

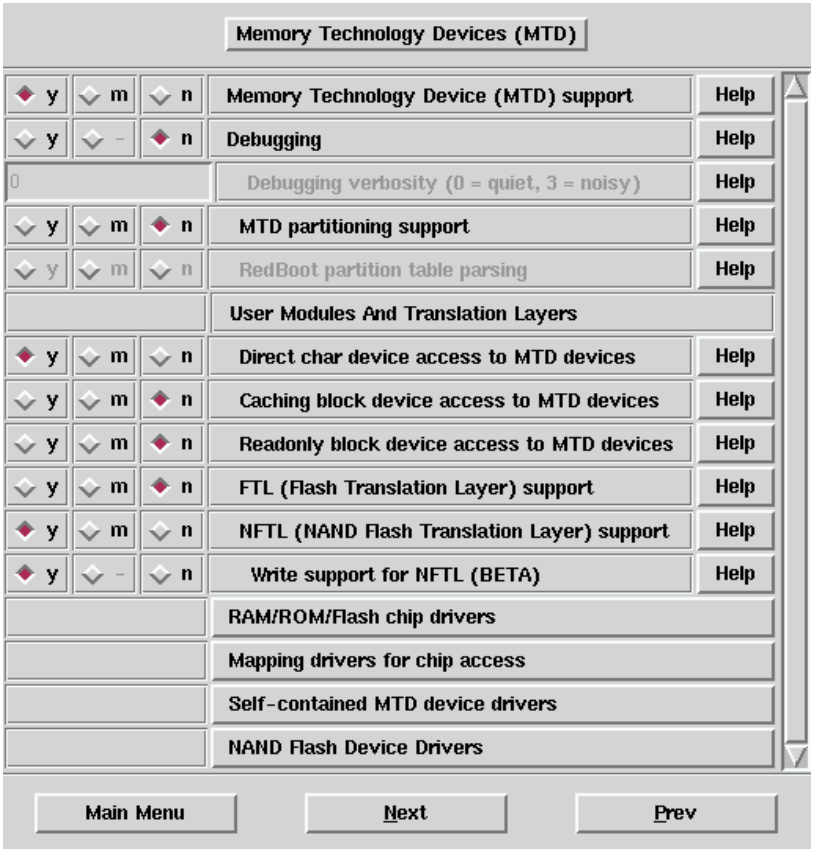

Figure 3: Validation du support MTD

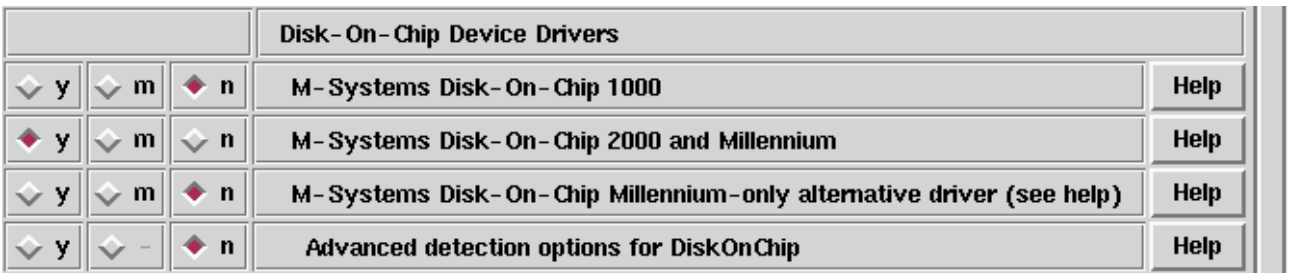

Figure 4: Validation du support DoC dans MTD

Le DoC est également accessible à travers des fichiers spéciaux en mode bloc mais il est nécessaire de les ajouter au système. Pour ce faire, il faut récupérer l'archive du projet MTD sur le site http://www.Linux-mtd.infradead.org. via le serveur CVS. Dans le répertoire `mtd/util` ainsi créé on trouve le script `MAKEDEV` permettant la création des points d'entrée sous `/dev`. Le nommage des fichiers est similaire à l'IDE ou au SCSI soit `/dev/nftla` pour le premier périphérique et `/dev/nftla1` pour la première partition de ce périphérique. On peut bien entendu utiliser les programmes classiques de partitionnement, formatage et montage sur ces fichiers spéciaux.
Suite à l'installation du noyau contenant le support MTD et au redémarrage du système la détection du DoC doit être visible dans les traces. On peut également vérifier sa présence grâce à la structure `/proc`.

```
# cat /proc/mtd
dev:  size        erasesize name
mtd0: 02000000    00004000  "DiskOnChip 2000"
```

Par défaut, les DoC sont également livrés avec un système de fichiers DOS, il est donc nécessaire de les formater pour l'utilisation en EXT3 ou autre système de fichiers Linux. La procédure est identique sauf que l'on utilise le fichier spécial `/dev/nftla`.

Démarrage à partir du DoC

Le DoC a une géométrie spéciale qui n'a rien à voir avec un disque IDE ou SCSI. De ce fait il faut utiliser une version modifiée de LILO pour pouvoir démarrer sur ce support. Cette version adaptée (soit `lilo-mtd`) est disponible dans le répertoire `patches` des sources MTD chargées à partir du site. Le fichier `lilo.conf` adapté est très similaire aux fichiers précédents.

```
boot=/dev/nftla
install=/boot/boot.b-mtd
map=/boot/map
read-only
vga = normal
# End LILO global section
image = /boot/bzImage-2.4.20_mtd
        root = /dev/nftla1
        label = Linux
```

Si la partition du Doc est montée sur `/mnt/doc`, on installe le secteur de démarrage par une syntaxe identique à celle `lilo`.

```
# lilo-mtd -r /mnt/doc -v
Patched LILO for M-Systems' DiskOnChip 2000
LILO version 21, Copyright 1992-1998 Werner Almesberger

Reading boot sector from /dev/nftla
Merging with /boot/boot.b-mtd
Boot image: /boot/bzImage-2.4.20_mtd
Added Linux *
Backup copy of boot sector in /boot/boot.5D00
```

```
Writing boot sector.
```

Utilisation de CRAMFS

Dans les paragraphes précédents, nous avons envisagé la construction de notre système dans une architecture classique, soit le système de fichiers principal monté en lecture/écriture et utilisant un format de système de fichiers journalisé (EXT3, JFFS2 ou autre). Cette structure est simple à mettre en oeuvre mais le système n'est pas à l'abri d'un incident vu qu'il sera la plupart du temps arrêté sur une coupure d'alimentation. De ce fait le système de fichiers peut être endommagé et donc le redémarrage parfois rendu impossible.

Principe de l'architecture

Si l'on observe la structure d'un système Linux, on s'aperçoit qu'une grande majorité du système de fichiers peut être configuré en lecture seule. Mises à part quelques exceptions que nous décrirons plus loin, les parties nécessitant l'accès en lecture écriture sont les suivantes.

- Le répertoire `/var` contenant des données de fonctionnement souvent volatiles
- Le répertoire `/tmp` encore plus volatile!
- Les répertoires de configuration (exemple: définition d'adresse IP, etc.).
- Les répertoire d'utilisateurs ou applicatifs (exemple: stockage de données enregistrées par le système). Ces derniers pourront être placés sur un véritable disque dur et il est également possible de monter la partition à la demande afin de limiter les risques.

Outre la dernière catégorie que nous ne traiterons pas ici, il est donc relativement simple de mettre en place l'architecture suivante:

- La majorité du système est sur une partition en lecture seule.
- Le répertoire `/var` (point de montage) est sur un disque mémoire.
- Le répertoire `/tmp` est un lien symbolique sur `/var/tmp`.
- Le répertoire de configuration utilisera un format classique (EXT2, EXT3). La partition est de très faible taille. On peut même imaginer de stocker cette configuration sur une partition non formatée (au format TAR directement écrit sur le fichier spécial de la partition) ce qui limite les risques de problèmes de système de fichiers dus à la coupure d'alimentation.

Mise en place de l'architecture système du routeur

Au niveau de l'implémentation nous supposerons que nous utilisons un DoC de 8 Mo. Le principe est d'utiliser 3 partitions sur la flash.

- Un première partition `/dev/nftla1` correspondant à `/boot` et contenant le noyau et les composants de LILO. Il n'est pas nécessaire que cette partition soit montée lors du fonctionnement du système mais seulement lors de la mise à jour éventuelle du noyau depuis un environnement de développement. De ce fait, cette partition sera formatée en EXT2.
- Une deuxième partition `/dev/nftla2` contenant le système de fichiers. Elle sera formatée en CRAMFS comme décrit plus loin.
- Une troisième partition `/dev/nftla3` contenant les quelques fichiers de configuration. Elle est de très petite taille et utilise en première approche un formatage EXT2.

Le fichier `/etc/fstab` décrivant les montages sur le système cible aura l'allure suivante.

```
bash-2.05# cat /etc/fstab
/dev/nftla2     /                       cramfs  defaults        1       1
/dev/nftla3     /data                   ext2    defaults        0       0
none            /dev/pts                devpts  mode=622        0       0
none            /proc                   proc    noauto          0       0
/dev/ram0       /var                    ext2    defaults        0       0
```

On note que la partition `/boot` n'est pas montée mais que nous avons une partition `/var` montée sur un disque mémoire `/dev/ram0`.

Pour la deuxième partition (système de fichiers principal), nous utiliserons le système de fichiers CRAMFS. Ce dernier est présent dans l'arborescence du noyau Linux standard. C'est un système de fichiers compressé (gzip), limité à 256 Mo, chaque fichier étant limité à 16 Mo. Il existe une petite documentation dans les sources du noyau (répertoire `Documentation/filesystems/cramfs.txt`). Pour utiliser CRAMFS, il faut disposer du programme `mkcramfs`. La version livrée avec certaines distributions ne fonctionne pas forcément très bien et il vaut mieux récupérer la version officielle disponible sur le site du projet soit http://sourceforge.net/projects/cramfs.

Pour construire une partition CRAMFS sur un support physique on devra tout d'abord créer l'image du répertoire au format CRAMFS.

```
# mkcramfs my_root my_root.img
Directory data: 14092 bytes
Everything: 4252 kilobytes
Super block: 76 bytes
CRC: 8647fdf8
```

On pourra ensuite copier l'image sur la partition par un simple `dd` ou un `cp`.

```
# dd < my_root.img > /dev/nftla2
```

On pourra également la monter sur le système pour vérification en utilisant l'interface *loopback*.

```
# mount -t cramfs -o loop root_dir.img /mnt/cramfs
```

Au niveau du noyau, il faudra bien entendu valider le support de CRAMFS en statique au niveau du menu *File systems* de la configuration du noyau. Puisque nous utilisons également un disque mémoire, il faudra valider le support *ramdisk* le menu *Block devices*. La taille par défaut de 4 Mo pour le disque mémoire est suffisante pour notre application. Les deux figures qui suivent indiquent les configurations à effectuer.

Figure 5: Validation du support CRAMFS

Figure 6: Validation du support RAMDISK

Si nous faisons référence aux différentes publications concernant la construction d'un système Linux embarqué, nous savons que le démarrage du système est divisé en 5 étapes.

1. Le démarrage du système par LILO (Linux LOader) ou un programme équivalent type GRUB

2. Le chargement du noyau

3. Le lancement par le noyau du processus `init` (soit `/sbin/init`)

4. lecture du fichier `/etc/inittab` par le processus `init`. Ce fichier contenant le nom du fichier de démarrage comme décrit ci-dessous.

```
# System initialization (runs when system boots).
si:S:sysinit:/etc/rc.d/rc.S
```

5. L'exécution du script ci-dessus

Sachant que le répertoire `/var` est placé dans un disque mémoire, il est nécessaire de construire l'arborescence de ce dernier à chaque démarrage du système. On aura donc dans le fichier `/etc/rc.d/rc.S` les lignes suivantes.

```
...
# Create /var (EXT2 filesystem)
/sbin/mke2fs -vm0 /dev/ram0 4096

# mount file systems in fstab (and create an entry for /)
# Mount /proc first to avoid warning about /etc/mtab
mount /proc
/bin/mount -at nonfs

# Populate /var
mkdir -p /var/tmp /var/log /var/lock /var/run /var/spool /var/lib/dhcp
/var/run/dhcpc
mkdir -p /var/cron/tabs /var/www/html /var/spool/cron /var/spool/mail /var/ppp
chmod a+rwx /var/tmp
...
```

Outre la création de `/var`, on notera le lien symbolique de `/etc/mtab` vers `/proc/mounts`. Ce lien est nécessaire car `/etc` n'est pas accessible en écriture.

```
lrwxrwxrwx    1 root     root           12 Jun 23  2003 mtab -> /proc/mounts
```

De même on notera l'utilisation fréquente des liens symboliques afin de permettre à des fichiers créés au démarrage d'apparaître dans le système de fichiers en lecture simple (voir exemple de `/etc/resolv.conf` ci-dessous). Les fichiers variables sont systématiquement placés dans le répertoire `/var/run`.

```
# ls -l /etc/resolv.conf
lrwxrwxrwx    1 root     root           20 Sep 24 07:53 /etc/resolv.conf ->
/var/run/resolv.conf
```

Concernant les fichiers de configuration, ils sont placés dans le répertoire `/data` associé à la troisième partition. L'arborescence est similaire à celle utilisée dans les précédentes publications (chapitre 5 de l'ouvrage *Linux embarqué 2ème édition*). Le principe est d'utiliser des variables d'environnement de l'interpréteur de commande BASH. Ces variables (comme l'adresse IP du routeur) sont stockées dans des fichiers au format texte dans le répertoire `/data/sysconfig`.

```
# ls -l /data/sysconfig/
total 6
-rw-r--r--    1 65534    nobody         41 Jun 23  2003 general
```

```
-rw-r--r--    1 65534    nobody        349 Sep 17 21:14 network
-rw-r--r--    1 root     root          174 Jun 23  2003 ppp
-rw-r--r--    1 root     root          150 Sep  6  2003 pppoe
-rw-r--r--    1 root     root           16 Jun 23  2003 syslogd.opts
-rw-r--r--    1 root     root          150 Jun 24  2003 wireless
```

Mise en place de la partie applicative (réseau)

Dans le cas présent, il est nécessaire d'initialiser une interface Ethernet et l'interface WIFI correspondant au contrôleur ATMEL connecté au bus USB. Le routeur doit également faire office de *firewall* en utilisant *NetFilter* (commande `iptables`). Il doit également disposer d'une fonction de serveur DHCP sur le domaine associé au réseau WIFI. Au niveau de l'implémentation, les démarrage des différents services s'effectue par des scripts exécutés par le script principal `rc.S` soit:

- `rc.inet1` pour le réseau Ethernet
- `rc.inet2` pour les démons
- `rc.wifi` pour l'interface WIFI
- `rc.firewall` pour le firewall (exécuté par `rc.wifi`)

Dans le cas présent, le routeur est connecté au réseau Ethernet 192.168.3.0 et utilise le réseau 192.168.4.0 pour les connexions WIFI.

Initialisation de l'interface Ethernet

L'interface est initialisée par l'appel au script `/etc/rc.d/rc.inet1` qui utilise le fichier de configuration `/data/sysconfig/network`. Dans le cas présent, le routeur dispose d'une adresse fixe sur le réseau local (Ethernet) soit 192.168.3.6 ce qui est indiqué par le type de protocole *None*. Il faut noter que le système est capable de gérer d'autres types de connexion (soit *PPP* et *PPPOE*) mais ce point n'a jamais été utilisé.

Le fichier décrit également les autres paramètres du réseau, tels les valeurs des DNS.

```
#PROTO=DHCP
PROTO=None

# IP parameters
HOSTNAME=pclight
DOMAINNAME=localdomain
DEVICE=eth0
IPADDR=192.168.3.6
GATEWAY=192.168.3.1
NETMASK=
NETWORK=
BROADCAST=
# Free
#DNS1=212.27.32.176
#DNS2=212.27.32.177
# Wanadoo
#DNS1=193.252.19.3
#DNS2=193.252.19.4

# Numericable
DNS1=85.68.0.8
DNS2=85.68.0.7
#DNS1=194.2.0.20
#DNS2=194.2.0.40

...
```

La fin du fichier concerne le script `/etc/rc.d/rc.inet2` chargé du démarrage des démons, soit dans notre cas `xinetd` et `syslogd`.

```
#
## Daemons
#
DAEMONS="xinetd syslogd"
```

Initialisation de l'interface WIFI

L'initialisation du WIFI (`rc.wifi`) est un peu plus délicate mais ne pose de pas problème particulier. Elle se décompose comme suit.

- Chargement du module de pilotage du contrôleur WIFI USB
- Initialisation de l'interface WIFI par la commande `iwconfig`
- Initialisation de l'adresse IP associée à l'interface WIFI par `ifconfig`

La suite concerne la mise en place des règles du firewall (NetFilter) et le lancement du serveur DHCP.

Dans le cas présent il existait deux pilotes disponibles pour le composant ATMEL. Le système donne le choix du pilote via le fichier de configuration `/data/sysconfig/wireless`.

```
# Wifi configuration
# Modules
#USBMOD=usb-uhci
USBMOD=uhci
WIFIMOD=usbvnetr
#WIFIMOD=at76c503-rfmd
# Interface Ethernet à router (pour firewall)
EDEVICE=eth0
# Mode WIFI
WMODE=ad-hoc
#  Identificateur du réseau WIFI
WESSID=mozart
# Adresse IP du routeur sur le réseau WIFI
WIPADDR=192.168.4.1
```

L'initialisation commence par le chargement des modules USB et contrôleur WIFI.

```
echo -n "Loading modules..."
modprobe $USBMOD
my_sleep 2
modprobe $WIFIMOD
my_sleep 2
```

On récupère ensuite le nom de l'interface WIFI (ici *eth3*) par analyse de `/proc/net/wireless`.

```
# Get wireless net device
WDEVICE=`get_wlan_dev`
```

On effectue ensuite l'initialisation par `iwconfig` des paramètres WIFI soit le mode et l'identificateur .

```
echo "Wifi init on $WDEVICE..."
iwconfig $WDEVICE mode $WMODE
my_sleep 1
iwconfig $WDEVICE essid $WESSID
my_sleep 1
```

On finit par l'initialisation par `ifconfig` de l'adresse IP associée au routeur sur le réseau WIFI.

```
ifconfig $WDEVICE $WIPADDR
```

Mise en place du firewall

Dans le cas présent, le routeur est installé sur un réseau local déjà protégé par un firewall ce qui fait que nous n'avons pas apporté un soin particulier à la mise en places des règles. Il est cependant nécessaire de mettre en place au minimum une règle de *masquerading* afin de partager la connexion Ethernet entre les différents systèmes connectés au réseau WIFI. Pour cela nous avions à l'époque récupéré sur Internet le script `rc.firewall-2.4` qui convient parfaitement. Le script est appelé à la fin du script rc.wifi. Les deux paramètres correspondent à l'interface *externe* (ici l'interface Ethernet) et l'interface *interne* (ici l'interface WIFI).

```
# Set fw rules
echo "Firewall init..."
/etc/rc.d/rc.firewall $EDEVICE $WDEVICE
```

Initialisation du serveur DHCP

La fonction de serveur DHCP est indispensable afin que les systèmes connectés au réseau WIFI récupèrent des adresses IP sur le réseau 192.168.4.0. Les paramètres du serveur sont décrits dans le fichier `network` cité plus haut soit.

```
#
## Dhcpd parameters
#
DBROADCAST=192.168.4.255
DNETMASK=255.255.255.0
DSUBNET=192.168.4.0
# Range
DADDR1=192.168.4.2
DADDR2=192.168.4.10
```

De part l'architecture, il est nécessaire que le fichier de configuration du serveur DHCP `/etc/dhcpd.conf` soit créé dynamiquement au démarrage du système. Le *root filesystem* étant en lecture seule, le fichier est en fait un lien symbolique vers `/var/run/dhcpd.conf`. Pour créer le fichier, on utilise un prototype `/etc/dhcpd.conf.proto` dont le contenu est fixe.

```
#dhcpd.conf
server-identifier localhost.localdomain;
option domain-name "localdomain";
option routers _WIPADDR_;
option domain-name-servers _DNS1_, _DNS2_;
option subnet-mask _DNETMASK_;
default-lease-time 3600;
max-lease-time 3600;

#ddns-update-style ad-hoc;

option broadcast-address _DBROADCAST_;
option subnet-mask _DNETMASK_;

subnet _DSUBNET_ netmask _DNETMASK_ {
  range _DADDR1_ _DADDR2_;
}
```

Les différentes variables (`_WIPADDR_`, etc.) sont remplacées au démarrage par les valeurs contenues dans le fichier `network`. Pour cela on utilise une ligne basée sur la commande `sed` (Stream EDitor)

décrite dans le script `rc.inet1`.

Suivi des connexions

Les connexions sont visibles au travers du fichier `/var/log/messages` mis à jour par le démon `syslogd`. Il est donc facile de tracer l'historique des dernières connexions.

```
# tail /var/log/messages
Mar  6 17:28:34 pclight dhcpd: DHCPREQUEST for 192.168.4.3 from
00:0f:b5:84:1b:47 via eth3
Mar  6 17:28:34 pclight dhcpd: DHCPACK on 192.168.4.3 to 00:0f:b5:84:1b:47 via
eth3
Mar  6 17:34:03 pclight dhcpd: DHCPREQUEST for 192.168.4.3 from
00:0f:b5:84:1b:47 via eth3
Mar  6 17:34:03 pclight dhcpd: DHCPACK on 192.168.4.3 to 00:0f:b5:84:1b:47 via
eth3
Mar  6 17:50:47 pclight -- MARK --
Mar  6 18:10:47 pclight -- MARK --
```

Conclusion

Ce projet a permis de mettre en application des sujets assez peu décrits dans la littérature comme l'utilisation des DoC ou de CRAMFS. Le système pourrait être enrichi par des fonctions de configuration graphiques similaires à ce qui est disponibles sur les produits commerciaux. Le système actuel est en cours d'évolution vers une architecture 2.6 et susceptible à terme d'être utilisé à plus grande échelle pour des applications réseau professionnelles.

Bibliographie

- L'ouvrage *Linux embarqué 2ème édition*, paru aux édition Eyrolles en novembre 2005 dont la présentation est accessible depuis http://www.ficheux.org.
- L'article *Embarquez Linux! (ou Linux everywhere)* paru dans Linux Magazine et disponible sur http://www.ficheux.org
- L'article *Linux embarqué* par Michel Stempin. Linux Magazine, Novembre 2001 disponible sur http://docs.mandragor.org/files/Misc/GLFM/lm33/Linux_Embedded.html.
- Le site *Embedded Linux page* par Patrice Kadionik sur http://www.enseirb.fr/~kadionik/embedded/embeddedlinux.html
- Site de la société M-Systems sur http://www.m-sys.com
- Site de la société SILINK sur http://www.silink.fr