OpenWrt : fresh air for (wlan) routers

Florian Fainelli
florian@openwrt.org

Rencontres Mondiales du Logiciel Libre 2006
Vandoeuvre-lès-Nancy
Durée : 30 minutes

Jeudi 6 Juillet 2006

Sommaire I

Sommaire II

Sommaire III

Sommaire IV

Qu'est-ce qu'OpenWrt

- contraction d'Opensource Wireless Technology
- distribution GNU/Linux minimaliste sous licence GPL
- ensemble de fichiers Makefile permettant de construire un système complet
- dépôt de paquetages et de mises à jour

Historique du projet

- OpenWrt a été crée à l'initiative de Gerry Rozema (aka groz) et Mike Baker (aka [mbm]) en novembre 2003.
- Dès le début, Gerry et Mike perçoivent le potentiel d'un firmware basé sur Linux et les limitations dues au firmware Linksys, et décident de le remplacer par un système minimaliste, construit avec un buildroot uClibc de l'époque.
- La philosophie est simple : tout en ligne de commandes via SSH

Contexte

- A l'époque où la première version d'OpenWrt voit le jour, les firmwares Sveasoft sont déjà sortis depuis quelques mois et ajoutent de nombreuses fonctionnalités au firmware Linksys original tout en gardant l'interface web identique.
- Quelques mois plus tard, le firmware DD-WRT sortira, un fork d'OpenWrt, la principale raison étant le manque d'interface web.

Etat de l'art

Aujourd'hui, OpenWrt est constitué de 5 développeurs principaux aidés par de nombreux contributeurs :

- Mike Baker ([mbm])
- Imre Kaloz (Kaloz)
- Nicolas Thill (Nico)
- Felix Fietkau (nbd)
- Florian Fainelli (florian)

Outils de développement

- dépôt subversion
- interface web Trac : https ://dev.openwrt.org

Organisation du dépôt Subversion

Le dépôt subversion est organisé en différents répertoires :

- 2 branches : **whiterussian/** et **buildroot-ng/**
- 5 tags : whiterussian_rc1 à 5
- 1 répertoire de paquetages : **packages/**
- kamikaze dans trunk (en cours de migration vers **buildroot-ng/**et **packages/**

Whiterussian

Whiterussian est actuellement la version dite "stable" du firmware OpenWrt. Elle fonctionne bien sur les matériels à base de cartes Broadcom 947xx et 953xx, soit pour les plus répandus :

- Linksys WRT54G v1.0 à v4
- Asus WL-500g (Deluxe, Premium)
- Motorola WR850G, WE500G
- Buffalo WBR-B11, WBR-G54, WLA-G54

Elle sert de base aux firmwares de nombreuses communautés Wifistes, et à des sociétés comme FON (firmware fonbasic).

Limitations de whiterussian

Bien que whiterussian fonctionne bien, le firmware est limité par :

- la nécessité d'utiliser un pilote binaire Broadcom pour la carte Wi-Fi, qui restreint à un noyau 2.4
- des paquetages difficiles à ajouter et à maintenir
- le support limité du matériel (uniquement Broadcom 47xx/53xx)
- une interface web qui repose beaucoup sur la présence d'une NVRAM

Kamikaze I

Devant ces difficultés et l'apparition croissante de nouveaux matériels fonctionnant sous Linux, la branche **Kamikaze** est ouverte.
De nouvelles plateformes sont ainsi supportées :

- Texas Instruments AR7 (noyau 2.4)
- Atheros AR531x (noyau 2.4)
- Aruba (noyau 2.6)
- x86 (noyaux 2.4 et 2.6)

Kamikaze II

- Broadcom SiByte (noyau 2.6)
- AMD Alchemy (noyau 2.6)
- Intel Xscale IX42x (noyau 2.6)
- Router Board RB532 (noyau 2.6)

Limitations de Kamikaze

Kamikaze souffre actuellement des inconvénients suivants :

- difficulté de stabilisation des noyaux, matériels non intégralement fonctionnels (Wi-Fi inopérant la plupart du temps)
- ajout et maintient de paquetages encore trop proche de Whiterussian
- maintient de 2 dépôts de paquetages compilés avec une toolchain différente

buildroot-ng

- abstraction et simplication d'écriture des fichiers **Makefile** et compatibilité avec la syntaxe précédente
- les paquetages dépendant fortement du noyau vont dans **buildroot-ng**, les autres dans **packages/**
- dépôt multi-architectures indépendamment du du système de base

Principaux chantiers

- Finalisation de **buildroot-ng**
- Portage AR7-2.4 vers AR7-2.6
- Portage Broadcom 63xx 2.6
- Refonte de **webif**
- Ré-écriture de la documentation utilisateur

Intérêts d'OpenWrt

- système complètement paramètrable du noyau au système de fichiers
- firmware unifié et identique quelle que soit la plateforme
- indépendance du système par rapport à des versions figées
- code intégralement GPL

Considérations légales

Plusieurs questions légales se posent lorsque vous avez connaissance qu'un matériel donné fonctionne sous Linux :

- le constructeur livre-t-il le code source du firmware ?
- ce matériel utilise-t-il des pilotes fournis sous forme de binaires ?
- est-ce bien Linux ou uClinux ?
- le code fourni est-il bien sous licence GPL ou compatible ?

Prouver qu'un matériel fonctionne sous Linux

Plusieurs moyens s'offrent à vous pour montrer qu'un matériel fonctionne sous Linux :

- récupérer le firmware binaire et tenter de le séparer en : bootloader, noyau, système de fichiers (attention au Big/Little Endian !)
- brancher une console série/JTAG sur le routeur
- utiliser un bug de l'interface web pour récupérer un dmesg, cat /proc/xxxx

Violation de la GPL

Conformément à la licence GPL, toute utilisation commerciale de logiciels libres oblige les constructeurs de matériel à :

- fournir les sources du noyau Linux
- fournir les sources des applications GPL présentent dans le sytème de fichiers
- fournir les sources de la chaîne de production GNU et des outils de création du système de fichiers

En cas de non-respect de la GPL, saisissez sans plus attendre :
`http ://gpl-violations.org`

Base de travail

Votre base de travail est donc constituée :

- des sources du noyau Linux, modifiées pour avoir le support de la carte, et avec les derniers correctifs de l'architecture (arm, mips, ppc ...)
- des pilotes binaires et firmwares pour la carte Wi-Fi, Ethernet, modem ADSL ...
- des outils binaires de création du firmware : CRC, version, bourrage ...

Il y a peu de chances que vous réussissiez à obtenir un firmware fonctionnel avec les outils constructeurs.

Evaluation du travail à réaliser

Compte tenu de ce dont nous disposons, pour obtenir un support fonctionnel du périphérique avec OpenWrt, et du fait que nous devons respecter la GPL, il nous reste à réaliser les tâches suivantes :

- analyser et générer les différences entre les sources du noyau Linux officiel et celles fournies
- créer un executable permettant de créer des images firmware valides (calcul CRC, versions, bourrage ...)
- garder la compatibilité entre les pilotes binaires et les versions actuelles du noyau Linux (attention à l'option VERSIONING)

Ajout d'une nouvelle architecture

Maintenant que nous avons les éléments nécessaires pour construire un système OpenWrt pour notre architecture cible, ajoutons le :

- ajout d'une entrée dans **target/Config.in**
- ajout d'un répertoire de la forme : **target/linux/architecture-2.x** (noyau 2.4 ou 2.6) contenant les patches spécifiques
- ajout d'un répertoire de la forme : **target/image/architecture** décrivant comment construire le firmware final

Conventions

- Les architectures doivent être nommées de la même manière que le noyau Linux sous **arch/**
- Nous vous recommandons de faire booter un noyau "vanilla" patché plutôt que de commencer à créer le firmware final ;)
- Pensez à séparer les patchs : support de la carte, pilotes, correctifs divers ...

target/Config.in

```
config LINUX_2_6_ARCHITECTURE
        bool "Architecture toto [2.6]"
        select mips
        select LINUX_2_6
        select PCI_SUPPORT
        select PCMCIA_SUPPORT
        help
          Une petite description
```

target/linux/architecture-2.x/Makefile I

```
include $(TOPDIR)/rules.mk

LINUX_VERSION:=2.6.16.7
LINUX_RELEASE:=1
LINUX_KERNEL_MD5SUM:=9682b2bd6e02f3087982d7c3f5ba824e

include ./config
include $(INCLUDE_DIR)/kernel.mk
```

target/linux/architecture-2.x/Makefile II

```
include $(INCLUDE_DIR)/kernel-build.mk

$(LINUX_DIR)/.patched: $(LINUX_DIR)/.unpacked
        [ -d ../generic-$(KERNEL)/patches ] &&
$(PATCH) $(LINUX_DIR) ../generic-$(KERNEL)/patches $(MAKE_T
        [ -d ./patches ] &&
$(PATCH) $(LINUX_DIR) ./patches $(MAKE_TRACE)
        @$(CP) config $(LINUX_DIR)/.config
        touch $@
```

target/image/architecture/Makefile I

```
include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/image.mk

define Build/Compile
        rm -f $(KDIR)/loader.gz
        $(MAKE) -C lzma-loader \
                BUILD_DIR="$(KDIR)" \
                TARGET="$(KDIR)" \
```

target/image/architecture/Makefile II

```
                install
endef

define Build/Clean
        $(MAKE) -C lzma-loader clean
endef
define Image/Prepare
        cat $(KDIR)/vmlinux |
$(STAGING_DIR)/bin/lzma e -si -so -eos -lc1 -lp2 -pb2 > $(K
```

target/image/architecture/Makefile III

```
endef

define Image/Build/périphérique
        dd if=$(KDIR)/loader.elf
of=$(BIN_DIR)/openwrt-périphérique-$(KERNEL)-$(2).bin
bs=131072 conv=sync
        cat $(BIN_DIR)/openwrt-$(BOARD)-$(KERNEL)-$(1).trx
>> $(BIN_DIR)/openwrt-périphérique-$(KERNEL)-$(2).bin
endef
```

target/image/architecture/Makefile IV

```
define trxalign/jffs2-128k
-a 0x20000
endef
define trxalign/jffs2-64k
-a 0x10000
endef
define trxalign/squashfs
-a 1024
```

target/image/architecture/Makefile V

```
endef

$(eval $(call BuildImage))
```

include/target.mk

```
...
$(eval $(call kernel_template,2.6,architecture,
2_6_ARCHITECTURE))
...
```

Déboguer et stabiliser

Outils de déboguage :

- GDB
- EJTAG (si disponible)
- ksymoops
- l'utilisation de printfk
- un noyau compilé avec les options de déboguage adéquates
- documentation du bootloader (RedBoot, CFE, YAMON, RomE ...)
- faire appel aux utilisateurs et développeurs

Problèmes ultérieurs

Une fois que vous avez réussi à avoir un noyau bootant sur votre matériel, il n'est pas encore utilisable. Il y a fort à parier que :

- les pilotes ne fonctionnent pas ou pas très bien
- la cartographie de la flash ne soit pas forcément reconnue
- le système soit lent, et nécessite des modifications dans le noyau

Personnaliser le système

Vous pouvez intégralement personnaliser votre système sur votre routeur point d'accès Wi-Fi, notamment :

- ajouter un portail captif, un serveur RADIUS
- gérer au mieux le pare-feu avec iptables
- ajouter des protocoles et piles réseau, des services ...
- ajouter des pilotes de périphériques supplémentaires : webcam, cartes Wi-Fi ...
- ajouter des des fonctionnalités à l'interface **webif**

Ajouter des paquetages

Nous vous invitons à participer à l'effort de migration des paquetages présents dans **kamikaze** vers la syntaxe **buildroot-ng**. Contrairement aux systèmes précédents où il fallait créer au moins 3 fichiers :

- Makefile
- Config.in
- ipkg/paquetage.control

buildroot-ng permet de s'affranchir de ces fichiers en les rassemblant en un seul fichier **Makefile**.

Hiérarchie

Les paquetages sont organisés sous forme la forme suivante :

```
packages/
  section/
    paquetage/
      Makefile
      patches/
```

packages/section/Makefile I

```
include $(TOPDIR)/rules.mk

PKG_NAME:=mon-paquetage
PKG_VERSION:=alpha-beta-4
PKG_RELEASE:=1
PKG_MD5SUM:=5988e7aeb0ae4dac8d83561265984cc9

PKG_SOURCE_URL:=ftp://ftp.openwrt.org/mon-paquetage
```

packages/section/Makefile II

```
PKG_SOURCE:=$(PKG_NAME)-$(PKG_VERSION).tar.gz
PKG_CAT:=zcat

PKG_BUILD_DIR:=$(BUILD_DIR)/$(PKG_NAME)-$(PKG_VERSION)
PKG_INSTALL_DIR:=$(PKG_BUILD_DIR)/ipkg-install

include $(INCLUDE_DIR)/package.mk

define Package/mon-paquetage
```

packages/section/Makefile III

```
SECTION:=libs
CATEGORY:=Libraries
TITLE:=Un programme de demonstation
DESCRIPTION:=Un programme de démonstration également
URL:=ftp://ftp.openwrt.org/mon-paquetage
endef

define Build/Configure
$(call Build/Configure/Default,--option-supplémentaire=paqu
```

packages/section/Makefile IV

```
endef

define Build/Compile
        rm -rf $(PKG_INSTALL_DIR)
        mkdir -p $(PKG_INSTALL_DIR)
        $(MAKE) -C $(PKG_BUILD_DIR) \
                DESTDIR="$(PKG_INSTALL_DIR)" \
                all install
endef
```

packages/section/Makefile V

```
define Package/mon-paquetage/install
        install -m0755 -d $(1)/usr/lib
        $(CP) $(PKG_INSTALL_DIR)/usr/lib/libpaquetage.so.*
endef

define Build/InstallDev
        mkdir -p $(STAGING_DIR)/usr/include
        $(CP) $(PKG_INSTALL_DIR)/usr/include/en-tête-paquet
```

packages/section/Makefile VI

```
        mkdir -p $(STAGING_DIR)/usr/lib
        $(CP) $(PKG_INSTALL_DIR)/usr/lib/libpaquetage.{a,so
        touch $(STAGING_DIR)/usr/lib/libpaquetage.so
endef

define Build/UninstallDev
        rm -rf \
          $(STAGING_DIR)/usr/include/en-tête-paquetage.h \
          $(STAGING_DIR)/usr/lib/libpaquetage.{a,so*}
```

packages/section/Makefile VII

```
endef

$(eval $(call BuildPackage,mon-paquetage))
```

Obtenit de l'aide

N'hésitez pas à nous contacter à l'aide des moyens suivants :

- IRC : irc.freenode.net #openwrt et #openwrt-devel
- Mailing-list : openwrt-devel@openwrt.org
- Forum : http ://forum.openwrt.org

Devenir développeur

- N'hésitez pas à nous soumettre des patchs ajoutant des paquetages
- Testez et rapportez les bugs que vous rencontrez, tout correctif est le bienvenu
- Faîtes fonctionner OpenWrt sur une nouvelle architecture, stabilisez un portage existant ..

Merci à tous

Merci de votre attention, n'hésitez surtout à pas poser vos questions