Linux on commodity network hardware

LUGOD talk

Josh Parsons

August 15, 2005

What are the devices I'm talking about

If you have small office / home ("SoHo") network, chances are you are using a router made by Linksys, ASUS, D-Link, or Motorola.

If you do, then probably your router is already running linux.

(And a lot of other GNU software)

You can use the *firmware upgrade* feature of your router to install a *more open* version of the firmware onto it

This has many advantages.

Advantages of a more open firmware

What do I mean by “more open”?

- Get more control over the device’s existing functionality. e.g.:
  - Set up more sophisticated firewall rules with iptables.
  - Override firmware limitations on wireless power/channel.
  - Remote administration with ssh.
- Install new software / new functionality. e.g:
  - Run a web server / wiki / blog / proxy.
  - Run an irc server or proxy.
  - P2P file share.
  - Email / printer / webcam / telephony / network sound / etc.

How does it work: Hardware

What's inside a typical SoHo router / network device?

- A *specialised "embedded systems" processor* — MIPS or ARM architecture — about 100-300MHz clock rate.
- Some *RAM* (between 8 and 32MB)
- Flash *ROM* containing the firmware (2–8MB)
- An *ethernet interface*
- Sometimes also *802.11a/b/g wireless* or *USB*.
- Sometimes other specialised network hardware such as an ethernet switch.

These devices are all pretty similar internally, even though they are marketed for different purposes.

How does it work: what happens at boot?

These devices treat their ROMs like a disk. The ROM is *partitioned* into several areas:

- The *boot loader* (like LILO, GRUB, SYSLINUX)
- A re-writable *configuration area*.
- The *linux kernel*.
- A *root filesystem*.

When you switch the device on, the boot loader loads the kernel into RAM, and it begins running using the root filesystem.

Reflashing the firmware replaces only the kernel and root, leaving the boot loader and configuration intact, so that a failed reflash does not *brick* the device.

What devices are available?

This is very much a partial list!

- Linksys’ popular WRT54-series routers.
- Linksys’ NSLU2 “network storage unit”.
- Asus’ WL500 series wireless routers.
- Other routers by D-Link, Motorola, Belkin, ...

These are all the sorts of things you might buy in Radio Shack.

The manufacturers sometimes change the hardware completely with a minor version number change. Check that there is a supported “more open” firmware for a device before you buy it!

What firmware is available?

Usually you will have two options for user-supported firmware. There'll be a *conservative* firmware that preserves the functionality of the manufacturer's firmware, and a *clean* firmware that just converts the device into a generic linux server.

- For the WRT54*: HyperWRT / OpenWRT
- For the NSLU2: Unslung+Optware / OpenSlug
- For the WL500*: Oleg's+Optware / OpenWRT

In either case, there'll be a range of add-on *packages* available to add more functionality over-and-above what's supplied by the firmware.

The Linksys NSLU2

Features:

- Intel XScale processor (bigendian ARM) running at 133MHz. Can be "de-underclocked" to 266MHz.
- 8MB flash, 32MB RAM.
- One 100TX ethernet port.
- Two USB 2.0 ports.
- Intended as a dedicated samba server.

The NSLU2 is fondly known to its users as the "slug".

I maintain the Optware/Unslung package repository and build system.

Unslinging the NSLU2

The NSLU2 was very popular with embedded linux hackers when it first appeared because it was one of the first such device to feature USB 2.0. So it's practical to attach a USB hard drive.

This is important because there's not a lot of room on the flash to install software. And the hard drive can be used for swapping.

The *Unslung* firmware has a feature that allows it to copy parts of the firmware that would underwise be in RAM to the NSLU2's harddrive. This saves RAM, and allows packages to be installed on the harddrive. The process is called *unslinging*.

The moment of truth

I will now attempt to reflash and unsling the slug I have with me. There are several methods for doing this:

1. Using the web interface. Requires firmware to be functional. Does not work on every version of unslung.
2. Using the boot loader's upgrade mode. Works even if the firmware is broken. Requires a special (but opensource) utility.
3. Using the boot loader and tftp. Works even if the firmware is broken. Requires only standard tools, but it is slightly tricky to use them.
4. Using a soldering iron.

I am going to use method #2.

What can we do now?

I unslung the slug to a 1GB flash drive. If we had internet access, we could now begin installing packages from the Optware/Unslung package repository onto the slug.

Packages available include:

- Apache, and a choice of other php-enabled web servers.
- Mediawiki for use with the above.
- Typical linux server stuff: ssh, ldap, cvs, ipsec, nfs.
- Things you might not think of: asterisk, esd, x11, qemu.

Developing packages for unslung

Though it is possible to install a compiler and other development tools on the slug itself (and some packages can only be built that way) it is much easier in the long run to cross-compile on a PC or other, faster, system.

It is important to keep known-to-work recipes for building packages in a systematic way. These recipes are known as "meta-data". They are like RPM's .spec files.

The package system used with unslung is called "optware". Optware consists of a build system (a set of compilers, tools, and makefiles that will cross compile packages for unslung) and a set of meta-data for each package.

Openslug

Unslung preserves the stock firmware’s web interfaces and samba server. These are tied to the stock firmware’s kernel, gcc, and glibc configuration, which is a bit out of date, and includes some non-standard patches to the linux USB subsystem.

If we gave up on those, and just installed a clean gnu/linux system on the slug, we could upgrade the kernel to 2.6, and enable many more kernel features.

This is the purpose of the Openslug firmware.

Openslug is based on the OpenEmbedded build system, and has its own package repository.

What's interesting about Unslung development?

Being involved with Optware/Unslung has been a great learning experience for me.

- The big-endian ARM architecture is quite exotic, so I'm often found portability problems with famous pieces of software and been able to contribute fixes. This was true of php and gcc.
- I've learned how to write more portable code myself.
- Doing firmware development, I've learned a lot about the linux kernel and GNU C library.
- Optware/Unslung is by far the largest open-source project I've been closely involved in. We have about 60 active developers located all over the world, communicating by irc.

We will see more of these devices.

The manufacturers like Linux for both technical and social reasons.

The technical reasons are its stability and good support for IP networking and for MTD. The social reason is that it is free as in beer to anyone who can compile it.

They take the same attitute to hardware: the NSLU2 is very similar to intel's development boards. Linksys clearly wanted a generic system on which they could run (mostly) pre-written software.

We will see more and more of these devices. This is a good thing, as it will mean that Linux will become more mainstream.

But it is important to get the manufacturers to respect the GPL.

The Unslung/Optware Community.

Unslung and Optware are a good example of user-supported / user-developed software done right.

What makes it work so well?

- Revision-control systems. We use monotone and cvs.
- Wikis. We have a community rule that encourages users to add to and improve the wiki.
- Freenode irc. The core developers are available to help on #nslu2-linux. Community rules about bothering them.
- Easy to become a developer. Publically post a working package recipe and you get cvs write access. We have had no "rogue developers" yet, and if we did, any damage would be reverted. A wiki-like model of development.