

Tom's Networking

How To: Hacking the Linksys NSLU2 – Part 5 Moving to Unslung

By Jim Buzbee – October 7 2004

Introduction

If you've been following along with my articles [[Part 1](#)] [[Part 2](#)] [[Part 3](#)], [[Part 4](#)], you've done a lot of work on your own, but it's now time to converge with the NSLU2 development community. We've built a few packages ourselves, but there's no way we can match the output of the world-wide NSLU2 development community. If we align ourselves with their efforts, we'll be able to take advantage of all the software packages that have been built for the NSLU2. These packages include our favorites such as an nfs server, an iTunes server, and new packages like an ssh server, an ftp server, a telnet client, etc.

There are several efforts underway in the developer community, but the focus of our convergence will be the Unslung firmware and package system. This firmware brings several basic extensions to the box. First, it's designed to maintain standard Linksys functionality, while allowing a high degree of customization. This is done with a more extensive variation of the flash modification we made in my last article.

Second, it adds a standard package scheme that allows easy installation of new software packages from an Internet accessible package library. And finally, it takes an approach that frees up 10 Megabytes of RAM. That's a lot in a box that only starts with 32 Megabytes. Freeing up this RAM makes it feasible to think of running more demanding applications such as a database system or a full-featured web server.

How did the NSLU2 developers magically free up this RAM while not only keeping the standard Linksys functionality, but adding a great deal of new features? Let's find out.



Disclaimer: It goes without saying, but I'll say it anyway. Exploring the NSLU2 by looking at its internal file structures using any method that doesn't modify its code should leave your warranty intact. But modifying the NSLU2 in any way will void your warranty.

TomsNetworking, Tom's Guides Publishing and I are not responsible for any damage that the information in this article may cause to your NSLU2 or any data it manages.

So [download a copy of the current firmware](#) before you start, and don't go trying to get help from Linksys if you break it.

Boot-time Alternatives

In order to understand how Unslung frees up the memory, we must understand the basic boot-time architecture of the NSLU2. As we touched on in my [last article](#), the first code to run when the box is powered on is the [RedBoot](#) boot loader. Among other things, the boot loader is responsible for copying the Linux kernel and ram disk from flash into RAM.

When the copying is complete, control is handed over to the Linux kernel. The kernel starts up and mounts the RAMdisk as the root or base filesystem. It then starts up the initialization processes from within that root filesystem. The initialization processes mount the hard drive and/or the flash drive and then continue on to the web server, the network file servers, etc. At this point the box is running and ready for use.

All of the basic functionality of the NSLU2 is contained in the RAMdisk because the box is designed to boot even if there's no hard drive attached. That's a nice feature, but if you think about it, it can be very wasteful. You may have a world of free space on your 250 Gigabyte hard drive, but you're stuck trying to squeeze all desired executables, libraries, html files, etc into the filesystem on a tiny RAMdisk.

What if you could put all of the files on the disk drive instead of the RAM drive? Then you'd be able to allocate a much larger filesystem and in the process, free up the memory that was previously allocated to the RAM drive. It would also be nice if you could be backward compatible with the Linksys boot sequence. If the drive had the filesystem on it, you'd use it. If a disk was not plugged in or if the disk didn't have a filesystem on it, you'd use the filesystem on the RAMdisk just like before. This is the approach taken in the Unslung firmware. Now let's explore how it's done.

Pick your Filesystem

It's common practice to build a Linux kernel with a pre-specified indicator of which drive to use for the root filesystem. This can be done either in the kernel itself or it can be passed to the kernel from the boot loader. In the case of the NSLU2, Linksys hard-coded the kernel, specifying that the root filesystem was a RAMdisk. That's all Linksys ever intended to use. But like most things in Linux, there's a method to avoid the use of hard-coded variables such as the root filesystem designator.

Linux has the capability to make a run-time determination of where the root filesystem is located. This capability is commonly used where a single kernel is built to run on varying hardware platforms. For example, the same kernel image can boot on a system where the root can either be found on an IDE drive or a SCSI drive. Before Linux chooses where to

find the root, it pokes around and determines what drivers to load and where to find the root. When Linux figures it out, the proper root is mounted and the boot continues normally. This is the feature used in Unslung.

The Unslung code for root detection is not part of the kernel proper. Instead it's found in a Linux feature called the initrd, or initial ram disk. The initrd is a special temporary ram disk designed to hold user-code that's executed before the root filesystem is mounted. The Unslung developers first put a dummy root designator in the kernel and then added code to the initrd to check the hard drives for the real root filesystem. If one of the attached drives has a root filesystem, Linux is told to use it rather than fall back to the RAMdisk. Once Linux decides to use the hard drive for the root filesystem, the 10 Megabyte RAMdisk is freed up, and the RAM falls back into the free pool for other uses.

Installing Unslung

The next feature of Unslung that we'll explore is the customization feature. Much like we did in my last article, Unslung adds hooks to the standard startup scripts, but it goes much further. Unslung's scripts—called diversion scripts—add a hook to every startup script. The advantage is that you can insert code at any point in the startup sequence.

Additionally, the diversion scripts have a feature that allows you to completely skip or replace the standard Linksys functionality. For example, since I don't use SMB any more, my diversion script simply does a "return 0" which tells the hook in the Linksys script to skip the remaining portion of the Samba startup. This would also allow me to startup Samba with different options or even use my own version of Samba before returning 0. Unslung adds this capability to every Linksys startup script.

The last major feature that Unslung brings to the table is a standardized package system called [ipkg](#), which was originally developed for hand-held Linux systems. Since then, its use has been expanded to other devices with limited resources. Using a standard package system makes it easy to add new features to the box.

For example, it takes a one-line command to add a secure shell server to our little box. The package is then downloaded from the Internet, configured and installed automatically. As of this writing, there were 19 packages available for installation, but work is underway to converge with a more established build system which will bring more than 1500 packages to the NSLU2!

Enough overview, let's try it out. As before, I'm assuming that you've enabled Telnet on your box as specified in my first article. If you haven't, there's now a slightly easier method detailed [here](#).

If you've been following along with my earlier articles, the first thing we need to do is backup all of our previous changes. The Unslung firmware uses some of the same directories that we created in the /share/hdd/conf directory, bin, etc, and rc.d. We'll just

move our directories out of the way and let the Unslung installation do its thing. When it's done, we can put our specific changes back in place.

First, Telnet in to the NSLU2 and rename the directories to avoid any conflicts:

```
cd /share/hdd/conf
mv bin bin.old
mv etc etc.old
mv rc.d rc.d.old
```

This time we won't be building our own firmware, relying instead on a pre-built firmware, UNSLUNG-1.11-beta.zip, that's available [here](#). Download it to your computer and unzip it into a working directory. You'll end up with three files, a README file and two different firmware files. One firmware image is for using Linux ext3 formatted USB flash drives, the other is for using Windows vfat formatted USB flash drives. For this article, we'll use the vfat flash image, UNSLUNG-1.11-beta-V23R25.bin, since its behavior more closely follows the standard Linksys firmware. Browse the README file for detailed information on this firmware, and for instructions on how to build it yourself if you so desire.

When you think you're ready to go, shut down your NSLU2 and unplug your hard drive. Power the NSLU2 back up and install the Unslung firmware using the standard Linksys flash upgrade utility. When the box boots back up, enable Telnet as shown in my first article using the Management/telnet.cgi URL. Telnet into the box using root for a user and uNSLUng for a password.

Exploring Unslung

At this point we're using the new firmware, but functionality hasn't changed much. We're still using the standard RAMdisk and no new packages have been installed. For the next step, hot-plug your hard drive back into USB port 1. After a few seconds, entering a df command should show your drive mounted.

Now we'll populate the root filesystem on disk using a copy of the RAM filesystem. The new unslung command for this is:

```
/sbin/unsling
```

This command should give a few lines of output as it copies the RAMdisk to the hard drive. When it completes, type:

```
sync
reboot
```

The box will reboot as normal. When it comes back up you'll need to re-enable Telnet once again using the standard telnet.cgi URL. Your user name and password should be

preserved from your previous modifications. A `df` command will show your `conf` partition mounted on both the `/` directory and in the original location where the Linksys utilities expect to find it. And you should have an additional 10 Megabytes of free memory! To verify this, enter:

```
cat /proc/meminfo
```

You should see a small amount of memory listed as free and a larger amount listed in buffers and cached. This memory is available for applications when needed.

Now we can re-enable Telnet using a diversion script. The details of creating diversion scripts can be found in the README file that came with the firmware, but here's what I did. I made a copy of `/etc/inetd.conf` in the directory `/opt/etc/`. Then I created an `xinetd` diversion script called `rc.xinetd` in the `/unslung` directory:

```
#!/bin/sh
```

```
cp /opt/etc/inetd.conf /etc/
```

```
return 1
```

This will run before `inetd` is started and just makes sure that the configuration file has Telnet enabled. The `return 1` statement signifies that the rest of the standard script should be executed. If you're curious about the execution path, you can see how the diversion calls are made in the original `rc` files, e.g. `/etc/rc.d/rc.xinetd`. A reboot should now cause the box to come back up with Telnet running.

Now, on to the package system and some new applications.

Adding Applications

As mentioned earlier, the package system in use with Unslung is `ipkg`. First we'll get our package database up to date. While logged into the box, execute the `ipkg` command with an "update" parameter:

```
ipkg update
```



NOTE: Make sure that the NSLU2 is connected to a LAN that has Internet access or `ipkg` won't work.

This should give a few lines of output as it updates the internal database. I've seen several of the `ipkg` commands get occasional failures when communicating with the base repository. If this occurs, just try again. If you continue to have problems, make sure that your NSLU2 has a default gateway and DNS server set up from the standard Linksys configuration screen.

Now let's see what packages are available for our little box:

```
ipkg list
```

You should see a list of packages that includes both nfs and mt-daapd. Let's try to install nfs. First we need the portmapper:

```
ipkg install portmap
```

You should see several lines of output as the package is fetched from the main nslu2 package repository and then installed. Next install the nfs server itself:

```
ipkg install nfs-server
```

You should see status messages as before, indicating a successful install. If you had nfs installed from my previous article, you can copy the backed-up exports file into the etc directory:

```
cp /share/hdd/conf/etc.old/exports /etc/
```

Next time you reboot, nfs should be running. You may have noticed another nfs package when we did the ipkg list. This is a different package that supports version 3 of the nfs protocol. The biggest (no pun intended) feature of this new server I've noticed is the ability to handle files larger than 2 Gigabytes. I don't have a lot of experience with this package other than to verify that it runs. If you want to try it, un-install the first nfs package using the "remove" parameter with ipkg.

If you want to try mt-daapd, it's installed the same way:

```
ipkg install mt-daapd
```

Much easier than the install in my previous article! After the install completes, copy your backed up config file into the /etc directory:

```
cp /share/hdd/conf/etc.old/mt-daapd.conf /etc/
```

If this is your first mt-daapd install, copy the configuration file from the /opt/etc directory to the /etc directory. As a side note, I've had some reports of privilege problems with mt-daapd regarding access to the MP3 files. If you have this sort of issue, either change the protection on your MP3 tree, or change the "runas" user in the configuration file to a user with privilege to read the music files. Double check the mt-daapd configuration file to make sure that all of your paths are correct, and the next time you reboot, you should once again be iTunes enabled!

Now let's try a new package. Telnet has been handy, but everyone should know by now that it's a security hole waiting to happen. Telnet user names and passwords are sent in clear text across the network. If someone is sniffing your network, you're had. Secure shell (ssh) is a much better solution. For boxes with limited resources like ours, the ssh server of choice is called dropbear. Installing it is as easy as:

```
ipkg install dropbear
```

This installation takes a bit longer because it has to generate cryptographic keys, but when it completes, you should be ssh enabled. From my Linux or Macintosh OSX box, I can now log into my NSLU2 like so:

```
ssh -l root 192.168.1.10
```

Once you're ssh enabled, you may want to consider removing the telnet setup script from the /unslung directory.

Expanding the NSLU2 even further

To keep your system up to date, occasionally do an "ipkg update" followed by an "ipkg list" to see what new packages are available. If you're ambitious, build some of your own and contribute to the effort!

As a last step in the Unslung process, the developers would appreciate feedback. Just go to the NSLU2 Yahoo page and enter your experiences.

As cool as Unslung is, it's not the final step in the evolution of the NSLU2 from a dedicated NAS box to an Open Source application platform. In my next article, I'll describe the results of efforts currently underway to expand the NSLU2's horizons to over 1500 applications!

In the meantime, you can keep up with my activities on my [Linux on the NSLU2 page](#) or check out the [NSLU2 Linux development group](#) site.