A backup server using NSLU2, unslung and rsnapshot

In a previous article I explained how you can use rsnapshot to build a backup server that stores daily/weekly/monthly data snapshots in a disk-efficient way.
The assumption there was, that you would use a Slackware Linux machine as the backup server.

This article however, will show you how you can use a cheap off-the-shelf NAS (*Network Attached Storage*) plus a big external USB hard disk to build a *rsnapshot* backup server that is silent and power-efficient.

The hardware & software I used to build my backup server are:

- LinkSys NSLU2 *NAS* (commonly called a *slug*)
- External USB 2.0 hard disk enclosure with 1 TB disk storage
- USB Flash drive with 1 GB storage
- uNSLUng firmware
- The uNSLUng package repository

The NSLU2 is a small and silent, *ARM CPU* powered network storage server that does not have disk storage built in - you need to attach one or two external USB hard drives to it's USB connectors on the back. It is one of the many relatively cheap solutions for out-of-the-box network storage demands. It has a spartan administrative user interface, and is not customizable. However, it's firmware is built on Linux so the source code modifications have been made available to the public. Initially, there was only the “unslung” firmware alternative which enables you to add additional software to the device. Nowadays, several alternative opensource firmwares are available, all sprouting from the original unslung concept. You can find all of these (like OpenWrt and SlugOS) referenced on the nslu2-linux homepage. The range of supported devices has grown, too.

Below you find the steps I took to add extra functionality to the NSLU2.

Choice of firmware

You can find a good overview of available alternative firmwares for the NSLU2 on the nslu2-linux homePage. It depends on what you want to do with the device, and in my case I did not want to change the NAS function of the NSLU2, only add functionality to it (such as a SSH server and rsnapshot software). So, I opted for the uNSLUng firmware, which also was the first on the market.

Hardware considerations

The original NSLU2 firmware from LinkSys allows you to add at most two external USB storage devices to the connectors at the back. The alternative firmwares like *unslung* allow you to add a USB hub to “Port 1” and therefore, connect more than two storage devices.
However, the *unslung* firmware uses one of the external USB devices as the *root device* where it will install additional software. You will have to make a judgement call on whether you want the unslung packages and configuration files to be stored on the same disk as where your backup data is going to accumulate.
In particular, if you want to be able to remove your external hard drive from the NSLU2 and use that on another computer, the disk must not contain *unslung* configuration data or else your NSLU2 with *unslung* will not reboot properly!

There are some articles on the nslu2-linux Wiki that you should read before making the decision about what devices to connect to what port:

- Disk behaviour with LinkSys R63 firmware
- Which USB port to use for uNSLUng6

What I decided, is to plug a *1 GB USB flash drive* (the minimum required storage for an unslung root device is 512 MB) into *Port 2* of the NSLU2 and use that as the root device for “unslinging” the firmware. In order to function as a root device, it must be natively formatted by the unslung firmware. The concept of “native format” means that the device gets equipped with two partitions, both with a *ext3* filesystem. One partition will be mounted as `/share/flash/conf` and used to store configuration data. The second partition will be mounted as `/share/flash/data` and will be available for the NAS data storage. You will have to carefully read the above URLs if you decide to use two natively formatted disks. Your initial decisions will affect all further actions with respect to location and use of these storage devices.
A storage device with EXT3 filesystem which has not been natively formatted by unslung will not be visible in the NSLU2 administrative interface.

To *Port 1*, I have attached an external USB *1 TB hard disk*. This device has been manually formatted as and *ext3* filesystem using the `mkfs.ext3` command (i.e. it is not natively formatted by unslung). This makes it easy to unplug this disk from the NSLU2 and use it elsewhere without disrupting the NSLU2's fnctionality (the USB flash device which holds the root device will for ever remain attached to the NSLU2).
We will be using this big disk as the storage device for our *rsnapshot* backups.

Administrative access to the NSLU2

The firmware that ships with the NSLU2 does not have a telnet server. The only management interface is the web interface.
The NSLU2 will start out of the box on IP address "`192.168.1.77`" so you should be able to quickly access it at http://192.168.1.77. If you do not use the subnet **`192.168.1.0/255.255.255.0`** in your network, then you will have to configure a computer with an IP address in that range and use a cross-cable or a ethernet switch in order to make your computer and the NSLU2 talk to each other.
Alternatively you can configure an *IP alias* for your computer's network card: assign an address in the NSLU2's network range to that alias interface and then connect the NSLU2 to your LAN.
The default *username/password* combination of the web interface of the NSLU2 is "`admin/admin`".

If you are re-flashing your *slug* (for instance if you try to recover from a bad flash, see below) then the device's IP address will not be `192.168.1.77` but the address it was using before the re-flash - i.e. the one you gave it.

Once you flashed the NSLU2, you will have to enable the telnet server of the *uNSLUng* firmware before you can do anything else (do not attach any external storage yet!). The default *username/password* combination for the uNSLUng firmware is "`root/uNSLUng`".
To enable the telnet server, you will have to use the web-based management interface - http://192.168.1.77/Management/telnet.cgi.

Flashing new firmware

In order to flash a NSLU2 with custom firmware you will have to install the upslug2 program on the linux workstation you are going to use for the configuration of the device. This small program will do the hard work. I have created a Slackware package which you can download from my repository.
The upslug web page explains the flashing process. Basically, you will have to put your NSLU2 in "*upgrade mode*" (text taken from that page):

1. Shutdown the slug
2. Using a paper clip (or pushpin), push and hold in the reset button. (The reset button is located on the back of the NSLU2 above the power connection.)
3. While holding in the reset button, press and release the power button.
4. Watch the orange Ready/Status LED and after approx 10 seconds the LED will turn solid red. Quickly release the reset button.
5. You should be in upgrade mode which is indicated by the Ready/Status LED alternating between red and green.
   If the Ready/Status LED is flashing yellow rather than red/green, you didn't release the reset button quickly enough after the Ready/Status LED turned red, and the Slug is now in "*assign mode*", whatever that means. Don't panic; just remove the power and start again.

This is how the process looks from the command line. Make sure you have the upslug2 binary in your $PATH and an uNSLUng firmware file ready (I downloaded the `Unslung-6.10-beta.bin` image file, which is contained in Unslung-6.10-beta-firmware.zip):

```
$ upslug2 --device eth2 --verify --image Unslung-6.10-beta.bin
```

```
  NSLU2     00:14:bf:63:4b:23 Product ID: 1 Protocol ID: 0
  Firmware Version: R23V29 [0x2329]

  Upgrading LKG634B67 00:14:bf:63:4b:23
```

In the above command I used "`–device eth2`" to show how you can flash the *slug* in case you have multiple network interfaces and the *slug* is directly connected to your `eth2` interface.

Do not turn off the NSLU2 while you are flashing it! Also do not try to use a wireless network connection for this.

If you are re-flashing a bad flash (it happened to me: the flash got corrupted somehow and the *slug* would no longer properly boot), leave off the "–verify" because that will verify the flash content prior to re-flashing and report failure:

```
$ upslug2 --device eth2 --verify --image Unslung-6.10-beta.bin

  NSLU2     00:14:bf:63:4b:23 Product ID: 1 Protocol ID: 0
  Firmware Version: R23V29 [0x2329]

Upgrading LKG634B67 00:14:bf:63:4b:67
    . original flash contents  * packet timed out
    ! being erased             - erased
    u being upgraded           U upgraded
    v being verified           V verified

  Display:
    <status> <address completed>+<bytes transmitted but not completed>
  Status:
    * timeout occurred         + sequence error detected

  17fdbf+0005c0
...VVVVVVVVvvUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
00:14:bf:63:4b:67: flash verification error (address 0x17FDC0, length 1472)
[std::exception]
 The verification step failed, the flash has not been written
 correctly (or maybe there is a bug in upslug2).  Try repeating
 the verification step and, if that fails for the same reason,
 try repeating the whole upgrade.
```

Recovering from a *bad flash* is straight-forward: just re-flash. As long as you do not overwrite the *RedBoot* sector, you are doing nothing that can cause irreversible harm.

Unslinging to flash device

After flashing with new firmware, enable the telnet server, login as *root* and while logged on perform the "*unsling*":

- Plugin an unformatted flash disk (512MB or larger) into the “`Disk 2`” port
- In the Web GUI, http://192.168.1.77/Management/disk_fs.htm check that the flash disk is detected, and let the NSLU2 format it. After formatting it must display as “`Formatted (EXT3)`”!
- In the telnet session (stay logged in!) enter

```
unsling disk2
```

. You will have to enter a new password for root then.

- Afer the *unslining* process completes, a comment is left in your telnet terminal that you have to reboot the machine:

```
Leave the device disk2, /dev/sda1, plugged in and reboot
(using either the Web GUI, or the command "DO_Reboot")
in order to boot this system up into unslung mode.
```

Configuring network address

If your LAN's network address is not “`192.168.1.0`”, then you will have had to configure a network interface with an IP address in that network range in order to communicate with the device (see section 'administrative access').
Now is the time to reconfigure your NSLU2 for the actual IP address it will be using in your network. Open the URL http://192.168.0.250/Management/setup.cgi?next_file=lan.htm and configure a static IP address for the device. You have the option to use DHCP, but I have had mixed success with DHCP so I prefer a static IP address.

Note:
if you were using a cross-cable between your own computer and the UNSL2, you will have lost your network connection to the device now. You will have to shutdown the device (use the power button on the front bottom) and then connect the device to a LAN network socket after reconfiguring the IP address and power it up again.

In order to install additional packages as outlined in the next section, you will have to re-enable telnet first through the GUI (http://new.ip.address/Management/telnet.cgi)

Additional packages

After *unslinging*, install the following packages as a minimum:

- openssh, rsnapshot, cron, syslog-ng, util-linux, less, e2fsprogs, nfs-utils, man, screen, procps, sendmail, logrotate, nail, diffutils, ntpclient

```
# ipkg update
# ipg list
# ipkg install openssh
# ... etc ...
```

Please follow carefully the post-installation instructions shown on-screen after installing some of these packages (such as syslog-ng, nfs-utils) or they will not work properly. Sendmail will for instance state:

```
sendmail will run as a daemon now. Alternatively, execute:
echo 'smtp    stream  tcp     nowait  root    /opt/sbin/sendmail -bs' >>
/etc/inetd.conf
echo '0-59/15 * * * * root /opt/sbin/sendmail -q  & > /dev/null 2>&1' >>
/etc/crontab
```

Installing these base packages will automatically install their dependencies as well:

- for openssh : openssl, zlib
- for rsnapshot : coreutils, perl, rsync, openssh
- for syslog-ng : glib, eventlog
- for util-linux : ncurses (actually it is a 'suggests')
- for less : ncursesw
- for e2fsprogs : e2fslibs
- for nfs-utils : portmap, e2fsprogs
- for man : groff, less
- for screen : termcap
- for procps : ncurses
- for sendmail : procmail
- for logrotate : popt
- for nail : openssl

You can list dependencies with a command like this:

```
# ipkg info openssh
```

You need to re-login to make the new binaries in `/opt/bin` (installed with *coreutils*) available to you.

After installing OpenSSH, the sshd daemon is running. Verify that you can login and then disable telnet through the GUI: http://192.168.1.77/Management/telnet.cgi.

Configuring rsnapshot

- Configure the remote `rbackup` account for password-less login (see my rsnapshot wiki article). Note that for completely automatic login, the ssh key file *must* be called:

  ```
  /root/.ssh/id_rsa
  ```

  Verify that you can login from the *slug* as user `rbackup` on the remote server (in the examples I will call that server *mycomp.mylan.lan*):

  ```
  # ssh rbackup@mycomp.mylan.lan
  ```

  If the "`validate-rsync.sh`" script cannot be executed (issues with permissions, ownership) you will receive this error:

  ```
    bash: line 1: /home/rbackup/validate-rsync.sh: Permission denied
  ```

```
  Connection to mycomp.mylan.lan closed.
```

otherwise you will see this error:

```
  Rejected 3
  Connection to mycomp.mylan.lan closed.
```

Which means that all is setup properly for *rsnapshot*!

- Copy the `rsnapshot.conf` defaults file to your own (the unslung package automatically does that for us):

```
# cp /opt/etc/rsnapshot.conf.default /opt/etc/rsnapshot.conf
```

- Edit the configfile and check for correct syntax:

```
# /opt/bin/rsnapshot configtest
```

- After plugging in the big drive into port "*Disk 1*". Format it as `ext3` and tune it's `fsck` parameters (check what the drive's device is!!!!):

```
# umount /share/hdd/data/HDD_1_1_1
# mkfs.ext3 -m 0 /dev/sdb1
# tune2fs -c 0 -i 0 /dev/sdb1
```

- Make a root directory for the snapshot backups. We are going to use

```
/mnt/thevault/.private/.snapshots
```

as the root directory. Since we are mounting the big storage drive at

```
/mnt/thevault
```

, you will have to mount the disk and then create the directory structure

```
.private/.snapshots
```

on that disk's partition so that the rest of the instructions work:

```
mount /dev/sdb1 /mnt/thevault
mkdir -p /mnt/thevault/.private/.snapshots
```

- Add the following line to `/etc/fstab` so that the disk is available after reboot:

```
/dev/sdb1   /mnt/thevault   ext3    defaults,usrquota   1  1
```

NOTE: this disk is not formatted by the slug, so it will *not* show up in the slug's GUI!

- For ordinary users to access their backups *read-only* via NFS, we do the following:

```
# mkdir /mnt/thevault/.snapshots
```

Set the proper permissions on all directories:

```
# chmod 0700 /mnt/thevault/.private
# chmod 0755 /mnt/thevault/.private/.snapshots
# chmod 0755 /mnt/thevault/.snapshots
```

In `/opt/etc/exports`, add the directory `.private/.snapshots` as a read only NFS export:

```
  /mnt/thevault/.private/.snapshots  127.0.0.1(ro,no_root_squash)
```

In `/etc/fstab`, mount `.private/.snapshots` read-only under `.snapshots`

```
  localhost:/mnt/thevault/.private/.snapshots  /mnt/thevault/.snapshots
nfs  ro   0 0
```

Now restart the NFS daemon:

```
/opt/etc/init.d/S56nfs-utils condrestart
```

Now mount the read-only snapshot root (happens automatically at boot):

```
# mount /mnt/thevault/.snapshots
```

- In `/opt/etc/mail/aliases` add an email alias for user root so that you will receive emails generated from root's cronjobs.
- Add these lines to root's crontab (crontab -e) so that daily and weekly snapshots are being generated (you can add more as desired, and at other times of course):

```
0 */4 * * *       /usr/local/bin/rsnapshot hourly
30 23 * * *       /usr/local/bin/rsnapshot daily
0 23 * * *       /usr/local/bin/rsnapshot weekly
```

As you may notice, these crontab entries schedule the jobs with larger intervals to run a bit before the jobs that trigger more regularly.(*daily* runs 30 minutes before *hourly*; and *weekly* in turn runs 30 minutes before *daily*). This is to prevent for instance the *weekly* rsnapshot job to run before the *daily* job (the same goes for combinations of other intervals). If you would schedule them the other way round, a problem may arise in case the larger interval job would start (re)moving backup directories before the shorter interval job has finished it's work.

It seems that my changes to `/etc/fstab` are being overwritten at boot. However, there is an alternative to mounting through fstab: If the file `/unslung/rc.local` exists, it will be executed. So, I copied `/etc/rc.d/rc.local` to that file (it did not exist):

```
cp -a /etc/rc.d/rc.local /unslung/rc.local
```

removed the line

```
if ( [ -r /unslung/rc.local ] && . /unslung/rc.local ) ; then
return 0 ; fi
```

and then added these lines at the bottom:

```
mount -t ext3 -o defaults,usrquota /dev/sdb1 /mnt/thevault
mount  -t  nfs  localhost:/mnt/thevault/.private/.snapshots
/mnt/thevault/.snapshots
```

The rsnapshot log file

In `/opt/etc/rsnapshot.conf` I have uncommented the line

```
logfile         /opt/var/log/rsnapshot
```

and disabled the syslogging. This creates a logfile which grows over time (rsnapshot just appends to the file when it runs).
I use `logrotate` to keep that logfile trimmed on a weekly basis and keep a few weeks worth of log data. Add this to a new file called "`/opt/etc/logrotate.d/rsnapshot`":

```
/opt/var/log/rsnapshot {
    weekly
    rotate 4
    copytruncate
    compress
    notifempty
    missingok
}
```

I thought it would be nice to have the *slug* send me the log file through email. For this purpose, I have installed the *nail* package. The nail program which gets installed using "`ipkg install nail`" is incapable of invoking a local sendmail, so we have to instruct it to contact a remote SMTP server for email delivery, by creating a file `~/.mailrc` containing these two lines (set the "smtp" and "from" to values that make sense to your network):

```
set smtp=smtp-host.inyour.lan
set from=slug@inyour.lan
```

Then, create a cronjob that sends the rsnapshot logfile once a day (and pick a time at which it is likely that no rsnapshot process is running to avoid unfinished logs). This is what I added to the *slug*'s crontab file for root using "`crontab -e`":

```
0 12 * * *      echo "Mail from the slug." | /opt/bin/nail -a
```

```
/opt/var/log/rsnapshot -s "Rsnapshot activity log" you@inyour.lan
```

I have not managed to make the *slug*'s cron send me emails yet using sendmail ... all emails seem to piling up in the local mail spool. If anyone knows what I missed, let me know on the discussion page