Network UPS Tools

Jason Healy, Director of Networks and Systems

Last Updated Mar 18, 2008

Contents

Chapter 1

Network UPS Tools

Last updated 2008/03/18

1.1 Introduction

To prevent data loss and corruption, Suffield has several **U**niterruptable **P**ower **S**upplies (**UPS**es) that provide battery-backed power in the event of an electrical outage.

Most of our UPSes have a serial port, which allows a computer running monitoring software to receive status updates from the unit. This way, the host can shut down when the batteries of the unit run low.

Because each of our UPSes have more than one server attached to them, we needed a centralized way to manage the UPSes and shut down all affected servers during an outage. We settled on a software package called NUT, which stands for **N**etwork **U**PS **T**ools.

The software is easy to compile for UNIX-based operating systems, and there is also a Windows client available. NUT provides monitoring, reporting, alerts, and a client-server model that allows for very complex setups involving multiple servers and UPSes.

1.2 Suffield's Setup

At Suffield Academy, we have a single centralized host that acts as the master server for UPS information. It has a multi-port serial card, and all UPS units

are attached directly to it. All other hosts connect to this server via the network and poll it for UPS status information.

This centralized setup allows us to easily add new servers to our power configuration. We simply add client software to the machine and configure it to poll a particular UPS instance on the server. The server automatically broadcasts power outage events to the clients, which shut themselves down before the batteries are completely drained.

1.3 Server Setup

Suffield uses a setup where all the UPS units are connected to a single centralized server. This section deals with setting up such a centralized system, though it is relatively straightforward to adapt it to a multiple-master configuration as well.

1.3.1 Install the Software

Get and install the `nut` package. This means compiling it from source, or downloading a version through your distribution's packaging system. Note that some package systems break NUT up into several pieces (*e.g.*, drivers, client, and server), so you may need to install more than one package.

Under Debian Linux (our platform of choice), this ought to do it:

```
apt-get install nut
```

1.3.2 Port Identification

The first step is to get a list of the port devices on your server, and figure out which ones go with which UPS. For example, if you've plugged a UPS into serial port 1, it might be connected on `/dev/ttyS0` under Linux. If you have multiple serial ports (perhaps through an add-on board or other interface), you'll need to come up with a mapping of all the device names to the ports they use.

We highly recommend labelling the ports externally so that you may easily verify their device assignments.

You must change the permissions on all ports so they are accessible by the daemon running NUT. Under Debian, the files should be owned to user `nut` and group `nut`. Additionally, make sure no other users have write access to the device:

```
sudo chown nut:nut /dev/<device node>
```

```
sudo chmod 660 /dev/<device node>
```

1.3.3 Configuring the UPSes

Begin by editing the `ups.conf` file. This file defines all the UPS hardware that is directly connected to this machine.

For each UPS, you must specify the following:

- A name for the UPS
- The device it is connected to
- A short description
- When to shut down the load (`sdorder`)
- How to shut down the load (`sdtype` – certain models only)

The `sdorder` flag allows certain UPSes to be powered down before others. Use this to enforce an ordering on the powerdown sequence (lower numbers are powered down before higher numbers; -1 excludes from the sequence entirely).

The `sdtype` is special to the `apcsmart` driver (which we use with our APC UPSes). It determines the shutdown command to send to the UPS (*e.g.*, power off but don't restart automatically, power off now, power off with delay).

A sample stanza might look like:

```
[nicole]
    driver = apcsmart
    port = /dev/ttyS0
    # Set load to turn back on automatically when power returns
    sdtype = 0
    desc = "Smart-UPS 1000"
    sdorder = 10
```

Note: we name our UPS hardware after bratty spoiled people (the "SSW" naming scheme), which is why you'll see several women's names in our sample configs.

1.3.4 Configuration Access Controls

Once you've defined the hardware attached to the machine, you must now specify who has access to the hardware (via the `upsd` daemon).

Configuring Network Access

Begin by editing the `upsd.conf` file. This file allows you to specify ACL statements defining network hosts (or blocks).

You must add two lines for each host or netblock you wish grant access to. The first defines the host/netblock with a name, and the second grants (or revokes) access. Any configuration should include entries for the local machine, and a default deny:

```
# Define blocks
ACL localhost 127.0.0.1/32
ACL all 0.0.0.0/0
ACL myservers 172.16.0.0/12

# Grant or revoke access (deny by default)
ACCEPT localhost
ACCEPT myservers
REJECT all
```

Finer-grained control of who can log in from where is possible by combining ACLs with user-based permissions (see next section).

Configuring User Access

You may further restrict who has access to which UPS (and what they can do with it) by specifying users. In our case, the "division of labor" is quite clear-cut (a server which controls all the hardware, and numerous clients that may only poll for status). Thus, we only need a few users: one with full control, one that can monitor and take action on all UPSes, and one that can only monitor.

If your setup is more complex, you may need additional users. Additionally, you may wish to create individual users for particular machines for finer-grained control.

Here's a sample minimum configuration:

```
# Local user with full manual control over all UPSes
[superuser]
    password = supersecret
    allowfrom = localhost
    actions = SET FSD
    instcmds = all

# upsmon agent user with full control over all UPSes
# (used for local monitoring agent on master server)
[masteruser]
    password = secret
    allowfrom = localhost
```

```
    upsmon master

# upsmon agent with read-only privs
# (used for remote monitoring agents on other machines)
[slaveuser]
    password = kindasecret
    allowfrom = localhost myservers
    upsmon slave
```

1.3.5 Configuring Monitoring

The server is now configured to "listen" to several pieces of UPS hardware. We must now specify which UPSes this machine will monitor and interact with, so that we can automatically shut down machines, send pages and e-mails, and take other actions.

We do this by editing the `upsmon.conf` file. This file is quite large, so we'll take several steps to make all the edits necessary.

Monitoring

You must specify the UPSes you wish to monitor. Since we have a "one central server" approach, we add monitoring lines for **all** UPSes attached to the machine.

The central server should be allowed to interact with the UPS hardware (*i.e.*, turn it off and on), so we use our "master monitor" name and password for each UPS.

```
MONITOR nicole@localhost 1 masteruser secret master
MONITOR paris@localhost 1 masteruser secret master
```

Note that each `MONITOR` line must have a "power value" associated with it, specifying how many outlets on the current machine are powered by this UPS. Unfortunately, setting the power value to 0 means that the system will not automatically report status on the UPS. Thus, for a central master system, you'll need to assign a positive integer for every UPS that you're managing.

This complicates things a little, as now you have a machine that appears to be powered by many UPSes. The remedy is as follows:

- Set the power value of the UPS(es) that provide power to the server high enough so that it is greater than the power value of all the other UPSes combined.
- Set the `MINSUPPLIES` value to this high value.

The net effect is that the UPSes connected to the server will be "weighted" to count more than the other UPSes. This way, the system won't actually initiate a shutdown until it's lost the UPSes that are cruicial to its operation, but will still process events from the other UPSes.

External Processes

NUT has the ability to run an external program whenever an event is received from a UPS. To make this as flexible as possible, NUT includes a binary called `upssched` which processes events from the UPS and fires off a script of your choosing when they occur.

It's a little complicated because of this extra binary, but it works well and is quite flexible.

Begin by editing `upsmon.conf` and adding an entry for `NOTIFYCMD`. We use the built-in `upssched` binary, as it provides several useful features (like timers):

```
NOTIFYCMD /sbin/upssched
```

(Note that the location of the binary may be different on your system.)

Once you've done that, move down in the file and add entries for `NOTIFYFLAG`. We must tell `upsmon` what to do with each event it receives from a UPS; it can be any combination of the following:

- Write a message to the system log
- Write a message to all logged in users
- Execute the `NOTIFYCMD`
- Ignore the event

We've opted to log all messages to syslog and execute `upssched`:

```
NOTIFYFLAG ONLINE SYSLOG+EXEC
NOTIFYFLAG ONBATT SYSLOG+EXEC
NOTIFYFLAG LOWBATT SYSLOG+EXEC
NOTIFYFLAG FSD SYSLOG+WALL+EXEC
NOTIFYFLAG COMMOK SYSLOG+EXEC
NOTIFYFLAG COMMBAD SYSLOG+EXEC
NOTIFYFLAG SHUTDOWN SYSLOG+EXEC
NOTIFYFLAG REPLBATT SYSLOG+EXEC
NOTIFYFLAG NOCOMM SYSLOG+EXEC
```

Now, save your changes and edit the file `upssched.conf`. This file determins what `upssched` does when it receives an event. At first, the file seems a little limiting; you may only specify one script to execute for **all** events. However, it provides one very important feature which makes it worth it: **timers**.

`upssched` has the ability to trigger a **timer** for any event, and to cancel a timer that has not executed. For example, you could set a 30-second timer to send an e-mail when an UPS goes on battery, and then cancel the timer when the UPS goes back on line power. This gives you an easy way to "buffer" events before they get acted on, which is very useful for power "blips" (*e.g.*, outages lasting fewer than 30 seconds).

Begin by declaring the command to execute for all events. We have written our own script, called `upssched-dispatch` (download it from our website), which is written in Perl:

```
CMDSCRIPT /usr/local/bin/upssched-dispatch
```

You may need to specify values for `PIPEFN` and `LOCKFN`, depending on your operating system layout and configuration defaults. For example, on a Debian system you need the following:

```
PIPEFN /var/run/nut/upssched.pipe
LOCKFN /var/run/nut/upssched.lock
```

Finally, you must specify the events you wish to act on. We do this by specifying `AT` commands in the config file. The general form is:

```
AT <event> <upsname> <action> <arguments>
```

The event may be any UPS event (*e.g.*, `ONBATT`, `COMMBAD`). The upsname is the name@host identifier for the UPS. The action is one of `EXECUTE`, `START-TIMER`, or `CANCEL-TIMER`. The arguments get passed to your command script.

You need to pick an arrangement of actions such that you're notified of all the events you want to hear about. See our configuration file for an example of how we've done it.

One important note: you may use `*` in place of a particular UPS name. However, if you do so with a timer, you may find that some events get masked by others. If you're going to use timers, you should set a specific timer for each individual UPS.

1.3.6 Enable NUT

Now that NUT is configured, you're ready to turn it on. Under some systems, you may need to edit a file so NUT knows to officially launch and take over power management for your system.

Under Debian, you must edit the `/etc/default/nut` file and enable the UPSD and UPSMON subsystems.

Once you've done this, you can start the daemon, and check your syslog file to see if it's started properly. You should see status messages from all your UPSes.

1.3.7 NUT CGI

NUT includes one final piece of functionality: the ability to report UPS status via a web interface. Most packaged versions of NUT include this functionality as a separate install, so it may not be included with the standard NUT daemon.

The CGI is simple to configure; enable it as you would any other CGI (see your web server documentation for more info), and then add the files `hosts.conf`, `upsstats.html`, `upsstats-single.html`, and `upsset.html` to your configuration directory (you may omit `upsset.html` if you do not wish to allow editing UPS configurations via the web).

The `hosts.conf` file simply lists the UPSes you wish to make available via the web interface.

The HTML files contain a template file that is used by the CGI to generate pages about all (or one) UPS(es). The default files show a reasonable amount of information, though you may wish to edit them to your taste (to display additional information, or to alter the formatting).

1.4 Compiling NUT

Precompiled versions of NUT exist for many free operating systems (such as Linux or the BSDs). If you use one of these operating systems, consult your package manager to see if a ready-made version is available.

We have compiled our own versions of the NUT software for the operating systems that we use at Suffield (see previous sections for downloads). Here is a brief guide for compiling from scratch, geared towards Mac OS X.

1.4.1 Getting the Sources

There are three major versions of NUT available as of this writing:

- The "old" 1.4 series.
- The "stable" 2.0 series.
- The "unstable" 2.1 series.

While the 1.4 and 2.0 series are protocol-compatible (*i.e.*, a 1.4 client can talk to a 2.0 server), the configuration files and structure are **not** compatible. Thus, you should try to stick with the same major release across platforms. We currently use the stable 2.0 series.

Download the latest stable client from the NUT downloads page:

http://www.networkupstools.org/source.html

Unpack the distribution on your machine, and move into the top-level directory.

1.4.2 Add a User

For security purposes, NUT prefers to run under its own username. You should create a local username on your machine using the standard account creation tools. The user should not have login access to the account, so disable logins via a bogus password or other account locks.

We use `nut_upsmon` as the username for our custom installations.

1.4.3 Build the Software

In the top-level configuration directory, run the `configure` script. You may wish to provide options to `configure` to customize its behavior; here are the options we use:

- `--prefix=/usr/local/nut_upsmon`, which locates all the installed software to a single directory on the filesystem (makes for easier packaging).
- `--with-statepath=/usr/local/nut_upsmon/state`, which tells NUT to store driver information in the local software tree. Defaults to `/var/state`, but we move it to make it easier to track.
- `--with-user=nut_upsmon`, which is the user we created in the previous step.

- `--with-drivers=apcsmart`, which reduces the drivers we use. In reality, the clients probably won't need **any** drivers, as they'll be querying our master server. We throw the drivers in "for free" in case we need them (we only use APC model UPSes, so this is an easy choice). You should include only the UPSes that your site uses.

Once the configure step is done, run `make` to compile the software.

When `make` is finished, you may install the software directly by running `make install`, or you may divert the install to another directory for packaging. Since we build packages, we opt for the latter:

```
make DESTDIR=/tmp/nut_upsmon install
```

1.4.4 Configure the Software

You'll need to add configuration files to `/usr/local/nut_upsmon/etc/` for the clients. Because we run a centralized server, the clients should not need any configuration files dealing with the "server" parts of NUT. At a minimum, you'll need an `upsmon.conf` file (specifying a remote UPS to monitor), and you may optionally want an `upssched.conf` file if you wish to perform additional tasks (such as early shutdowns).

Our setup involves three custom files, which you may download below:

- `upsmon.conf` (download), which specifies a remote server to query for UPS state information.
- `upssched.conf` (download), which specifies a special script (`early-shutdown`; see below) to execute when a UPS has been on battery for a certain amount of time. This allows certain machines to be shutdown before the battery goes critical.
- `early-shutdown` (download), which is a script that receives events from `upssched` and shuts the machine down early.

1.4.5 Launch the Software

The software should be configured to run on boot. For UNIX-based systems, this probably involves editing the init scripts on the platform.

For Mac OS X, we prefer to use `launchd`, which is Apple's new way of automatically running software starting with 10.4. Unfortunately, `nut` doesn't play nice with `launchd` by default, as it detaches and forks from the controlling terminal.

We've written a simple wrapper script that sits between `launchd` and `upsmon` (the client part of `nut`). It's called `upsmon_launchd_wrapper`, and you can download it from our site. You'll also need the launchd plist file, which should be placed in the `/Library/LaunchDaemons` folder and activated.

1.5 Replacing Batteries

1.5.1 When to Replace?

Ordinarily, the UPS will let you know when it thinks it's time to replace its battery. Most UPSes self-test on a regular basis, and if one of these tests shows that the battery isn't holding a proper charge, a REPLBATT message will be issued by NUT. **Do not ignore these messages!** Batteries that don't hold a charge may provide little (or no) runtime.

1.5.2 Replacing the Battery

Order a replacement battery for the UPS. In the case of APC Smart-UPSes, the batteries can be swapped while the UPS is powered on. Simply unscrew the bezel, pull the large power connectors, and remove the old battery. Replace with the new battery and reverse the steps to close the UPS up.

1.5.3 Updating the Battery Date Variables

APC UPSes have internal variables that allow you to keep track of when the battery was installed. Unfortunately, these variables are not set automatically, so you'll need to freshen them up when you add the new battery.

To do this, run the `upsrw` command, and provide the date (you'll need the master user password):

```
upsrw -s "battery.date=2007Apr1" -u upsmaster "myups@localhost"
```

We use a textual month to prevent ambiguity in date format.

1.5.4 Refreshing UPS Battery Data

Once the battery is installed, you'll need to let the UPS know that there's a new battery. **APC recommends waiting at least 8 hours for the battery**

to get a full charge before you re-run the self test. Once 8 hours have elapsed, issue a self-test and the "replace battery" light should go out.

Unfortunately, this process isn't perfect, and sometimes the UPS will still think it needs a new battery. This can manifest in one of two ways:

1. The UPS continues to fire a REPLBATT event (the red light on the front panel remains lit)
2. The UPS reports a "low battery" condition, even though it is on line power and not over load. You will note that the estimated runtime is lower than it should be (possibly only 1-4 minutes).

If this happens, try the following until the situation is resolved:

The Easy Way

Issue a self-test on the UPS, by holding in the test button (or starting the test from software). The "replace battery" light should go out after the test.

The Hard Way

If the self-test doesn't seem to "take", you can perform a runtime calibration. You can start this by issuing an ups command:

```
upscmd -u upsmaster -p <password> myups@localhost calibrate.start
```

The calibration process must be started with the ups at a 100% charge! The calibration also must have a load attached (if you're nervous about production machines being attached, find a "dummy" load to run.

The process puts the UPS on battery, and drains the battery to 25% before automatically switching to line power. The UPS uses the load data and time to compute the new runtime estimate.

The Really Hard Way

If a software calibration doesn't help, then a forced calibration may be necessary. The following are taken from the NUT FAQ (http://www.networkupstools.org/faq/) and also from phone instructions given by an APC technition.

1. Disconnect any serial monitoring cables (we don't want the server interfering with the UPS)

2. Hook up a dummy load (we're going to drain the UPS completely, so don't use any production machines). Ideally, the load should be about 30% of the UPS capacity (2 out of 5 LEDs lit on the front).

3. With the load attached and stable, pull the plug. The NUT FAQ says that the UPS must remain grounded, but APC didn't mention this. If you want to stay grounded, keep the UPS plugged in, but attach it to a device that can cut the circuit without disconnecting from ground (a surge protector, perhaps).

4. Kick back and wait for the UPS to drain completely. It will beep a lot; just go do something else for a while and let it drain.

5. Once the UPS has drained, plug it back in and turn it back on. The UPS should now have re-calibrated itself with the runtime data.