
# Mesh Lab Manual

Version 0.21
November 2008

Original written by David Johnson
Revised by Kevin Duff

# Table of Contents

# 1 Getting Started

This section is intended to give you all the information you need to get started and gain access to the lab.

## 1.1 Network Access at the Meraka Institute

It is easy to use your laptop on Meraka's network. Plug your laptop into any ethernet point, enable DHCP, and you should have network connectivity and Internet Access. No proxy configuration is required.

## 1.2 Access to the Mesh Server

The mesh server runs FreeBSD. You can ask anybody who has root access on this machine to create an account for you. The machine is administered by John Hay.

If you require root access to the machine, you'll need to be a member of the wheel group.

Once you have credentials, you can log into the server as follows:

```
ssh username@meshy.dhcp.meraka.csir.co.za
```

This can normally be abbreviated to:

```
ssh meshy.dhcp
```

## 1.3 Access to the Visualization Server

The visualization server, meshlab.dhcp.meraka.csir.co.za, server runs Linux. Acquire credentials as described above. You'll need somebody to update /etc/sudoers if you need root access.

## 1.4 Accessing the Mesh Remotely

The mesh server and visualization server are not visible from the Internet, nor any network outside of Meraka. If you require remote access, you'll need an account on a machine on Meraka's network with a live IP.

You can request a login to zibbi.meraka.csir.co.za from John Hay (j.hay@csir.co.za).

## 1.5 Etiquette and Sharing the Mesh

Multiple users are able to simultaneously log into the mesh server. Mesh users need to be aware that other people might be running experiments, and that they should be careful not to interrupt those experiments.

Please check the login message, and check who else is logged in, before you do anything rash!

### 1.5.1 Checking Who Else is Logged Into The Server

To obtain a list of users logged into the FreeBSD Server, just type:

```
users
```

### 1.5.2 Sending an Instant Message to Other Users

You can use the wall command to send a message to every user on a system. General syntax of wall command is as follows:

```
wall
Message
Message
...
....
..
```

When the message is complete, **press Control-D**.

### 1.5.3 Modifying the Login Message

If you are running an experiment over an extended period of hours, you might want to notify anybody who subsequently logs into the server.

For example, you might want to say:
“Kevin Duff will be running experiments from 8h00 to 23h55 on 31/12/08. Please do not interfere with the mesh during this period. My contact details are k.duff@somewhere.net and +27-(0)82-123-4567”

Each time a user logs in to a system, the message in the file /etc/motd is displayed. To modify the message:

1. Login as root or become root user using su
2. Use text editor vi to edit /etc/motd:

   # vi /etc/motd

3. Edit and save the changes. Next time a user logs in they will see the message.
4. Don't forget to remove your message once your experiment is finished!

# 2 Notation used in this lab manual

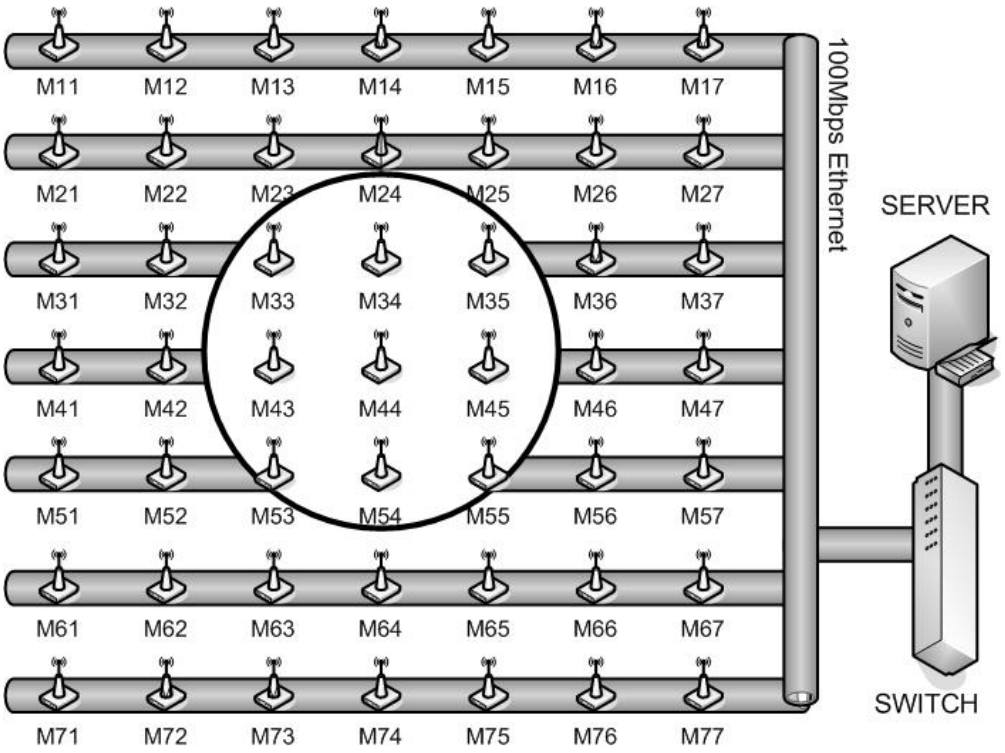
When describing commands the following notation is used

**`command <arg1> <arg2> [arg3]`**

Any argument contained in angular brackets e.g. <arg1> is a compulsory argument
Any argument contained in square brackets e.g. [arg3] is an optional argument

# 3 Description of the lab setup

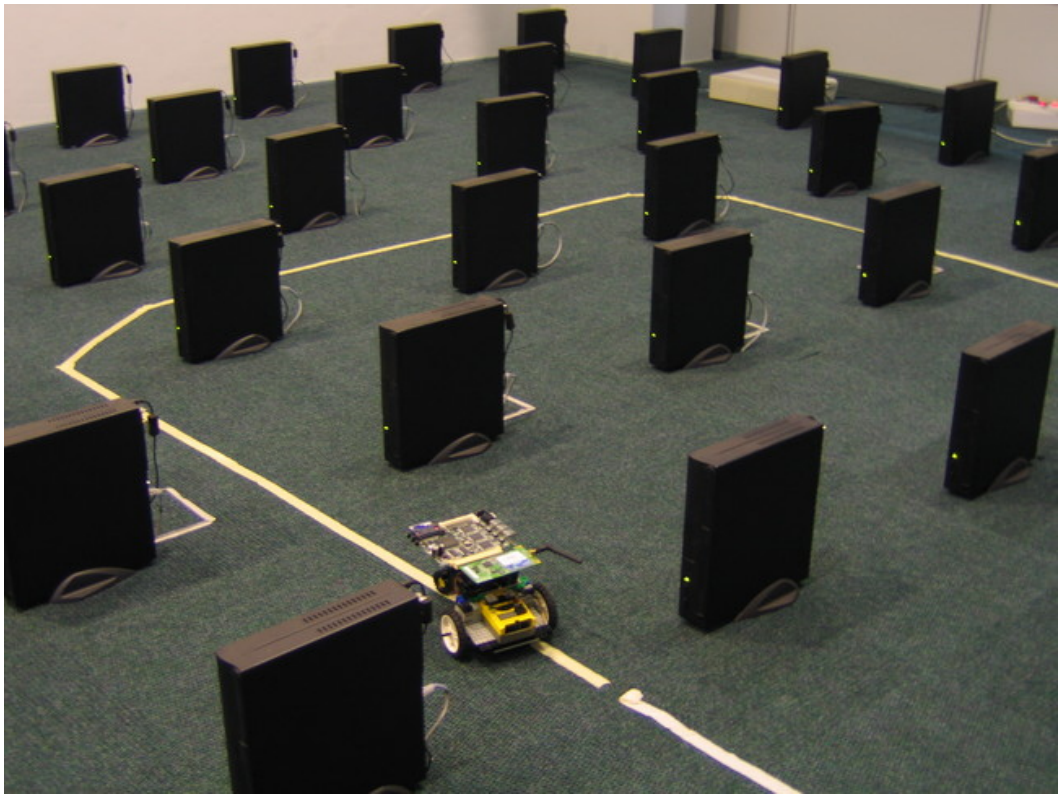
**Figure 1: Architecture of the mesh lab**




The mesh test bed consists of a wireless 7x7 grid of 49 nodes, which was built in a 6x12 m room as shown in Figure . A grid was chosen as the logical topology of the wireless test bed due to its ability to create a fully connected dense mesh network and the possibility of creating a large variety of other topologies by selectively switching on particular nodes.

Each node in the mesh consists of a VIA 800 C3 800MHz motherboard with 128MB of RAM and a Wistron CM9 mini PCI Atheros 5213 based Wi-Fi card with 802.11 a/b/g capability. For future mobility measurements, a Lego Mindstorms robot with a battery powered Soekris motherboard containing an 802.11a (5.8 GHz) WNIC and an 802.11 b/g (2.4 GHz) WNIC shown in Figure 2 can be used.

**Figure 2: The mesh grid with a line following robot**



Every node was connected to a 100 Mbit back haul Ethernet network through a switch to a central server, as shown in Figure 1. This allows nodes to use a combination of a Pre-boot Execution Environment (PXE), built into most BIOS firmware, to boot the kernel and a Network File System (NFS) to load the file system.

The physical constraints of the room, with the shortest length being 7m, means that the grid spacing needs to be about 800 mm to comfortably fit all the PC's within the room dimensions. At each node, an antenna with 5 dBi gain is connected to the wireless network adapter via a 30 dB attenuator. This introduces a path loss of 60 dB between the sending node and the receiving node. Reducing the radio signal to force a multi hop environment, is the core to the success of this wireless grid.

The wireless NICs that are used in this grid have a wide range of options that can be configured:

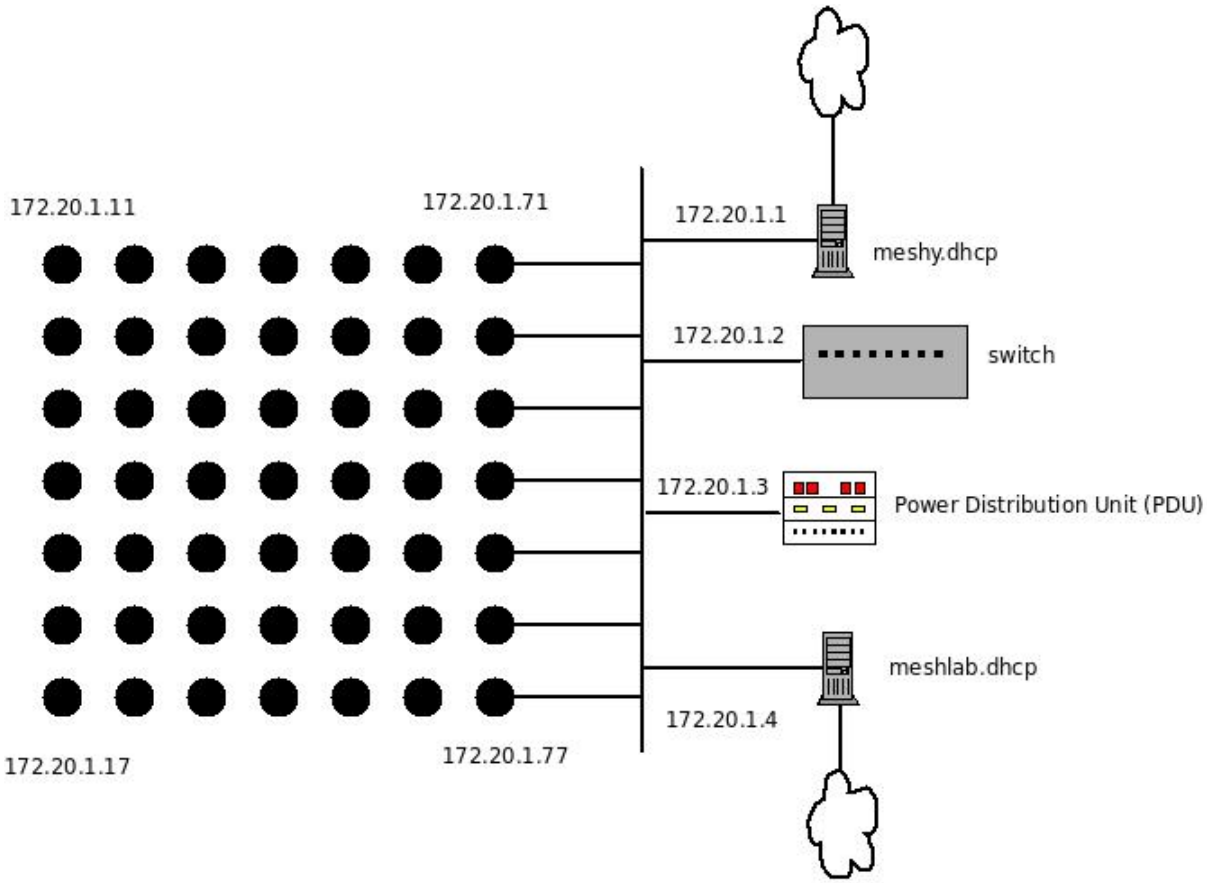
- Power level range: The output power level can be set from 0 dBm up to 19 dBm.
- Protocol modes: 802.11g and 802.11b modes are available in the 2.4 GHz range and 802.11a modes are available in the 5 GHz range
- Sending rates: 802.11b allows the sending rate to be set between 1 Mbps and 11 Mbps and 802.11g allows between 6 Mbps and 54 Mbps

This network was operated at 2.4 GHz due to the availability of antennas and attenuators at that frequency, but in future the laboratory will be migrated to the 5 GHz range, which has many more available channels with a far lower probability of being affected by interference.

# 4 Architecture of the laboratory

There are currently 2 servers in the lab which are connected to the grid. The first server, *meshy*, which runs FreeBSD is used to hold the operating system of all the nodes. Each of the nodes boot their operating system off this server. The second server, *meshlab*, which runs Ubuntu is used to host some of the monitoring and viualisation software to allow users to monitor the status of each node and their connectivity with other nodes.

The network structure is as follows: It is organised such that the last octet of the IP address is the same as the xy grid position.




Most of the software for the mesh lab is in the form of perl or shell scripts which have been written to simplify the process of controlling nodes in the lab or carrying out measurements in the lab. All the commands discussed in this manual are in the form of such scripts and they can be examined by browsing the /home/djohnson/bin directory.

## 4.1 The Free BSD server

Name: meshy,dhcp.meraka.csir.co.za

This server hosts:

- DHCP server which nodes use to allocate an IP address – based on a lookup table which maps MAC addresses of the NIC's to an IP address
- PXE bootloaders which nodes in the grid boot off to start the node's operating systems
- A number of operating systems in the /export directory such as Ubuntu Linux and freebsd operating systems. These can be booted by each node in the grid.

This machine has two network interfaces

Interface 1 (connected to icomtek intranet): meshy.dhcp.meraka.csir.co.za
Interface 2 (connected to the indoor mesh): 172.20.1.1

root password: none
To become root just type

```
su
```

When it asks for a Password just press [Enter]

## 4.2 The Linux Server (Ubuntu and Fedora)

Name: meshlab.dhcp.meraka.csir.co.za

The ubuntu partition on the server hosts:

- Visualization software
- Network monitoring software

The Fedora partition on the server hosts:

- ns2 network simulator
- omnet++ network simulator
- nctuns network simulator

This machine has two interfaces

Interface 1 (connected to icomtek intranet): meshlab.dhcp.meraka.csir.co.za
Interface 2 (connected to the indoor mesh): 172.20.1.2

root password on both Ubuntu and Fedora: eckels

# 5 How the nodes boot their operating system

The nodes are only equipped with 128M of flash memory and 256M of ram and can either boot their operating system of the flash memory or using their built in network interface.

## 5.1 Network booting

This is the default boot mode of the nodes. The Bios has been setup so that when a node starts it will automatically try to boot off the network card first and then try to boot off the flash card if the network boot fails.

Here are the steps during the boot process

1. Node in the mesh issues a DHCP request with PXE extension flags to get an IP address and the IP address of the boot server
2. DHCP Server replies with IP address for node (we have created a mapping of Lan MAC addresses to IP addresses in the DHCP table to ensure that the same IP address is always handed out based on position in the gird)
3. Boot server which happens to be the same as the DHCP in our case (meshy) replies to the node with the Boot server IP address.
4. Node then request sthe filename of the network bootstrap program and issues a special DHCP request called a Boot Image Negotiation Layer (BINL) request
5. Boot server replies with a filename of the network bootstrap program – to understand where this filename is set – see file /usr/local/etc/dhcpd.conf
    1. In the case of booting a FreeBSD node this has been set to /tftpboot/pxeboot
    2. In the case of booting a Linux node this has been set to /tftpboot/pxelinux.0
6. Node requests bootstrap program from server via TFTP
7. Boot program sends network boot program to PC via TFTP
8. Node stores network bootstrap program in memory. Node executes bootstrap program and it can now download further files it may need from the server and the whole boot process then begins ... fetching the kernel etc.

There are some important files to understand should you wish to change the operating system that you want to boot.

### 5.1.1 The main dhcp configuration file ... /usr/local/etc/dhcpd.conf

If you want to add a new operating system that the nodes can boot or you want to change the location of an existing operating system on the server then you need to edit this file. Look for the sections called class. Each of this classes specify the pxeboot program as well as the root path of the operating system. If you want to create a new operating system for the nodes. Just copy and paste the whole class section of another operating system and only make changes to the class name, the filename of the pxeboot program (shouldn't need to change this) and the option root-path entry.

### 5.1.2 The location of the operating systems

The operating systems on the meshy server are all stored in the /export directory. The following operating systems are installed

1. /export/linux/hardcore ... A customized 2.6 linux developed by Elektra and David with a very bare bones Linux making use of busybox ... small enough to fit into flash
2. /export/linux/ubuntu_dapper ... The ubuntu linux system with compile tools and sources included for compiling software on the node. Kernel version 2.6.16.
3. /export/linux/hardapper3 ... A copy of ubuntu_dapper with a real-time kernel, upgraded to 2.6.24.
4. /export/freebsd/6-stable ... FreeBSD stable version
5. /exportfreebsd/7-stab le ... FreeBSD current unstalbe release

### 5.1.3 Commands to change the operating system

There is a script which automates a lot of the changes needed in the dhcp config files called **`mk-dhcp-classes.`** This command is used to change the operating system that is booted on each node. It can be used to change the booted operating system on selection of nodes or on all the nodes.

The syntax for **`mk-dhcp-classes`** is:

```
mk-dhcp-classes [-d <default_class>] <range> <class> ...
- The default class is fbsd6
- range is in the form: x1y1-x2y2
- current available classes: fbsd6-ndis, fbsd6, fbsd7,
linux-dapper, linux-gutsy, linux-nfs, linux-hardcore
```

**Example 1**: Half the grid (rows 1-3) runs fbsd6 FreeBSD and half the gird (rows 4-7) runs ubuntu-dapper linux

```
mk-dhcp-classes 11-37 fbsd6 41-77 linux-dapper
```

**Example 2**: The whole grid runs fbsd7 FreeBSD except for node 71 which runs hardcore linux

```
mk-dhcp-classes -f fbsd7 71 linux-hardcore
```

### 5.1.4 Booting the correct linux – the pxelinux.cfg file

If you change the version of linux you are booting from a previous boot with the mk-dhcp-classes program, you also need to edit the file /export/tftpboot/pxelinux.cfg/default which contains a line with the default linux kernel it will start as well as some parameters that are passed to it. Just uncomment the correct operating system from the list. For example if you changed to linux hardcore you would uncomment these lines

```
DEFAULT bzImage-hardcore
APPEND ip=dhcp root=/dev/nfs
nfsroot=172.20.1.1:/export/linux/hardcore ramdisk_
size=16432 console=ttyS0,115200
#TIMEOUT 100
```

### 5.1.5 Restarting the dhcp server

Once you have followed all these steps to change the operating system which the nodes boot, you will want to restart the dhcp server for the changes to take effect.

```
/usr/local/etc/rc.d/isc-dhcpd.sh restart
```

## 5.2 Flash booting

In order to boot off the flash disk – the simplest method is to disable the dhcp server so that the PXE bootloader gets no IP addresses leases from the server. Do do this simply run the command

```
/usr/local/etc/rc.d/isc-dhcpd.sh stop
```

# 6 Controlling the nodes

Once the nodes have been booted, there are a number of useful commands that are able to control the nodes from the server.

## 6.1 Powering up and powering down the nodes

As described in the architecture of the lab, the nodes are powered up via a relay bank which is able to switch on columns of nodes , where each column consists of 7 nodes. The realy bank is attached to the FreeBSD server, meshy, and is controlled by a piece of software called *meshpower*

The syntax for meshpower is:

```
meshpower status
meshpower <bank|all> <on|off> [time(s)]
```

If the **`status`** option is used then the command will return the current status of the relay bank reporting which banks are on and which banks are off, with an output that looks as follows:

```
<wait for mesh lab to be live>
```

If the **`<bank|all> <on|off> [time(s)]`** options are used then the following occurs

Argument 1: Either a number from **1 to 7** which represents a bank or column in the grid is specified or the term **all** is used. Using a number from 1 to 7 will switch on or off that specific column. Using the term **all** will switch on of off all columns in the grid in sequential order from bank 1 to bank 7.

Argument 2: **on** or **off** is specified depending whether you want to turn a bank or column on of off.

Argument 3: A delay time in seconds can be specified. This is the time interval between adjacent banks being turned on or off. If no argument is given, the default delay time is 10 seconds. Specifying longer delay intervals is often needed to prevent the server being overwhelmed by too many dhcp requests or too many nfs (network file system) mounts at the same time.

Some examples of using this command are

**example1**: `meshpower 2 off`
turns off bank 2.

**example2**: `meshpower all on 30`
turns on all banks with a 30 second interval between rows switching on

## 6.2 Logging into the mesh nodes

Once the nodes are powered up and their operating systems are booted, they usually usually start two important services: **ssh server** to allow you to login to the nodes using a secure shell and **nc** to allow you to send remote commands to the nodes.

To login to a particualr node at a specific xy coordinate use the following command

```
ssh root@172.20.1.xy
```

a short hand script has also been created which allows you to abbreviate this to

```
msh <xy>
```

So to log into a node at position 3,3 you would use the command

```
msh 33
```

You will then be logged into the node and can issue any unix command at the command line which will be executed on that node.

## 6.3 Sending remote commands to the nodes

There are often times where you may want to execute the same command on many nodes at once to avoid logging into each node individually and issuing the same command on each node.

A script has been written to carry out this task which makes use of the nc (netcat) program which sends commands on a specific port to the node which is listening for commmands on that port.

To execute a remote command, execute

```
lexec |<all>| or |<range>| or |<x><y>| <command>
```

The first argument is the range of nodes to execute the command on or a specific node to execute the command on. The all argument can also be used to specify all the nodes in the grid.

If a range is used then the syntax is $<x_1y_1\text{-}x_2y_2>$

**Example 1**: Making all the nodes change to channel 6

```
lexec all "iwconfig ath0 channel 6"
```

**Example 2**: Rebooting node 24

```
lexec 24 "iwconfig ath0 channel 6"
```

**Example 3**: Turning off the radios on bank 4 to 7

```
lexec 41-77 "ifconfig ath0 down"
```

## 6.4 Configuring Wireless Devices

You'll probably want to configure wireless devices appropriately before you can use them.

A handy script is installed on meshy.dhcp called /root/bin/shakemesh. This uses lexec to configure wireless settings on the entire mesh. This script requires netcat to be installed on all nodes.

```
/root/bin/shakemesh
```

Another script is available which can be run locally on nodes. Use it thus:

```
/export/linux/ubuntu_dapper/root/shakenode ath0
```

You can copy then modify either of these scripts to suit your needs.

# 7 Monitoring the Mesh

## 7.1 Monitoring via a Serial Cable

The RS-232 serial cable provides a useful way to monitor the console log. It is especially useful for diagnosing booting and start-up problems. It is very handy when you don't have physical access to the lab.

The serial cable connects the visualization server, meshlab.dhcp.meraka.csir.co.za, to one of the mesh nodes. The cable is normally connected to Node 47.

To monitor a node via the serial cable:

1. Ensure that the serial cable is connected between meshlab and the machine you wish to monitor.
2. Open the file /export/tftpboot/pxelinux.cfg/default .
3. Locate the correct section for the operating system you wish to boot.
4. You should see a line which looks something like:
   ```
   APPEND ip=dhcp root=/dev/nfs
   nfsroot=172.20.1.1:/export/linux/ubuntu_dapper ram
   disk_size=16432 initrd=initrd.img-2.6.12
   ```
5. For serial monitoring, the line should end with:
   ```
   console=ttyS0,115200
   ```
   Append this string if it is not present.

Now log into meshlab.dhcp and start a terminal emulator. You can use either minicom (which needs to be configured before you use it) or picocom:

```
picocom -b 115200 /dev/ttyS0
```

If you reboot node 47, after about one minute you should see information appearing in the terminal emulator as the node starts up.

Only one person at a time can monitor the serial port. If you attempt to start picocom while somebody else is running an instance, you'll get an error like:

FATAL: cannot lock /dev/ttyS0: File exists

If you get such an error, first confirm that nobody else is presently using the mesh. If somebody has left picocom running but is not presently working, you can kill their process by issuing the following command:

```
sudo pkill picocom
```

# 8 Installing Software on the Nodes

This section describes procedures for installing new software on the nodes. It was written for the ubuntu_dapper installation; for other versions or operating systems the procedure will be slightly different.

## 8.1 General Information

Because the nodes mount much of their filing system (/etc) on volatile RAMdisks, one cannot use package managers as in a usual installation. Here's how we do it:

## 8.2 Mounting an NFS Share from your Linux Box

1. Start a Terminal on a Linux machine. You can use your personal computer, if it runs Linux, or the visualization machine meshlab.dhcp. The machine will need to have Internet access if you wish to download anything. You'll also need an NFS client installed on your machine.

2. Create a folder to mount the NFS share, if you haven't done so already:

   ```
   mkdir /export
   ```

3. Mount the share:

   ```
   mount meshy.dhcp.meraka.csir.co.za:/export /export
   ```

4. You can now read and write files on the server. When needed, you can chroot into the environment to perform maintenance:

   ```
   chroot /export/linux/ubuntu_dapper
   ```

If you wish to compile software on the physical node, you can log into one of the nodes and use the same procedure.

## 8.3 Modifying files in the Node's /etc Directory

Under ubuntu_dapper, the nodes mount their /etc directory to a RAMdisk. Thus anything a node writes to /etc will be lost after the next reboot.

Files in /etc are copied from the node's /conf directory during startup. To make changes to any of the files in /etc, you'll need to modify the relevant file in /conf. For example, to edit the /etc/apt/sources.list file, use a command like the following on your Linux machine:

```
edit
/export/linux/ubuntu_dapper/conf/base/etc/apt/sources.lst
```

You'll need to reboot the nodes to make them recopy /etc, before any changes take effect.

## 8.4 Installing a Package from Source

1. Download source.
2. Extract archive and copy to a folder in /export/linux/ubuntu_dapper/usr/src
3. Chroot into the new environment:

```
chroot /export/linux/ubuntu_dapper
```

4. Compile and install the package. Refer to package documentation; in general the procedure is:

```
make clean
make
make install
```

5. Reboot nodes.

## 8.5 Installing a Package from an Ubuntu Repository

Because mesh nodes do not have Internet access, and because much of their filing system is mounted on a volatile RAMdisk, one cannot simply use apt_get on a node as one would expect.

We do it by using a separate machine to download packages, then we use one of the nodes to install the package. To download a package, do the following on your Linux box:

1. Chroot to the tartget environment:

```
chroot /export/linux/ubuntu_dapper
```

2. You might want to check that your repositories are correctly configured:

```
less /etc/apt/sources.list
```

3. Use apt-get to download the package, but don't install it. The -d switch tells aptitude to download only:

```
apt-get -d install packagename
```

Now that the package is downloaded, you'll use a node to install it:

4. SSH into a node (we use node 11 for example):

```
msh 11
```

5. Mount the NFS share as described above, then chroot:

```
chroot /export/linux/ubuntu_dapper
```

6. Because the package is already downloaded, the node will not need to access the Internet. Use apt-get as usual to install the package:

   ```
   apt-get install packagename
   ```

7. Reboot nodes.

## 8.6 How to flash a node in the massive mesh

1. Create a tar image of the linux file system. Change to the root folder of the linux filesystem:

   ```
   tar cvf .../hardcore-linux.tar .
   ```

2. Make an image of the linux file system on the flash card:

   ```
   dd if=/dev/sdb of=hardcore-linux.img
   ```

3. Create a partition on the compact flash card:

   ```
   cfdisk /dev/sdb
   ```

4. Make the file system:

   ```
   mkfs.ext3 /dev/sdb1
   tune2fs -c o /dec/sdb1
   ```

5. Untar file system to flash drive:

   ```
   tar xvf hardcore-linux.tar -C /media/disk
   ```

6. Start qemu with info to boot:

   ```
   qemu -append root=/dev/hda1 -kernel/tmp/bzImage /dev/sdb
   ```

7. Run lilo on flash system once qemu is launched:

   ```
   lilo
   ```

8. Now test the flash card:
   ```
   qemu /dev/sdb
   ```

## 8.7 Installing a New Operating System

This is not for the faint of heart! Ubuntu does not have a facility to install to an NFS share for NFS booting, so we had to do it by hand. Here are a couple of pointers.

1. First install to your local machine. Assuming your machine is an x86, the architecture will be similar and you should be able to copy the entire installation to the network once it is ready.
2. We were unable to install directly to an NFS share, as the installer complained about locking problems. There are known issues related to locking when an NFS share is mounted to a BSD server.
3. You might want to read up on debootstrap, a Debian utility. debootstrap is used to create a Debian base system from scratch, without requiring the availability of dpkg or apt. It does this by downloading .deb files from a mirror site, and carefully unpacking them into a directory which can eventually be chrooted into
4. If you don't want to start from scratch, you could try making a copy of one of the existing installations and modifying or upgrading what you need.
5. Don't forget to update PXELinux.cfg, dhcpd.conf and run mk_dhcp_classes, as described earlier in this document.
6. You'll probably want to have a look at the init scripts of the existing installations, which do some tricks. Below are some notes from David in that regard:

### 8.7.1 Trick 1:

When I built dapper I had some startup scripts basically copy files from folders which where labelled according to the nodes IP address - remember at least it starts off with an ethernet address which it got from the DHCP server (always the same - because we locked MAC addresses to IP's in the DHCP server).

Have a look at:

/export/linux/ubuntu_dapper/etc/rc2.d/S15diskless.sh

I did everything at run level 2 so you may want to scratch around in there for hints.

### 8.7.2 Trick 2:

When Elektra and I built linux hardcore - we thought about the problem of uniqueness and realized that it's only a small few things that are unique between nodes such as ip address of ethernet and wifi ... so we used a startup script which generated these on the fly. Have a look at:

/export/linux/hardcore/rcS

This is a real bare bones linux - it doesn't even use init.d scripts just one single start script - this one :)

# 9 Layout of File System

This chapter describes the layout of important parts of the file system on the mesh server, meshy.dhcp.meraka.csir.co.za.

Important directories are listed in the table below.

| Directory | Contains |
|---|---|
| /root/sbin | Script files for controlling the mesh |
| /home/djohnson | Further sample script files |
| | |
| /export/linux | Various Linux versions |
| /export/freebsd | FreeBSD 6 and 7 |
| /export/tftpboot | PXE and Netbooting Files |
| | |
| /export/linux/ubuntu_dapper/conf | Configuration files (/etc) copied at startup for ubuntu_dapper |
| /export/linux/ubuntu_dapper/conf/base/etc | /etc folder for nodes under ubuntu_dapper |
| /export/linux/ubuntu_dapper/etc/rc2.d/ | Startup scripts for ubuntu_dapper |

# 10 Additional Information

The information in this section is a summary of numerous emails, exchanged between Kevin Duff and David Johnson. In working on the mesh, Kevin discovered numerous idiosyncrasies which caused him difficulty. In most cases David was able to provide a solution and Kevin has documented these in the hope of making life easier for subsequent researchers.

## 10.1 How to Disable A Wireless Device

Wireless devices can be disabled using the ifconfig command:

```
ifconfig ath0 down
```

However, we are presently uncertain as to whether or not this command will entirely disable the radio and prevent interference.

Below is an extract from the man page for iwconfig:

txpower

For cards supporting multiple transmit powers, sets the transmit power in dBm. If W is the power in Watt, the power in dBm is P = 30 + 10.log(W). If the value is postfixed by mW, it will be automatically converted to dBm.
**In addition, on and off enable and disable the radio,** and auto and fixed enable and disable power control (if those features are available).

I first tried:

```
iwconfig ath0 txpower off
```

and the driver seemed to ignore this. Then I tried:

```
iwconfig ath0 txpower 0
```

and hey presto - the iwconfig status output says:

```
Bit Rate=11 Mb/s   Tx-Power=off   Sensitivity=1/1
```

That ought to disabled the radio.

## 10.2 Multi-radio Nodes

Every alternate node in the mesh is equipped with multiple radios. The nodes with multiple radios are:

| | | | |
|---|---|---|---|
| 11 | 13 | 15 | 17 |
| 31 | 33 | 35 | 37 |
| 51 | 53 | 55 | 57 |
| 71 | 73 | 75 | 77 |

Each of these nodes has three 802.11 wireless cards installed. However, at the time of writing only one card in each node was connected to an external antenna; the remaining two cards have no antennae connected.

### 10.2.1 Hardware Differences between Multi-radio and Single-radio Nodes

The single-radio nodes are all equipped with Wistron CM9 mini PCI Wireless cards. The CM9s are installed on single-slot mini PCI adapters.

The Wistron CM9 is an 802.11a/b/g Dualband mPCI card, using the Atheros 5004 chipset with the 5213 as main chip.

In the case of multi-radio nodes, Routerboard 14 mini PCI adapters are used, which have 4 mini PCI slots to accept up to 4 wireless cards. Three wireless cards are installed, they are MicroTik R52 mini PCI wireless cards.

The MikroTik R52 is a wireless 802.11a/b/g miniPCI card for multiband high speed applications, and uses the Atheros AR5414 chipset

There there is a compatibility problem between the CM9 cards and the Routerboard 14 when the MadWiFi driver is used, thus CM9 cards cannot be used on multi-radio nodes.

### 10.2.2 Caveat: Numbering of athx Devices under udev

**udev** is the device manager for the Linux 2.6 kernel series. Primarily, it manages device nodes in /dev. It is the successor of devfs and hotplug, which means that it handles the /dev directory and all user-space actions when adding/removing devices.

udev provides the ability to have persistent device names. However, we discovered when upgrading from kernel version 2.6.16 (ubuntu_dapper) to 2.6.24 (ubuntu_hardapper) that the numbering of ath0..ath3 was changed, resulting in confusion on the nodes.

For the hardapper3 installation, Kevin hacked the S15diskless.sh startup script to delete all athx devices, then recreate ath0.

udev provides a facility to identify a wireless interface by its MAC address, however this is not presently utilised on the mesh.

In summary: If you discover that a node cannot communicate wireless with other nodes, there is a good chance that the wrong wireless card (with no external antenna) is being used. You'll need to try communication on other devices (ath1,ath1) to confirm this.

## 10.3 MadWiFi Version & Packet Size Problem

MadWifi is short for *Multiband Atheros Driver for Wireless Fidelity*. In other words: it is a Linux kernel device driver for Atheros-based Wireless LAN devices.

The following problem was experienced under the ubuntu_dapper installation:

*I did some diagnostics using ping. For some reason, I can't ping with packet sizes larger than 196 bytes:*

```
root@mesh25:~# ping -s 195 172.30.1.26

PING 172.30.1.26 (172.30.1.26) 195(223) bytes of data.
203 bytes from 172.30.1.26: icmp_seq=1 ttl=64 time=1.09
ms...
```

*Whereas, if I use a larger packet size ping just sits and sits, doing nothing:*

```
root@mesh25:~# ping -s 197 172.30.1.26

PING 172.30.1.26 (172.30.1.26) 197(225) bytes of data.
<nothing...>
```

The problem was resolved by upgrading the MadWiFi drivers in to the latest version (madwifi – 0.9.4).

# 11 Table of useful commands

| Command | Arguments | Description |
|---|---|---|
| meshpower | <bank\|all> <on\|off> \| time(s)\|<br><br>**or**<br><br>status | This command turns on or off a row of nodes in the grid. The row will consist of nodes 11-17, 21-27 ... 71-77. If you want to turn on all the nodes in the grid use the all argument and you can then specify a time in seconds between when each bank will be switched on. If the time argument is left out, it will default to a 10 second interval between banks switching on.<br><br>**example1**: meshpower 2 off<br>turns off bank 2.<br><br>**example2**: meshpower all on 30<br>turns on all banks with a 30 second interval between rows switching on<br><br>When the status argument is used it will report which banks are on and which banks are off |
| mk-dhcp-classes | [ -d <default class> ] <range> <class> ... | This command is used to change the operating system that is booted on each node. It can be used to change the booted operating system on selection of nodes or on all the nodes.<br><br>The range is in the form of x1y1-x2y2 where x1y1 is the starting grid position and x2y2 is the end grid position. If the range is between different rows then |
| pingmesh | None. | Pings all nodes via the wired network, reports any nodes which are down. |
| shakemesh | None. | Configure wireless devices to default values for all nodes on the mesh. |
| | | |
| | | |