Mesh Lab Manual (Linux)

Version 0.3
May 2009

Written by David Johnson and Kevin Duff

Table of Contents

1 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

2 Getting Started

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

2.1 Access to the Mesh Server

There are two mesh servers, one running Linux and one running Free-BSD The Free-BSD machine has been retired and only acts as an NTP server but can be re-activated as the mesh server at any stage (See Appendix A). See Section 10 for a description of the architecture of the lab.

You will need to ask the administrators of the mesh lab to create an account for you. Once you have credentials, you can log into the server as follows:

If you are located at Meraka :

Linux meshlab.dhcp server

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

Free-BSD meshy.dhcp server

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

The Linux and free-BSD mesh servers are not visible from the Internet, nor any network outside of Meraka. If you require remote access you will need to ask the Administrators to set-up a reverse tunnel to your server.

If you are remotely located and a tunnel has been configured for you:

```
ssh -p 10002 username@localhost
```

2.2 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!

2.2.1 Checking Who Else is Logged Into The Server

To obtain a list of users logged into the Free-BSD Server, just type:

```
users
```

2.2.2 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.
```

2.2.3 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:

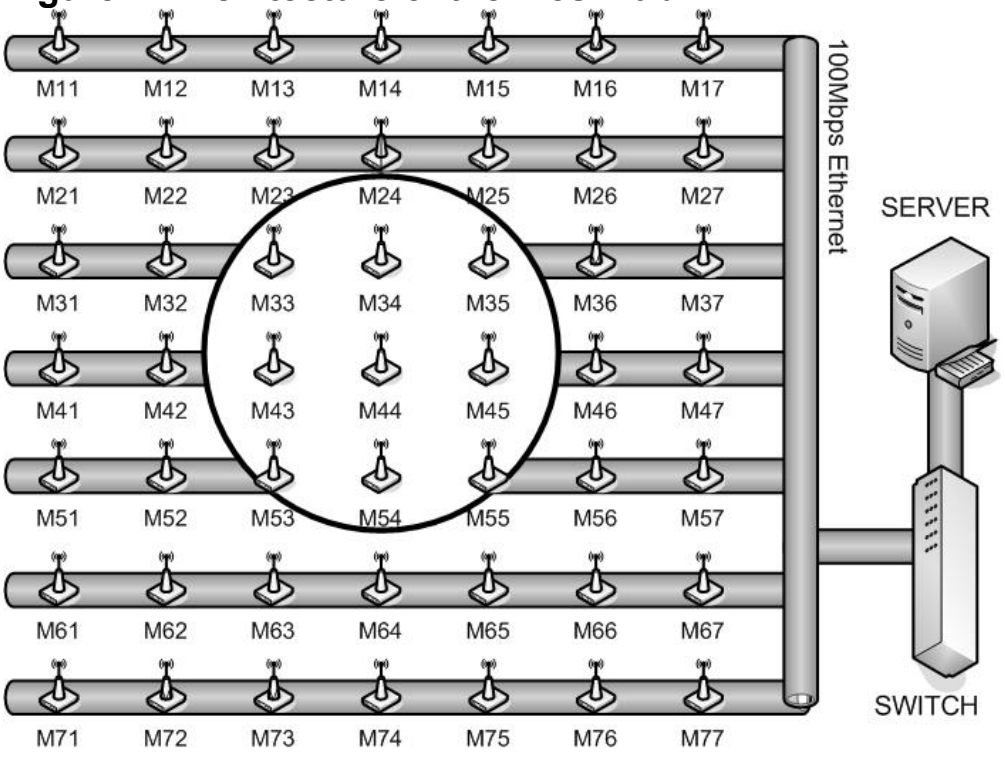
1. Login as root or become root user using su
2. Use text editor vi to edit /etc/motd:

```
# 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!

3 Description of the lab set-up

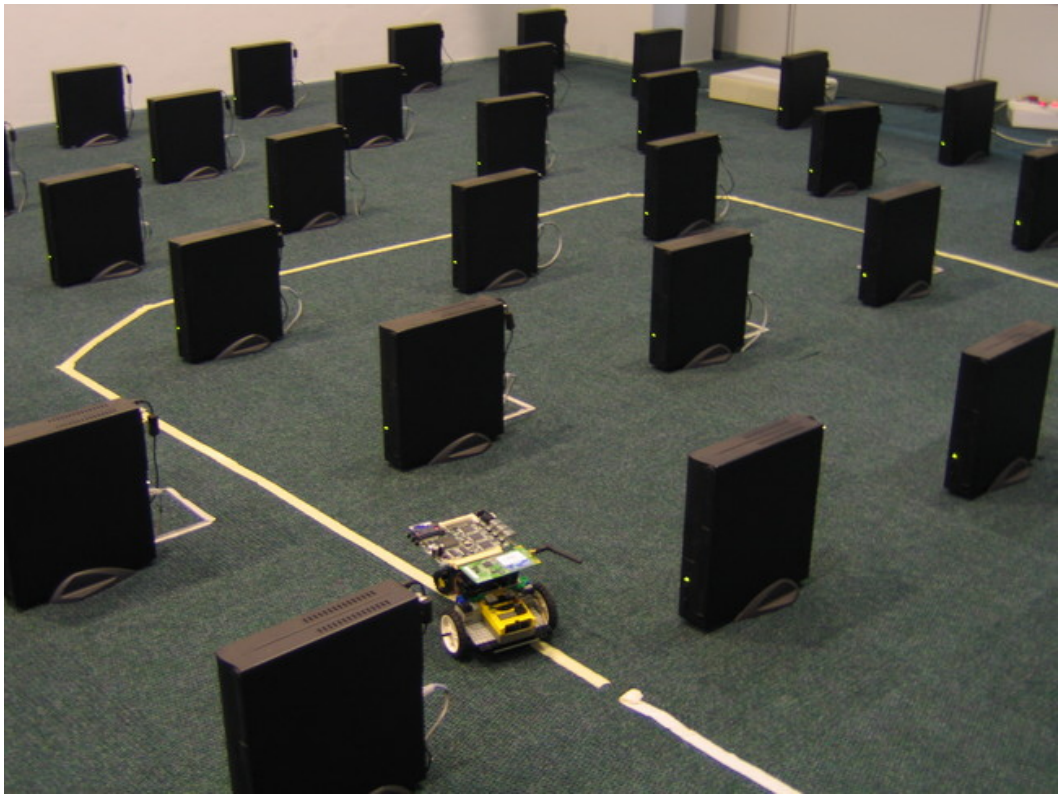
Figure 1: Architecture of the mesh lab



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 2. 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

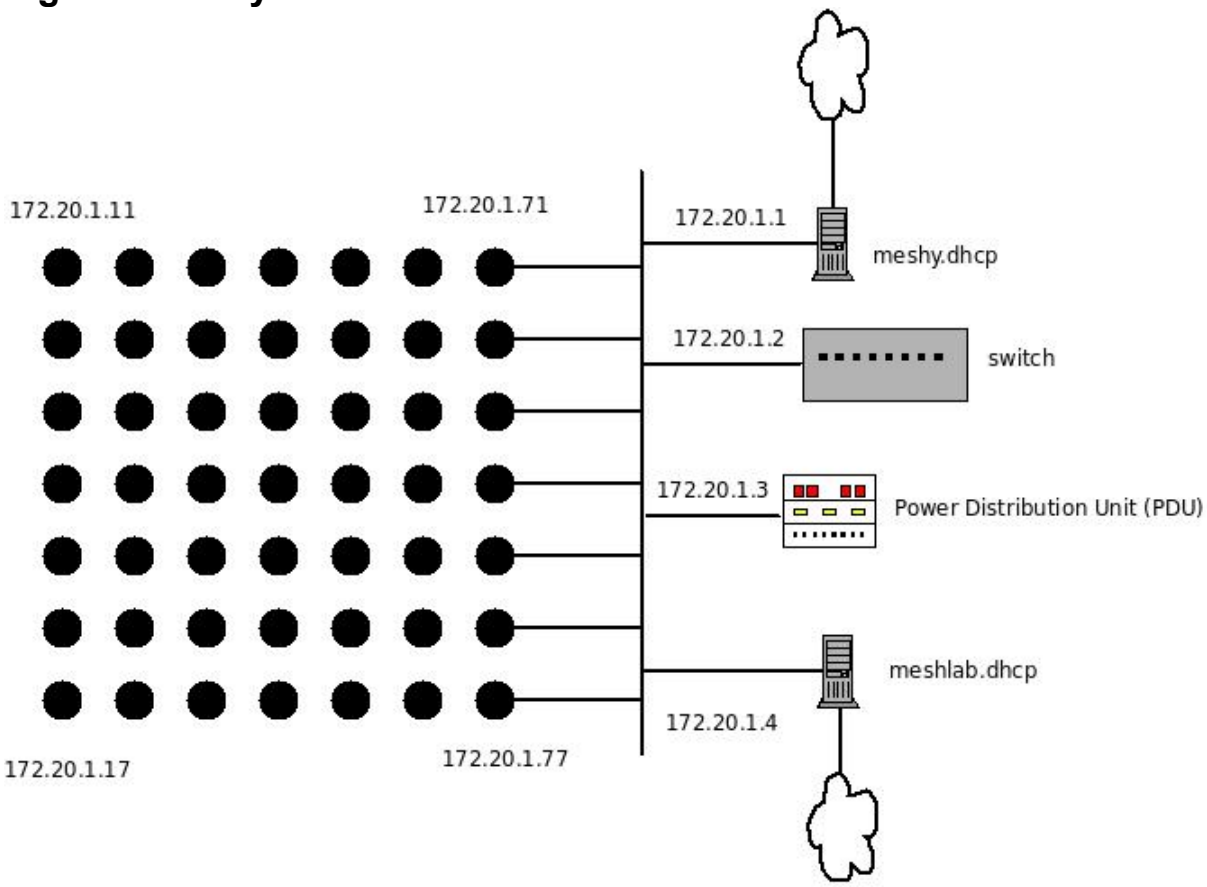
This network was operated at 2.4 GHz due to the availability of antennas and attenuators at that frequency, but in future the laboratory will be migrated to the 5 GHz range, which has many more available channels with a far lower probability of being affected by interference.

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 Free-BSD is the original server which held all the operating systems for all the nodes and has now been retired. The second server, *meshlab,* now contains an *Ubuntu hardy* operating system as well as a very small minimal Linux OS nicknamed *hardcore linux. The* Linux server also hosts some of the monitoring and visualisation software to allow users to monitor the status of each node and their connectivity with other nodes.

The network structure is shown in figure 3: It is organised such that the last octet of the IP address is the same as the xy grid position.

Figure 3: IP layout of the mesh lab



The IP address of the Free-BSD server (meshy.dhcp) mesh side interface is 172.20.1.1 and the Linux server (meshlab.dhcp.) mesh side interface is 172.20.1.4. There is also a switch at IP address 172.20.1.2 which can be managed using Telnet and a Power Distribution Unit (PDU) at 172.20.1.3 which can also be managed using Telnet.

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 (172.20.1.1)

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

This server hosts:

- **Disabled -** 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
- **Disabled** – TFTP server with PXE boot-loaders which nodes in the grid boot off to start the node's operating systems
- **Disabled** - NFS server which mounts 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.
- **Enabled** – NTP time server to synchronize the nodes date and time

To re-enable the services on this server see Appendix A – Note you should never enable the same services on both servers.

This machine has two network interfaces

Interface 1 (connected to Meraka 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) (172.20.1.4)

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

The Ubuntu partition on the server hosts:

- Ubuntu LTSP system which includes
  - 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
  - TFTP server with PXE boot-loaders which nodes in the grid boot off to start the node's operating systems
  - NFS server which mounts a number of operating systems in the /export directory including Ubuntu hardy and a minimal Linux system which fits on 128M of flash
- Visualization software from University of Hungary
- Network monitoring software from University of Hungary
- A scratchbox environment for compiling code for OpenWRT running on the Linksys WRT54G

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 Meraka 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: Coin#123

4.3 The 3COM Switch (172.20.1.2)

This switch can be administered using a Telnet session

```
Telnet 172.20.1.255
user: manager
pass: manager
```

4.4 The Power distribution Unit (PDU) (172.20.1.3)

The PDU is used to power down and power up rows of mesh nodes in order to recover from serious lock-ups. There are 7 rows and each row consists of nodes in the IP address range 172.20.1.***rx*** where ***r*** is the row and ***x*** is a number from 1 to 7.

The PDU can be managed using a Telnet session

```
Telnet 172.20.1.3
user: apc
pass: apc
```

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 set-up 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 (meshlab.dhcp) replies to the node with the Boot server IP address.
4. Node then request the 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 /etc/dhcp3/dhcpd.conf on meshlab.dhcp
    1. In the case of booting a Free-BSD node this has been set to /var/lib/tftpboot/bootx/pxeboot
    2. In the case of booting a Linux node this has been set to /var/lib/tftpboot/bootx/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 ... /etc/dhcp3/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. You will then need to copy your new operating system into a folder in

/export and create a new boot folder in /var/lib/tftpboot to contain the boot-loader , kernel and ram-disk.

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-hardy … Ubuntu hardy LTS release to 2012
3. /export/linux/ubuntu-hardy-click … Ubuntu hardy with kernel modification to work with MIT Click in kernel mode
4. /export/freebsd-6-stable ... Free-BSD stable version
5. /export/freebsd-7-stable ... Free-BSD current unstable 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-hardy linux

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

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 The /var/lib/tftpboot/bootx directories

If you change the version of linux you are booting from a previous boot with the mk-dhcp-classes program, you will notice that each class points to a different directory for the boot-loader. The tftp server specifies that the root of the boot-loader is in */var/lib/tftpboot* and the class entry says which folder the boot-loader is found in. for example ubuntu-hardy is in */boot1/pxelinux.0.*

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.

```
/etc/init.d/dhcp3-server 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 boot-loader gets no IP addresses leases from the server. Do do this simply run the command

```
/etc/init.d/dhcp3-server 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 relay bank is attached via Ethernet to the central 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:

```
 Power: "On  On  On  On  On  On  On  Off "
```

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 particular 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 commands 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-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 "reboot"
```

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 to their default settings.

```
shakemesh
```

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

```
/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 to serial port 1 (/dev/ttyS0) of the Linux 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 /var/lib/tftpboot/bootx/pxelinux.cfg/default. (where x is 1,2,3.... and depends on which operating system you are editing)
3. You should see a line which starts with :
   `APPEND . . .`
4. 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 hardy 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 on the nodes themselves, a chroot environment is needed instead. Here's how we do it:

8.2 Chroot into the host file system

1. Login to the server meshlab.dhcp.meraka.csir.co.za

   from the Meraka network

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

   using a reverse tunnel

   ```
   ssh -p 10002 username@localhost
   ```

2. chroot into the ubuntu hardy file system and mount the /proc file system

   ```
   sudo chroot /export/ubuntu-hardy
   mount -t proc /proc /proc
   ```

8.3 Installing a Package from Source

You are in the chroot environment and ready to install some code from source

1. Download source using a command like ***wget*** or with ***cvs*** or ***git***.
2. Extract archive and copy to a folder in /usr/src or to /home/user
3. Compile and install the package. Refer to package documentation; in general the procedure is:

   ```
   make clean
   make
   make install
   ```

4. Exit out of chroot and login to any node that booted this operating system to test the compiled code.

```
exit
msh 47
```

8.4 Installing a Package from an Ubuntu Repository

You are in the chroot environment and ready to install some code from the Ubuntu package repository.

1. Check that you are happy with the repositories needed for the package you want

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

2. Use apt-get to download the package

```
apt-get install packagename
```

1. Exit out of chroot and login to any node that booted this operating system to test the compiled code.

```
exit
msh 47
```

9 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
```

10 Keeping it clean and simple – linux hardcore

This system was thought out and build mostly by Elektra with some input from David Johnson. In order to fit linux on a 128M flash, a very minimal linux system was needed. To build this system a number of tricks were used.

- Build all code using uclibc which is a libc version which contains a very bare bones library
- Make use of busybox which contains a full set of linux tools such as mkdir, ls, cat … all in one executable.
- Make use of a very simple startup script to create uniqueness between each mesh node. The only unique aspects of each node are the ip address of Ethernet and wifi and the host name. The startup script generates these on the fly, have a look at /export/linux-hardcore/etc/init.d/rcS

The full system is contained in /export/linux-hardcore. It can be booted off this directory using the standard NFS boot process or burned on the built in flash cards and run from flash.

11 Setting up an LTSP Linux server

These are instructions to rebuild a Linux server similar to the one described in this manual to boot a set of thin client nodes connected via a network to a server.

These instructions build on the Linux Terminal Server Project (LTSP) who's purpose is to build a server for thin clients. Many features available in the LTSP project are well documented in the LTSP instruction manual available here (http://wiki.ltsp.org/twiki/bin/view/Ltsp/Documentation)

Start off with a fresh install of Ubuntu hardy server and then do the following.

11.1 Building the basic LTSP system

Install the LTSP server packages

```
sudo apt-get install ltsp-server-standalone bison flex
```

Create a directory to store all the thin client operating systems

```
sudo mkdir -p /export
```

Build the base ubuntu file system on the server

```
ltsp-build-client --base /export/ubuntu-hardy --dist hardy
```

The base system will now be contained in /export/ubuntu-hardy

11.2 Configuring the dhcp server

The dhcp3 server would have been installed as a dependency when you install the package *ltsp-server-standalone*

First you will need to decide which interface the dhcp server will boot on and then edit the file /etc/default/dhcp3-server and change the line with the keyword INTERFACES to point to the correct interface

```
INTERFACES="eth1"
```

When the ltsp package is installed it modifies /etc/init.d/dhcp3-server so that it reads the dhcp.conf file from /etc/ltsp/dhcp3.conf

```
# Allow ltsp to override
if [ -f /etc/ltsp/dhcpd.conf ]; then
      CONFIG_FILE=/etc/ltsp/dhcpd.conf
```

```
fi
```

Edit the default dhcp conf file /etc/ltsp/dhcp.conf

```
authoritative;

option domain-name-servers 172.20.1.4;
option domain-name "meraka.csir.co.za cids.org.za";
option broadcast-address 172.20.1.255;
option subnet-mask 255.255.255.0;
option routers 172.20.1.4;
        get-lease-hostnames true;
option ntp-servers 172.20.1.1;

option space PXE;
option PXE.mtftp-ip code 1 = ip-address;
option PXE.mtftp-cport code 2 = unsigned integer 16;
option PXE.mtftp-sport code 3 = unsigned integer 16;
option PXE.mtftp-tmout code 4 = unsigned integer 8;
option PXE.mtftp-delay code 5 = unsigned integer 8;
option PXE.discovery-control code 6 = unsigned integer 8;
option PXE.discovery-mcast-addr code 7 = ip-address;

ddns-update-style none;

default-lease-time 43200;
max-lease-time 43200;

class "linux-hardy" {
        match hardware;
        use-host-decl-names on;
        option vendor-class-identifier "PXEClient";
        vendor-option-space PXE;
        option PXE.discovery-control 11;
        option domain-name-servers 172.20.1.4;
        filename "/boot1/pxelinux.0";
        option root-path "/export/ubuntu-hardy";
}
```

```
include "/etc/ltsp/dhcpd-subclasses.inc";

subnet 172.20.1.0 netmask 255.255.255.0 {
}


# Our ethernet switch
host 172.20.1.2 {
        #hardware ethernet 00:12:A9:86:4B:A0;  #old switch
        hardware ethernet 00:0A:04:A0:E9:C0; #current switch
        fixed-address 172.20.1.2;
}

# Our power switch
host 172.20.1.3 {
        hardware ethernet 00:C0:B7:7D:1C:DF;
        fixed-address 172.20.1.3;
}

include "/etc/ltsp/dhcpd.inc";
```

There are some optional extra entries in the file:

NTP server – if you have one then this can be specified here:

```
option ntp-servers 172.20.1.1;
```

Some host names that need to have fixed IP addresses were given here:

```
# Our power switch
host 172.20.1.3 {
        hardware ethernet 00:C0:B7:7D:1C:DF;
        fixed-address 172.20.1.3;
}
```

```
# Our ethernet switch
host 172.20.1.2 {
        #hardware ethernet 00:12:A9:86:4B:A0;  #old switch
        hardware ethernet 00:0A:04:A0:E9:C0; #current switch
        fixed-address 172.20.1.2;
}
```

There are two files included in the this dhcpd.conf file : *dhcpd-subclasses.inc* and *dhcpd.inc*. The *dhcpd-subclasses.inc* file defines which operating system will boot on

a specific node with a given MAC address when there is more than one class section in the dhcp.conf file. It looks as follows:

```
subclass        "linux-hardy-click"    01:00:40:63:de:5b:ec;
subclass        "linux-hardy-click"    01:00:40:63:de:96:58;
subclass        "linux-hardy-click"    01:00:40:63:da:7e:e2;
```

The dhcpd.inc file maps host names to specific MAC addresses and looks as follows:

```
host mesh-1-1 {
        hardware ethernet 00:40:63:de:5b:ec;
        fixed-address mesh-1-1;
}

host mesh-1-2 {
        hardware ethernet 00:40:63:de:96:58;
        fixed-address mesh-1-2;
}
```

You also need to ensure that you have defined the IP addresses for each of the host names given in this file in the /etc/hosts file:

```
172.20.1.11             mesh-1-1
172.20.1.12             mesh-1-2
172.20.1.13              mesh-1-3
```

Once this is all setup you can restart the dhcp server with:

```
sudo /etc/init.d/dhcp3-server restart
```

To ensure that the dhcp server will always start up when the computer reboots, type the following:

```
ls -al /etc/rc2.d/ | grep dhcp:
```

you should see the following result

```
S40dhcp3-server -> ../init.d/dhcp3-server
subclass        "linux-hardy-click"    01:00:40:63:de:5b:ec;
subclass        "linux-hardy-click"    01:00:40:63:de:96:58;
subclass        "linux-hardy-click"    01:00:40:63:da:7e:e2;
```

If you don't see this then type the following to make dhcp startup when you reboot

```
sudo ln -s /etc/rc2.d/S40dhcp3-server /etc/init.d/dhcp3-
server
```

11.3 Configuring TFTP server

When a thin client, which uses a PXE boot process, boots, it gets an IP address from the DHCP server and then uses trivial File Transfer Protocol (TFTP) to transfer a small boot loader to the thin client machine. The standard tftp server that is installed with the *ltsp-server-standalone* package is *tftpd-hpa.*

The default configuration for tftp-hpa is stored in /etc/default/tftpd-hpa, the file should look something like this:

```
#Defaults for tftpd-hpa
RUN_DAEMON="yes"
OPTIONS="-l -s /var/lib/tftpboot"
```

In the default configuration the root of the tftp file system points to /var/lib/tftpboot, this directory is created for you when you install the *ltsp-server-standalone* package and the boot loader, *pxelinux.0* is also copied into that directory. This will be discussed in the next section.

To restart the tftp file server execute:

```
sudo  /etc/init.d/tftpd-hpa restart
```

To ensure that the tftp system always restarts when booting

```
ls -al /etc/rc2.d/ | grep tftp:
```

you should see the following result

```
 S20tftpd-hpa -> ../init.d/tftpd-hpa
```

If you don't see this then type the following to make tftp startup when you reboot

```
sudo ln -s /etc/rc2.d/S20tftp-hpa /etc/init.d/tftp-hpa
```

11.4 The PXE boot loader

The first executable file that is transferred to the client using TFTP is the PXE boot loader. The file is located by concatenating the base directory configured by the tftp server, */var/lib/tftpboot* and the subdirectory and file pointed to in the dhcp server configuration file which is /*boot1/pxelinux.0* in this example – recall this line in the *dhcp.conf* file:

```
filename "/boot1/pxelinux.0";
```

So the full path to the executable boot loader is:

```
/var/lib/tftpboot/boot1/pxelinux.0
```

From now on we will call */var/lib/tftpboot/boot1* the *<boot path>.* To configure the kernel that will be loaded by the boot loader, a configuration file needs to be placed in the *<boot path>/pxelinux.cfg* sub directory.

The pxelinux boot loader has a mechanism to uniquely map specific configuration files to specific hosts, for further reading consult the PXELINUX web site: http://syslinux.zytor.com/wiki/index.php/PXELINUX.

For this system, unique hosts are each given their own boot directory so this mechanism will not be utilised. The default configuration file will be used which is simply: *<boot path>/pxelinux.cfg/default*

The default file looks as follows:

```
DEFAULT vmlinuz ro initrd=initrd.img  console=ttyS0,115200
```

This tells the pxelinux to load the kernel *<boot path>/vmlinuz* as read only using initial ramdisk *<boot path>/initrd.img* and send standard out to the serial port *ttyS0*

You need to ensure these kernel files exist or else the linux kernel will not boot.

11.5 How to make different operating systems boot on each host

The dhcp.conf file allows you to specify different classes of hosts for example you could add these classes to */etc/ltsp/dhcp.conf*

```
class "linux-hardy" {
        match hardware;
        use-host-decl-names on;
        option vendor-class-identifier "PXEClient";
        vendor-option-space PXE;
        option PXE.discovery-control 11;
        option domain-name-servers 172.20.1.4;
        filename "/boot1/pxelinux.0";
        option root-path "/export/ubuntu-hardy";
}

class "linux-hardcore" {
        match hardware;
        use-host-decl-names on;
        option vendor-class-identifier "PXEClient";
        vendor-option-space PXE;
        option PXE.discovery-control 11;
```

```
        option domain-name-servers 172.20.1.4;
        filename "/boot3/pxelinux.0";
        option root-path "/export/linux-hardcore";
}
```

This allows you to specify different boot-loader executables for each host class, in this example the *linux-hardy* class used ***/boot1/pxelinux.0*** and the *linux-hardcore* class used the ***/boot3/pxelinux.0*** boot-loader The root path used by the kernel can also be specified here using *option root-path.* In this example the *linux-hardy* class used ***/export/ubuntu-hardy*** as its root path that nfs will mount and the *linux-hardcore* class used ***/export/linux-hardcore*** as the root path that nfs will mount.

To map specific host MAC addresses to these classes the file /etc/ltsp/dhcp-subclasses.inc is used:

```
subclass        "linux-hardy"    01:00:40:63:de:5b:ec;
subclass        "linux-hardy"    01:00:40:63:de:96:58;
subclass        "linux-hardcore"    01:00:40:63:da:7e:e2;
```

To simplify the process of editing this file, a tool called *mk-dhcp-classes* was created see section 14 for details on its use.

11.6 Configuring the NFS server

The *ltsp-server-standalone* package installs the *nfs-kernel-server* package as a dependency. To configure nfs, you need to let nfs know which directories you want to export. In the case of a thin client setup, you need to export all the directories that will be mounted as root using nfs or just the root of all these directories. In this setup all the host file-systems are contained in the directory */export*, so this is the only directory that needs to be exported.

Edit the exports file

```
sudo vi /etc/exports
```

Add the following line to this file

```
/export *(rw,async,no_subtree_check,no_root_squash)
```

By default nfs only creates 8 nfs server threads, if you use more clients than this you need to increase this to at least double the number of clients you are using just as a safety margin. For the massive mesh there are 49 clients, so at least 98 nfs server threads are required. In case of future expansion, the number of threads was set to 128.

Edit the nfs configuration file

```
sudo vi /etc/default/nfs-kernel-server
```

Change the following line:

```
# Number of servers to start up
RPCNFSDCOUNT=128
```

Now restart the nfs server

```
sudo /etc/init.d/nfs-kernel-server
```

11.7 Preparing the linux kernel

Once the boot-loader has finished loading it will look for the linux kernel specified in the pxelinux.cfg/default file. After the ltsp-build-client program has finished executing in section 24 it will copy all the linux kernel boot files to the */var/lib/tftpboot/ltsp/i386* directory.

Copy all the files from the default ltsp root directory to one of the custom boot directories:

```
sudo cp /var/lib/tftpboot/ltsp/i386/ */var/lib/tftpboot/boot1
```

By default the ltsp system makes use of a Network block device (NBD) rather than Network File system (NBD). NBD gives you the advantage of quick booting due to its use of a compressed file system but it doesn't provide a writeable file system which maps to a normal linux directory structure on the server.

In order to change from NBD to NFS you need to do the following

On the server, use the chroot command to maintain the LTSP chroot:

```
sudo chroot /export/ubuntu-hardy
```

Now edit /etc/default/ltsp-client-setup and change the value of the root_write_method variable to use bind mounts instead of unionfs, it should look like this afterwards:

```
root_write_method="bind_mounts"
```

Next, create the file /etc/initramfs-tools/conf.d/ltsp and add the following line (set the value of the BOOT variable to nfs):

```
BOOT=nfs
```

Regenerate the initramfs:

```
update-initramfs -u
```

Exit out of the chroot environment

```
exit
```

Update the current ltsp client kernel by using the current kernel from the ltsp chroot and copying it to the tftpboot directory

```
sudo ltsp-update-kernels -b /export/ubuntu-hardy -d boot1
```

11.8 Some customizations to the root filesystem

In order to customize the host operating system – you need to chroot into the root of the host file system:

```
sudo chroot /export/ubuntu-hardy
sudo mount -t proc /proc /proc
```

Now you will need to update your sources.list to point to your nearest ubuntu distribution. In this case the nearest repository was at http://goblin.meraka.csir.co.za

```
vi /etc/apt/sources.list
```

Contents of /etc/apt/sources.list file

```
## Main

deb http://goblin.meraka.csir.co.za/ubuntu/ hardy main restricted
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy main restricted

## Updates

deb http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates main restricted
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates main restricted

## Universe + Universe updates

deb http://goblin.meraka.csir.co.za/ubuntu/ hardy universe
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy universe
deb http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates universe
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates universe

## Multiverse + Multivese updates

deb http://goblin.meraka.csir.co.za/ubuntu/ hardy multiverse
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy multiverse
deb http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates multiverse
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy-updates multiverse

## Backports

deb http://goblin.meraka.csir.co.za/ubuntu/ hardy-backports main restricted universe
multiverse
deb-src http://goblin.meraka.csir.co.za/ubuntu/ hardy-backports main restricted universe
multiverse

## Partner offered by canonical

deb http://archive.canonical.com/ubuntu hardy partner
```

```
deb-src http://archive.canonical.com/ubuntu hardy partner

## Security

deb http://goblin.meraka.csir.co.za/ubuntu hardy-security main restricted
deb-src http://goblin.meraka.csir.co.za/ubuntu hardy-security main restricted
deb http://goblin.meraka.csir.co.za/ubuntu hardy-security universe
deb-src http://goblin.meraka.csir.co.za/ubuntu hardy-security universe
deb http://goblin.meraka.csir.co.za/ubuntu hardy-security multiverse
deb-src http://goblin.meraka.csir.co.za/ubuntu hardy-security multiverse

## PLF REPOSITORY (Unsupported.  May contain illegal packages)
deb http://packages.medibuntu.org/ hardy free non-free
```

Now update the package repository

```
sudo apt-get update
```

Install openssh server so that you can ssh into the nodes

```
sudo install openssh-server
```

Ensure that the resolv.conf file points to a valid DNS that can be reached by the nodes – if this contains an unreachable IP address, ssh can take up to a minute before logging in due to failed DNS lookup

```
sudo vi /etc/resolv.conf
```

Entry for DNS server at 172.20.1.4

```
search meraka.csir.co.za
nameserver 172.20.1.4
nameserver 146.64.28.1
```

Enable root password for the hosts

```
sudo passwd -u root
```

Allow IP forwarding in nodes

```
vi /etc/sysctl.conf
```

Uncomment these lines:

```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
net.ipv6.ip_forward=1
```

Get some extra packages, vim for a better vi and pmisc to give you the killall command.

```
apt-get install vim psmisc
```

11.8.1 The lts.conf file

This allows a lot of customization to be done to the LTSP system (see from Chapter 8 onwards of the LTSP manual or you can also find some information at:: http://www.ltsp.org/twiki/bin/view/Ltsp/Syslog). This file is located in /etc/lts.conf

```
vi /etc/lts.conf
```

An undocumented feature is the SYSLOG option. If you want a server to log remotely then you specify

```
SYSLOG=remote
```

If you want to log locally then you specify

```
SYSLOG=local
```

For local logging you will also need to edit the */etc/syslog.conf* file to specify all the information you would like to log. Simply copy the */etc/syslog.conf* file from your server machine to log most important system events.

By default the hosts will start up with a normal ubuntu GUI, in order to start up in a standard terminal specify how many tty terminals you would like

```
SCREEN_01=shell
SCREEN_02=shell
SCREEN_03=shell
SCREEN_04=shell
SCREEN_05=shell
SCREEN_06=shell
#SCREEN_07=ldm
SCREEN_07=shell
```

This would start up 7 shells which can be accessed with ***ctrl-alt-F1 or F2*** … etc. if *SCREEN_XX=ldm* is specified then it will start an X sessions on that pty.

If you want to start up some custom settings on each host machine then you can specify these using per thin client settings. Add an entry which has the host machine mac address followed by custom settings

```
[00:40:63:DE:95:F5]
SYSLOG=remote
```

will start remote logging on machine with MAC address 00:40:63:DE:95:F5

If you want to start up some custom scripts you can use the RCFILE_01 to RCFILE_10 options for example

```
[00:40:63:DE:95:F5]
RCFILE_01=/etc/init.d/myrcfile_machine1
```

will start the /etc/init.d/myrcfile_machine1 script on machine with MAC address 00:40:63:DE:95:F5.

Now log out of the chroot environment with

```
exit
```

Copy ssh keys from server to host filesystem to avoid having to enter a password when logging into host nodes.

Create the .ssh directory in the root account of the chroot filesystem

```
mkdir -p /export/ubuntu-hardy/root/.ssh
```

Create your ssh host keys

```
ssh-keygen
```

Add the public key to the list of authorized keys for the host system

```
cat .ssh/id_rsa.pub >> /export/ubuntu-
hardy/root/.ssh/authorized_keys
```

11.9 Customizing the kernel

If you need to rebuild the kernel for the ltsp clients then follow the following set of instructions. A good starting point for reading up on building custom kernels can be found here: http://www.howtoforge.com/kernel_compilation_ubuntu

First chroot into the host filesystem

```
sudo chroot /export/ubuntu-hardy
sudo mount -t proc /proc /proc
```

Install all the tools required to re-compile the linux kernel

```
apt-get install build-essential linux-source kernel-package
libncurses5-dev fakeroot wget bzip2
```

Unzip the source code

```
cd /usr/src
tar jxvf linux-source-2.6.24.tar.bz2
cd linux-source-2.6.24
```

Copy the configuration of the existing linux kernel to the build environemnt. This will capture all the existing kernel settings that have been configured to allow clients to boot up as thin clients.

```
cp /boot/config-`uname -r` ./.config
```

If you need to make some patches then put the patch code in /usr/src. For example this is a patch for the click system

```
cat ../linux-2.6.24.7-patch | patch --verbose -p1
```

Start the kernel configuration menu

```
make menuconfig
```

Remove the kernel debugging option – this prevents very large init ramdisks

```
     Kernel Hacking ---->
         [ ] Kernel debugging
```

Make the kernel and modules and install everything

```
make; make bzImage; make modules; make modules_install;
mkinitramfs -o initrd.img-2.6.24.3 2.6.24.3
```

Copy the kernel and init ram disk to the boot directory

```
cp  arch/x86/boot/bzImage /boot/vmlinuz-2.6.24.3-custom
cp ./initrd.img-2.6.24.3 /boot/initrd.img-2.6.24.3-custom
```

Link the new kernel and ram disk to the new images

```
cd /boot
rm vmlinuz
rm initrd.img
ln -s vmlinuz-2.6.24.3-custom vmlinuz
ln -s initrd.img-2.6.24.3-custom initrd.img
```

To ensure that you can still compile code which depends on the correct kernel source in the chroot environemnt you need to make a symbolic link from the chroot linux kernel version to the real linux kernel version that will boot on the hosts

For example if: *uname -r* shows *2.6.24-16-generic* in the chroot environment but the real kernel source is /lib/modules/2.6.24.3 then you need to do the following.

```
cd /lib/modules
mv 2.6.24-16-generic old_ 2.6.24-16-generic
ln -s 2.6.24-16-generic 2.6.24.3
cd 2.6.24.3
ln -s /usr/src/linux-source-2.6.24 source
```

```
ln -s /usr/src/linux-source-2.6.24 build
```

Exit the chroot environment

```
exit
```

Update the current ltsp client kernel by using the current kernel from the ltsp chroot and copying it to the tftpboot directory

```
sudo ltsp-update-kernels -b /export/ubuntu-hardy -d boot1
```

12 Table of useful commands

Command	Arguments	Description
meshpower	<bank|all> <on|off> | time(s)| **or** 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. **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 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. 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.

13 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.

13.1 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.

13.2 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.

13.2.1 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.

13.3 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).

14 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
/home/djohnson/bin	Script files for controlling the mesh
/export/ubuntu-hardy	Ubuntu hardy LTS release to 2012
/export/ubuntu-hardy-click	Ubuntu hardy adapted for the click system
/export/linux-hardcore	Linux hardcore filesystem
/export/freebsd-6-stable	FreeBSD 6 stable
/var/lib/tftpboot	PXE and Netbooting Files … see each directory boot1, boot2 … for individual operating system boot files
/etc/dhcp3	Contains all the important dhcp settings for booting LTSP nodes

Appendix A: How to re-enable the FreeBSD mesh server

The FreeBSD server was the original server used for the mesh lab and has served the lab well since 2005 when the mesh lab was first used. However it was decided that due to many users only having a good command of Linux and due to the fact chroot from FreeBSD into a linux host operating system was not working, a Linux server would be used instead. Most of the services on the FreeBSD mesh server (meshy.dhcp) were thus disabled in order for the Linux server (meshlab.dhcp) to take over this role.

If in the future the FreeBSD server needs to be recommissioned all these services can be re-enabled as follows

Login to the FreeBSD mesh server (meshy.dhcp)

```
ssh meshy.dhcp
```

Become root on the machine

```
su
```

Edit the rc.conf file

```
vi /etc/rc.conf
```

Change the following lines

```
nfs_server_enable="YES"
rpcbind_enable=”YES”
dhcp_enable=”YES”
rpc_lockd_enable=”YES”
rpc_statd_enable=”YES”
inetd_enable=”YES”
```

reboot the machine

```
reboot
```

Now disable the mesh side interface of the Linux server (meshlab.dhcp)

```
ssh meshlab.dhcp
sudo ifconfig eth1 down
```

If this is a permanent change then make sure the interface doesn't come back up on reboot

```
vi /etc/network/interfaces
```

Remove the line

```
auto eth1
```

Now you're back to the good old FreeBSD days back in 2005. Have a look at the manual for the FreeBSD system available at http://wirelessafrica.meraka.org.za/wiki/images/e/e7/Meshlab_manual_freebsd.pdf