

# LTSP – Linux Terminal Server Project – v3.0

**James McQuillan**

jam@LTSP.org

Copyright © 2001 by James A. McQuillan

## **Revision History**

Revision 1.0

2001-12-02

Revised by: jam

GNU/Linux makes a great platform for deploying diskless thin clients. The primary purpose of this document is to show you how to deploy thin clients using LTSP. But, this document also covers many issues with diskless workstations in general.

---



# Table of Contents

<b>Introduction.....</b>	<b>1</b>
1. Disclaimer.....	1
2. Copyright and License.....	1
<b>Chapter 1. Theory of operation.....</b>	<b>3</b>
<b>Chapter 2. Installing LTSP on the server.....</b>	<b>7</b>
2.1. Installing the RPM packages.....	7
2.1.1. LTSP Core package.....	7
2.1.2. LTSP Kernel package.....	7
2.1.3. LTSP X Core package.....	7
2.1.4. LTSP X Fonts package.....	7
2.1.5. Initialization of the server.....	8
2.1.6. Workstation specific configuration.....	8
<b>Chapter 3. Setting up the workstation.....</b>	<b>11</b>
3.1. Creating the boot floppy.....	11
<b>Chapter 4. Running the workstation.....</b>	<b>13</b>
<b>Chapter 5. Printing.....</b>	<b>15</b>
5.1. Client side setup.....	15
5.2. Server side setup.....	15
<b>Chapter 6. Trouble-shooting.....</b>	<b>19</b>
6.1. Trouble-shooting Etherboot floppy image.....	19
6.2. Trouble-shooting DHCP.....	19
6.2.1. Check the connections.....	20
6.2.2. Is dhcpd running?.....	20
6.2.3. Double-check the dhcpd configuration.....	20
6.2.4. Is ipchains or iptables blocking the request?.....	21
6.2.5. Is the workstation sending the request?.....	21
6.3. Trouble-shooting TFTP.....	21
6.3.1. tftpd is not running.....	22
6.3.2. Kernel not where tftpd expects to find it.....	22
6.4. Trouble-shooting NFS root filesystem.....	22
6.4.1. No init found.....	22
6.4.2. Server returned error -13.....	22
6.4.3. NFS Daemon problems (portmap, nfsd & mountd).....	22
6.5. Trouble-shooting the Xserver.....	24
6.6. Trouble-shooting the Display manager.....	25
6.6.1. Grey screen with large X cursor.....	25
<b>Chapter 7. Kernels.....</b>	<b>29</b>
7.1. Standard LTSP supplied kernels.....	29
7.2. Build your own kernel.....	29
7.2.1. Obtaining the source for the kernel.....	29
7.2.2. Kernel Patches.....	30
7.2.3. Configuring kernel options.....	31
7.2.4. Building the kernel.....	32
7.2.5. Tagging the kernel for Etherboot.....	33

# Table of Contents

<b>Chapter 8. Its.conf entries.....</b>	<b>35</b>
8.1. Sample lts.conf file.....	35
8.2. Available lts.conf parameters.....	35
8.2.1. General parameters.....	35
8.2.2. X–Windows parameters.....	37
8.2.3. Touch screen parameters.....	38
8.2.4. Local apps parameters.....	39
8.2.5. Keyboard parameters.....	39
8.2.6. Printer configuration parameters.....	39
<b>Chapter 9. Local Applications.....</b>	<b>43</b>
9.1. Benefits of running apps locally.....	43
9.2. Issues with setting up support for local apps.....	43
9.3. Server Configuration for Local Apps.....	43
9.3.1. lts.conf entries.....	43
9.3.2. Network Information Service – NIS.....	44
9.4. Application Configuration.....	44
9.5. Launching local applications.....	45
<b>Chapter 10. Other sources of information.....</b>	<b>47</b>
10.1. Online references.....	47
10.2. Print publications.....	47

# Introduction

The LTSP provides a simple way to utilize low cost workstations as either graphical or character based terminals on a GNU/Linux server.

In a traditional office setting, there are relatively high powered Intel based PC's spread around at every desk. Each with several gigabytes of hard disk space. Users store their own data on the local hard drives and backups are rarely (if ever) performed.

Does it really make sense to have a full computer at each desk?

We say no.

Fortunately, there is another way. Utilizing the LTSP, you can take very low-end PCs, remove the hard drive, floppy and CDRom, and add a bootable network card. Many network cards have bootrom sockets, just waiting for a bootrom to be inserted.

During the boot phase, the diskless workstation obtains its IP info and a kernel from the server, then mounts the root filesystem from the server via NFS.

The workstation can be configured in one of 3 modes:

- **Graphical X Windows**

Using X Windows, the workstation can be used to access any applications on the server, or on other servers within the network.

- **Character based Telnet sessions**

The workstation can invoke multiple telnet sessions to the server. Each Telnet session will be on a separate virtual screen. Pressing Alt-F1 through Alt-F9 will switch between the telnet sessions.

- **Shell prompt**

The workstation can be configured to drop you right into a bash shell at the console. This is very useful when debugging problems with X Windows or NFS.

The really neat thing is that you can have lots of workstations served by a single GNU/Linux server. How many workstations? Well, that depends on the size of the server, and the applications that will be used.

It's not unusual to have 40 workstations, all running Netscape and StarOffice from a Dual PIII-650 with 1GB of ram. We know this works. In fact, the load-average is rarely above 1.0!

---

## 1. Disclaimer

Neither the author nor the distributors, or any other contributor of this document are in any way responsible for physical, financial, moral or any other type of damage incurred by following the suggestions in this text.

---

## 2. Copyright and License

This document is copyright 2001 by James McQuillan, and is released under the terms of the GNU Free Documentation License, which is hereby incorporated by reference.

---



# Chapter 1. Theory of operation

Booting a diskless workstation involves several steps. Understanding what is happening along the way will make it much easier to solve problems, should they arise.

This example is based on the following configuration:

- Standard x86 based workstation
- Linksys LNE100TX network card with Etherboot bootrom
- Intel i810 based graphics chipset
- Server running Redhat 7.2
- DHCP
- Network address of 192.168.0.0/24

Assuming the server is configured with the LTSP package, this is what happens:

1. When you turn on the workstation, it will go through its "Power On Self Test" (POST).
2. During the self test, the bios will search for expansion roms. The network card contains an Etherboot bootrom, which is an expansion rom. The bios will detect the rom on the network card.
3. Once the POST is complete, execution will jump into the Etherboot code.
4. The Etherboot code will scan for a network card. Once it detects the card, the card will be initialized.
5. The Etherboot code will then broadcast a DHCP request to the local network. The request will include the MAC address of the network card.
6. The dhcpd daemon on the server will see the broadcast request from the workstation, and look in its configuration file for an entry that matches the MAC address of that workstation.
7. The dhcpd daemon will construct a reply packet, containing several pieces of information. This packet will be sent back to the workstation. The reply information includes:
  - ◆ IP address for the workstation
  - ◆ NETMASK setting for the local network
  - ◆ Pathname of the kernel to download
  - ◆ Pathname of the root filesystem to mount
  - ◆ Optional parameters to be passed to the kernel, via the kernel command line
8. The Etherboot code will receive the reply from the server, and it will configure the TCP/IP interface in the network card with the parameters that were supplied.
9. Using TFTP (Trivial File Transfer Protocol), the Etherboot code will contact the server and begin downloading the kernel.
10. Once the kernel has been completely downloaded to the workstation, the Etherboot code will place the kernel into the correct location in memory.
11. Control is then passed to the Kernel. The kernel will initialize the entire system and all of the peripherals that it recognizes.
12. This is where the fun really begins. Tacked onto the end of the kernel is a filesystem image. This is loaded into memory as a ramdisk, and temporarily mounted as the root filesystem. A kernel command line argument of **root=/dev/ram0** tells the kernel to mount the image as the root directory.

13. Normally, when the kernel is finished booting, it will launch the **init** program. But, in this case, we've instructed the kernel to load a shell script instead. We do this by passing **init=/linuxrc** on the kernel command line.
14. The **/linuxrc** script begins by scanning the PCI bus, looking for a network card. For each PCI device it finds, it does a lookup in the `/etc/niclist` file, to see if it finds a match. Once a match is found, the name of the NIC driver module is returned, and that kernel module is loaded. For ISA cards, the driver module **MUST** be specified on the kernel command line, along with any **IRQ** or address parameters that are required.
15. Once the network card has been identified, the `/linuxrc` script will load the kernel module that supports that card.
16. **dhclient** will then be run, to make another query from the DHCP server. We need to do this separate user-space query, because if we depend on the query that comes from Etherboot, it will get swallowed up by the kernel. And, the kernel will ignore any NFS server that might have been specified in the root-path. This is important if you want the NFS server to be different from the TFTP server.
17. When **dhclient** gets a reply from the server, it will run the `/etc/dhclient-script` file, which will take the information retrieved, and configure the `eth0` interface.
18. Up to this point, the root filesystem has been a ram disk. Now, the `/linuxrc` script will mount a new root filesystem via NFS. The directory that is exported from the server is typically `/opt/ltsp/i386`. It can't just mount the new filesystem as `/`. It must first mount it as `/mnt`. Then, it will do a **pivot\_root**. `pivot_root` will swap the current root filesystem for a new filesystem. When it completes, the NFS filesystem will be mounted on `/`, and the old root filesystem will be mounted on `/oldroot`.
19. Once the mounting and pivoting of the new root filesystem is complete, we are done with the `/linuxrc` shell script and we need to invoke the real **init** program.
20. `init` will read the `/etc/inittab` file and begin setting up the workstation environment.
21. `init` maintains an idea of *runlevel*. Each runlevel sets up a different set of services for the workstation. The LTSP workstation starts in runlevel '2'. That is set by the `initdefault` line in the `inittab` file.
22. One of the first items in the `inittab` file is the **rc.local** command that will be run while the workstation is in the '**sysinit**' state.
23. The `rc.local` script will create a 1mb ramdisk to contain all of the things that need to be written to or modified in any way.
24. The ramdisk will be mounted as the `/tmp` directory. Any files that need to be written will actually exist in the `/tmp` directory, and there are symbolic links pointing to these files.
25. The `/proc` filesystem is mounted.
26. If the workstation is configured to swap over NFS, the `/var/opt/ltsp/swapfiles` directory will be mounted as `/tmp/swapfiles`. Then, if there isn't a swapfile for this workstation yet, it will be created automatically. The size of the swapfile is configured in the `lts.conf` file.  
  
The swapfile will then be enabled, using the **swapon** command.
27. The **loopback** network interface is configured. This is the networking interface that has `127.0.0.1` as its IP address.
28. If Local apps is enabled, then the **/home** directory will be mounted, so that the apps can access the users home directories.



29. Several directories are created in the `/tmp` filesystem for holding some of the transient files that are needed while the system is running. Directories such as:

- a. `/tmp/compiled`
- b. `/tmp/var`
- c. `/tmp/var/run`
- d. `/tmp/var/log`
- e. `/tmp/var/lock`
- f. `/tmp/var/lock/subsys`

will all be created.

30. The X Windows system will now be configured. In the **lts.conf** file, there is a parameter called **XSERVER**. If this parameter is missing, or set to "**auto**", then an automatic detection will be attempted. If the card is PCI, then we will get the PCI Vendor and Device id, and do a lookup in the **/etc/vidlist** file.

If the card is supported by XFree86 4.X, the `pci_scan` routine will return the name of the driver module. If it is only supported by XFree86 3.3.6, `pci_scan` will return the name of the X server to use. The `rc.local` script can tell the difference because the older 3.3.6 server names start with 'XF86\_'.

31. If XFree86 4.x is used, then the **/etc/rc.setupx** script will be called to build an XF86Config file for X4. If XFree86 3.3.6 is used, then **/etc/rc.setupx3** will be called to build the XF86Config file.

The XF86Config file will be built, based on entries in the **/etc/lts.conf** file.

32. When the `rc.setupx` script is finished, it will return back to `rc.local`. Then the **/tmp/start\_ws** script will be created. This script is responsible for starting the X server.

33. The `/tmp/syslog.conf` file will be created. This file will contain information telling the **syslogd** daemon which host on the network to send the logging information to. The syslog host is specified in the `lts.conf` file. There is a symbolic link called `/etc/syslog.conf` that points to the `/tmp/syslog.conf` file.

34. The **syslogd** daemon is started, using the config file created in the previous step.

35. Control is then passed back to **init**. Init will then look at the **initdefault** entry to determine which **runlevel** to enter. As of `lts_core-2.08`, the value of `initdefault` is **2**.

36. A runlevel of **2** will cause init to run the **set\_runlevel** script which will read the `lts.conf` file and determine what runlevel the workstation will actually run in.

The standard runlevels for LTSP are **3**, **4** and **5**.

- ◆ **3** – This will start a shell. This is very useful for debugging the workstation.
- ◆ **4** – This will run one or more Telnet sessions in character mode. This is great if you are just replacing old serial terminals.
- ◆ **5** – GUI mode. This will bring up X windows, and send an XDMCP query to the server, which will bring up a login dialog box to let you log into the server. You will need a display manager listening on the server, such as **XDM**, **GDM** or **KDM**.



## Chapter 2. Installing LTSP on the server

The LTSP packages are available in both *RPM* and *TGZ* formats. Choose which format you prefer to install, and follow the appropriate section.

---

### 2.1. Installing the RPM packages

If you want to run X windows on the workstation, there are 4 packages that you will need to download. Keep in mind, for the purposes of this document, we have a workstation with a Tulip based network card and an Intel i810 based video chipset.

1. LTSP Core package
2. Kernel package
3. X Core package
4. X Fonts package

The X Fonts package isn't really required, but for a first time installation, it is recommended. Once you get the hang of setting up a server and diskless workstation, you can setup the X Font Server (XFS).

After the packages are installed, the LTSP system needs to be initialized. This processes includes making the changes to the system configuration files to enable the host system to offer the services needed by one or more workstations.

---

#### 2.1.1. LTSP Core package

Download the latest release of the ltsp *core* package, and install it using the RPM command.

```
rpm -ivh lts_core-3.0.0.i386.rpm
```

The above command will install the package into the `/opt/ltsp/i386` directory.

---

#### 2.1.2. LTSP Kernel package

Download the latest release of the ltsp kernel package. Install it with the RPM command.

```
rpm -ivh lts_kernel-3.0.0.i386.rpm
```

The above command will install the package.

---

#### 2.1.3. LTSP X Core package

Download the latest release of the ltsp X Core package.

```
rpm -ivh lts_x_core-3.0.0.i386.rpm
```

The above command will install the package.

---

#### 2.1.4. LTSP X Fonts package

Download the latest release of the ltsp X Fonts package.

```
rpm -ivh lts_x_fonts-3.0.0.i386.rpm
```

The above command will install the package.

---

### 2.1.5. Initialization of the server

Once the installation of the above packages is complete, you then need to change into the `/opt/ltsp/templates` directory. There are several files there that will configure the system files on your server. Each one of these files is responsible for one system file. take a look at those files and make sure that you agree with what they are going to do. They can potentially make your system vulnerable to a security attack. You may wish to make the changes to the system files manually. if you want to do it automatically, then run the `ltsp_initialize` command:

```
cd /opt/ltsp/templates
./ltsp_initialize
```

The above command will ask you many questions about which services you want to configure. It will initialize the following services:

- XDM – X Display Manager
- GDM – Gnome Display Manager
- Display manager startup script
- bootp
- NFS /etc/exports file
- tcpwrappers
- Port mapper
- syslogd
- TFTP startup script

---

### 2.1.6. Workstation specific configuration

Now, it's time to tell the LTSP server about your specific workstation. There are three files that contain information about the workstation.

1. `/etc/dhcpd.conf`
2. `/etc/hosts`
3. `/opt/ltsp/i386/etc/lts.conf`

---

#### 2.1.6.1. `/etc/dhcpd.conf`

The workstation needs an IP address and other information. It will get the following from the DHCP server:

- IP address
- hostname
- Server IP address
- Default gateway
- Pathname of kernel to load
- Server and directory path to be mounted as the root filesystem

For our example LTSP environment, we have chosen DHCP for handling IP address assignment to the workstations.

During the `ltsp_initialize` script, a sample `dhcpd.conf` file is installed. It is called `/etc/dhcpd.conf`. example you can copy that file to `/etc/dhcpd.conf` to use it as a basis for your dhcp configuration. You will need to modify the parts of this file that pertain to your specific workstation and server environment.

```
default-lease-time          21600;
```

```

max-lease-time                21600;

option subnet-mask             255.255.255.0;
option broadcast-address       192.168.0.255;
option routers                 192.168.0.254;
option domain-name-servers     192.168.0.254;
option domain-name             "ltsp.org";
option root-path               "192.168.0.254:/opt/ltsp/i386";

shared-network WORKSTATIONS {
    subnet 192.168.0.0 netmask 255.255.255.0 {
    }
}

group {
    use-host-decl-names        on;
    option log-servers         192.168.0.254;

    host ws001 {
        hardware ethernet      00:E0:18:E0:04:82;
        fixed-address           192.168.0.1;
        filename                 "/lts/vmlinuz.ltsp";
    }
}

```

**Figure 2–1. /etc/dhcpd.conf**

As of LTSP, version 2.09pre2, you no longer have to specify a particular kernel to load. The standard kernel package supports all of the network cards that Linux supports. There are two kernel files included in the LTSP kernel package. One kernel has the Linux Progress Patch (LPP) applied, and the other kernel does not. The names for the kernels are:

```

vmlinuz-2.4.9-ltsp-5
vmlinuz-2.4.9-ltsp-lpp-5

```

You may have noticed that the kernel is sitting in the `/tftpbboot/lts` directory, but in the "filename" entry in the `/etc/dhcpd.conf` file is missing the leading `/tftpbboot` component of the pathname. That is because on Redhat versions 7.1 and above, TFTP is run with the `-s` option. That causes the tftpd daemon to run in *secure* mode. That is, it does a **chroot** to the `/tftpbboot` directory when it starts. Therefore, all files that are available to the tftpd daemon are relative to the `/tftpbboot` directory.

Other Linux distributions may not have the `-s` option set for tftpd, so you will need to add the `/tftpbboot` prefix to the kernel pathname.

---

### 2.1.6.2. /etc/hosts

IP address to hostname mapping

Computers generally communicate just fine with IP addresses. Then, us humans come along and need to put names on the computers, because we can't remember the numbers. That's where DNS or the `/etc/hosts` file come into play. This IP address to hostname mapping generally isn't required, except in an LTSP environment. That's because without it, NFS will give you permissions errors when the workstation attempts to mount the root filesystem.

---

### 2.1.6.3. /opt/ltsp/i386/etc/lts.conf

There are a number of configuration entries that can be specified in the `lts.conf` file.

The `lts.conf` file has a simple syntax, that consists of multiple sections. There is a default section called **[default]** and there can be sections for individual workstations. The workstations can be identified by hostname, IP address or MAC address.

## 2.1.6. Workstation specific configuration

A typical `lts.conf` file looks like this:

```
#
# Config file for the Linux Terminal Server Project (www.ltsp.org)
#

[Default]
    SERVER            = 192.168.0.254
    XSERVER            = auto
    X_MOUSE_PROTOCOL   = "PS/2"
    X_MOUSE_DEVICE     = "/dev/psaux"
    X_MOUSE_RESOLUTION = 400
    X_MOUSE_BUTTONS    = 3
    USE_XFS             = N
    LOCAL_APPS         = N
    RUNLEVEL            = 5

[ws001]
    USE_NFS_SWAP       = Y
    SWAPFILE_SIZE      = 48m
    RUNLEVEL            = 5

[ws002]
    XSERVER            = XF86_SVGA
    LOCAL_APPS         = N
    USE_NFS_SWAP       = Y
    SWAPFILE_SIZE      = 64m
    RUNLEVEL            = 3
```

## Example 2–1. `lts.conf` file

The following is a list of some of those entries:

### ***XSERVER***

If your video card is a PCI card, and if it is supported by XFree86 4.1, then you just need the `lts_x_core` package. That contains all of the driver modules for X4.

There are several XFree86 3.3.6 packages available for LTSP. This is in case your card isn't supported by XFree86 4.1.

You can make entries in the `lts.conf` for each individual workstation, or you can make default entries that are shared by all workstations.

Our workstation has an Intel i810 video chipset, and it can correctly be auto detected, so we don't need any `XSERVER` entry in the `lts.conf` file. The `XSERVER` entry can be specified, if you want, or it can be set to 'auto' to show that it is going to be auto detected.

### ***RUNLEVEL***

We want to run the workstation in graphical mode, so we need to set the runlevel to '5'. This is done by another entry in the `lts.conf` file.

---

## Chapter 3. Setting up the workstation

Once the server is setup, it is time to focus on setting up the workstation.

The LTSP project is all about what happens after the kernel is in memory. There are several ways to get the kernel into memory, including Etherboot, Netboot, PXE and floppy disk.

For the purposes of this document, we will use a boot floppy with code from the **Etherboot** project.

---

### 3.1. Creating the boot floppy

Etherboot is a software package for creating ROM images that can download code over an Ethernet network to be executed on an x86 computer. Many network adapters have a socket where a ROM chip can be installed. Etherboot is code that can be put in such a ROM.

—Ken Yap

Etherboot is also Open Source, protected under the GNU General Public License, Version 2 (GPL2).

You can download the Etherboot package and configure it for the type of bootrom that you need. Then, you can compile the source to produce a bootrom image that can be written to an eeprom, or written to a floppy disk for testing.

A much simpler alternative is to go to Marty Connor's [www.Rom-O-Matic.net](http://www.Rom-O-Matic.net) website.

Marty has done an excellent job of putting a web based front-end on the configuration and compilation process of making bootrom images with Etherboot. On his site, you select what type of network card you have, and what kind of image you want. Then, you have an opportunity to modify many Etherboot configuration options. Then, you can press the 'Get ROM' button and a custom bootrom image will be generated while you wait.

Our workstation has a Linksys LNE100TX, version 4.1 card. This card has a ADMTEK Centaur-P chipset, so we need to select *centaur-p* as our NIC/ROM type.

We don't need to make any changes from the default configuration, so you can skip the 'Configure' button.

For the ROM output format, choose 'Floppy Bootable ROM Image'. This will cause it to contain a 512 byte header that is a boot loader for loading the etherboot image into ram where it can be executed.

Press the 'Get ROM' button. The bootrom image will be generated while you wait. It only takes a few seconds, and when it is done, your browser will pop-up a "Save As" window where you can designate where you want the bootrom image to be saved on your computer.

I usually save the file in the /tmp directory. So, for the Centaur-P image, specify something like:  
/tmp/eb-5.0.2-centaur-p.lzdsk

Once you've saved the image to your hard drive, you need to write it out to a floppy disk. Insert a floppy diskette into the drive and run the following command to write the floppy:

```
cat /tmp/eb-5.0.2-centaur-p.lzdsk /dev/fd0
```

---





## Chapter 4. Running the workstation

Assuming that the server and workstation are configured properly, it should just be a matter of inserting the boot floppy in the floppy drive and turning on the workstation.

The Etherboot code will be read from the floppy into memory, the network card will be found and initialized, the dhcp request will be sent on the network and a reply will be sent from the server and the kernel will be downloaded to the workstation. Once the kernel has initialized the workstation hardware, X windows will start and a login box should appear on the workstation, similar to the example below.



**Figure 4–1. Login screen**

At this point, you can log in. An important thing to keep in mind is that you are logging into the server. All of the commands that you run are actually running on the server, and displaying their output on the workstation. That's the power of X windows.

You can run any program that is supported by the server.

---



# Chapter 5. Printing

Aside from the workstation being a fully functioning GUI or character mode terminal, it can also act as a print server, allowing upto 3 printers to be attached to the parallel and serial ports.

This is all transparent to the user of the workstation. They won't even notice the small amount of traffic that is going through the workstation to the printers.

---

## 5.1. Client side setup

LTSP uses the **lp\_server** program on the workstation, to redirect print jobs from the server to the printer attached to one of the ports on the workstation.

To enable the printer on the workstation, there are a set of configuration entries in the **lts.conf** file.

```
[ws001]
PRINTER_0_DEVICE = /dev/lp0
PRINTER_0_TYPE   = P
```

The above entry will cause the **lp\_server** program to run as a daemon, listening on TCP/IP port 9100 for a print stream from the server. The print data will then be redirected to the printer attached to the **/dev/lp0** parallel port.

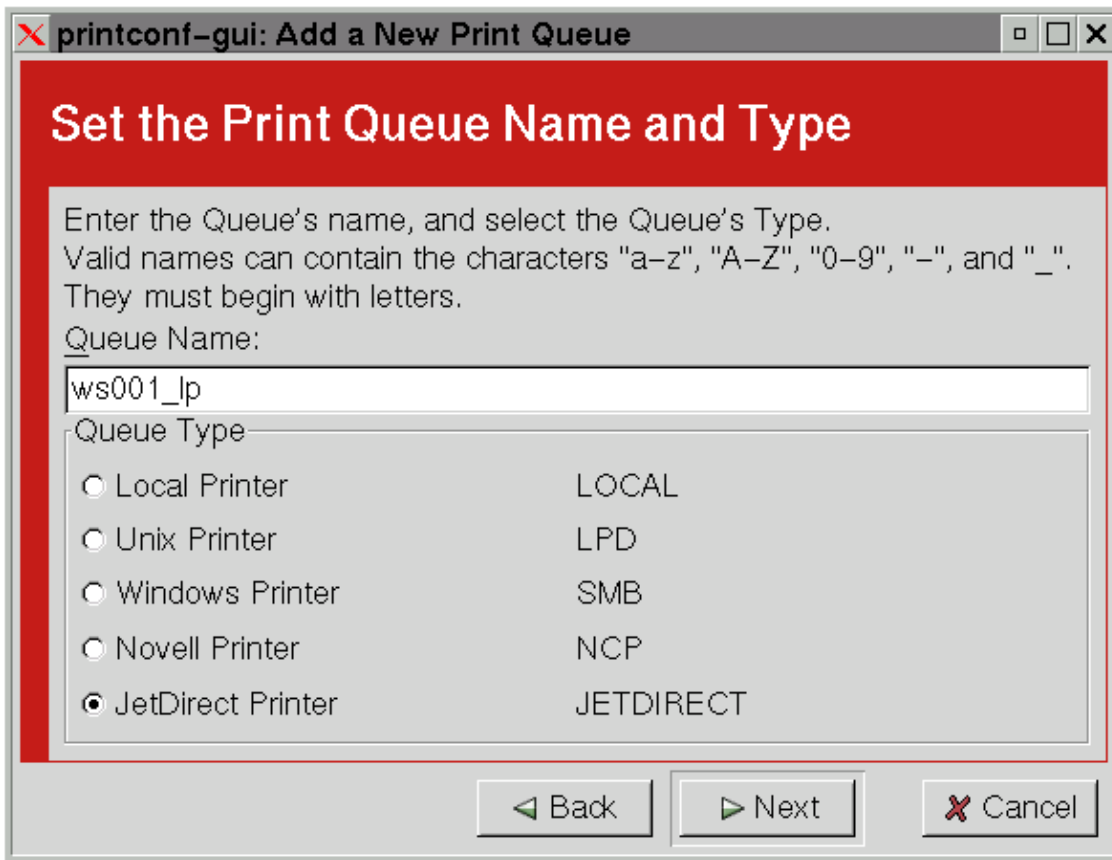
There are many more options available. Check the **lts.conf** section later in this document for more information on printer configuration entries.

---

## 5.2. Server side setup

Setting up the printer on the server is a matter of defining a print queue, using the printer configuration tool on the server.

On Redhat 7.2, there is both a GUI and a Text based printer configuration tool. The GUI tool is called **printconf-gui**, and the text based tool is called **printconf-tui**. Older versions of Redhat have a program called **printtool**. Printtool also exists on Redhat 7.2, but it will call **printconf-gui**. Other Linux distributions have their own printer configuration tool.



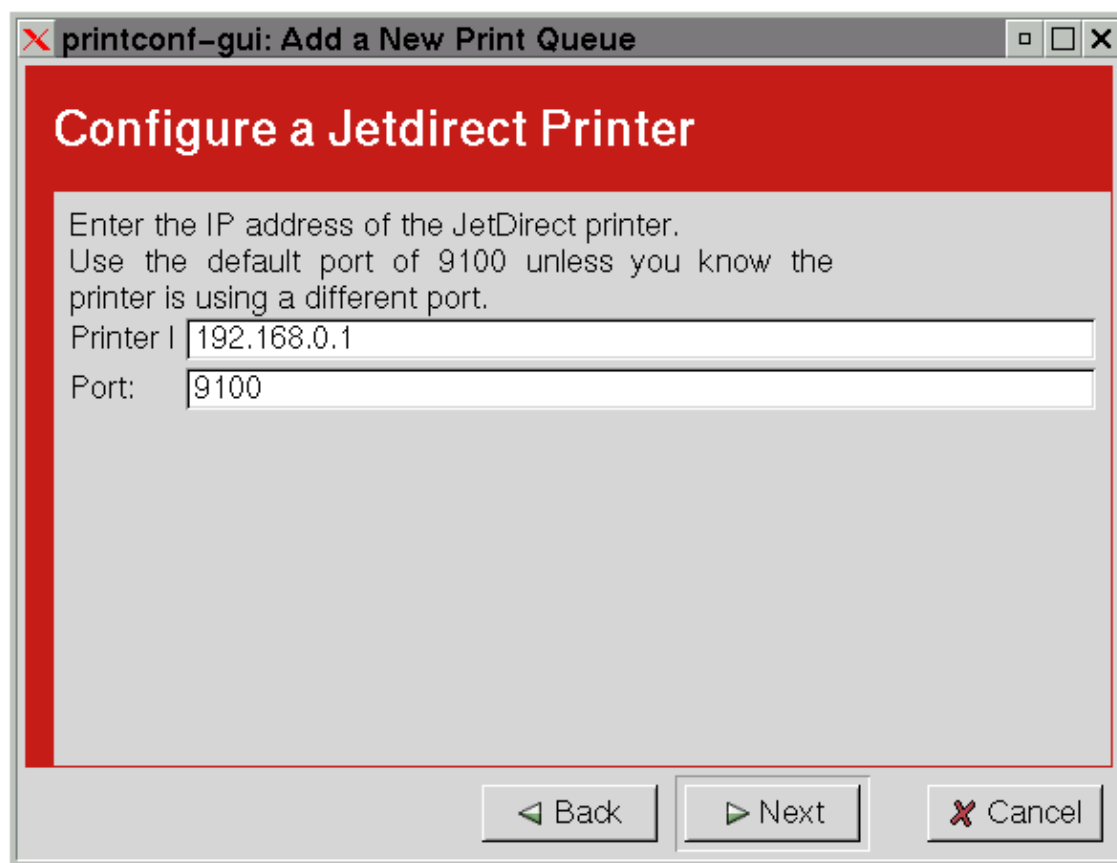
**Figure 5–1. Printconf–gui Adding new printer**

Once you launch the printer configuration tool, you need to add a new printer. The `lp_server` program allows the workstation to emulate an HP JetDirect print server. You just need to create a **JetDirect** printer.

You need to give the printer a Queue name. The name can be anything, but make it a meaningful name, and the name can contain only the following characters:

- "a-z" lower-case letters
- "A-Z" upper-case letters
- "0-9" numeric digits
- "-" hyphen
- "\_" underscore

The name chosen in the example above is **ws001\_lp**. This name makes it easy to see that the printer is associated with **ws001**.



**Figure 5–2. Printconf-gui Detail info**

There are two fields required to communicate with the printer:

1. IP Address or hostname of the workstation that the printer is associated with.
2. The TCP port that the **lp\_server** daemon is listening on.

The first printer you connect to a workstation will be on TCP/IP port **9100**. The second printer will be on port **9101**, and the third printer will be on port **9102**.

---



## Chapter 6. Trouble-shooting

If, after following through the previous chapters, your workstation doesn't boot, then you've got to start the process of trouble-shooting the installation.

The first thing to do is figure out how far through the bootup the workstation has gotten.

---

### 6.1. Trouble-shooting Etherboot floppy image

When you boot from the floppy, you should see something similar to this:

```
loaded ROM segment 0x0800 length 0x4000 reloc 0x9400
Etherboot 5.0.1 (GPL) Tagged ELF for [LANCE/PCI]
Found AMD Lance/PCI at 0x1000, ROM address 0x0000
Probing...[LANCE/PCI] PCnet/PCI-II 79C970A base 0x1000, addr 00:50:56:81:00:01
Searching for server (DHCP)...
<sleep>
```

The above example shows what you can expect to see on the screen when booting from a floppy. If you don't see those messages, indicating that Etherboot has started, then you may have a bad floppy disk, or you didn't write the image to it properly.

If, you see a message like the following, then it probably indicates that the Etherboot image you have generated is not the correct image for your network card.

```
ROM segment 0x0800 length 0x8000 reloc 0x9400
Etherboot 5.0.2 (GPL) Tagged ELF for [Tulip]
Probing...[Tulip]No adapter found
<sleep>
<abort>
```

If it does get to the point where it detects the network card and displays the proper MAC address, then the floppy is probably good.

---

### 6.2. Trouble-shooting DHCP

Once the network card is initialized, it will send out a DHCP broadcast on the local network, looking for a DHCP server.

If the workstation gets a valid reply from the DHCP server, then it will configure the network card. You can tell if it worked properly if the IP address information is displayed on the screen. Here's an example of what it should look like:

```
ROM segment 0x0800 length 0x4000 reloc 0x9400
Etherboot 5.0.1 (GPL) Tagged ELF for [LANCE/PCI]
Found AMD Lance/PCI at 0x1000, ROM address 0x0000
Probing...[LANCE/PCI] PCnet/PCI-II 79C970A base 0x1000, addr 00:50:56:81:00:01
Searching for server (DHCP)...
<sleep>
Me: 192.168.0.1, Server: 192.168.0.254, Gateway 192.168.0.254
```

If you see the line that starts with 'Me:', following by an IP address, then you know that DHCP is working properly. You can move on to checking to see if TFTP is working.

If instead, you see the following message on the workstation, followed by lots of <sleep> messages, then something is wrong. Although, it is common to see one or two <sleep> messages, after which the dhcp server replies.

```
Searching for server (DHCP)...
```

Figuring out what is wrong can sometimes be difficult, but here are some things to look for.

---

### 6.2.1. Check the connections

Is the workstation physically connected to the same network that the server is connected to?

With the workstation turned on, make sure that the link lights are lit at all of the connections.

If you are connecting directly between the workstation and the server (no hub or switch), make sure that you are using a cross-over cable. If you are using a hub or switch, then make sure you are using a normal straight-thru cable, both between the workstation and hub, and also between the hub and server.

---

### 6.2.2. Is dhcpd running?

You need to determine whether **dhcpd** is running on the server. We can find the answer a couple of ways.

**dhcpd** normally sits in the background, listening on udp port 67. Try running the **netstat** command to see if anything is listening on that port:

```
netstat -an | grep ":67 "
```

You should see output similar to the following:

```
udp      0      0  0.0.0.0:67          0.0.0.0:*
```

The 4th column contains the IP address and port, separated by a colon (':'). An address of all zeroes ('0.0.0.0') indicates that it is listening on all interfaces. That is, you may have an **eth0** and an **eth1** interface, and **dhcpd** is listening on both interfaces.

Just because netstat shows that something is listening on udp port 67, it doesn't mean that it is definately **dhcpd** that is listening. It could be **bootpd**, but that is unlikely, because **bootp** is no longer included on most distributions of Linux.

To make sure that it is the **dhcpd** that is running, try running the **ps** command.

```
ps aux | grep dhcpd
```

You should see something like the following:

```
root 23814 0.0 0.3 1676 820 ?        S 15:13 0:00 /usr/sbin/dhcpd
root 23834 0.0 0.2 1552 600 pts/0    S 15:52 0:00 grep dhcp
```

The first line shows that **dhcpd** is running. The second line is just our **grep** command.

If you don't see any lines showing that dhcpd is running, or

---

### 6.2.3. Double-check the dhcpd configuration

Does the `/etc/dhcpd.conf` file have an entry for our workstation?

You should double-check the 'fixed-address' setting in the config file, to make sure it exactly matches the card in the workstation.

---



## 6.2.4. Is ipchains or iptables blocking the request?

### 6.2.4.1. Checking for ipchains

Run the following command to see what it says:

```
ipchains -L -v
```

If you see something like this:

```
Chain input (policy ACCEPT: 229714 packets, 115477216 bytes):
Chain forward (policy ACCEPT: 10 packets, 1794 bytes):
Chain output (policy ACCEPT: 188978 packets, 66087385 bytes):
```

Then it isn't ipchains that is getting in the way.

---

### 6.2.4.2. Checking for iptables

Run the following command to see what it says:

```
iptables -L -v
```

If you see something like this:

```
Chain INPUT (policy ACCEPT 18148 packets, 2623K bytes)
  pkts bytes target      prot opt in      out     source            destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target      prot opt in      out     source            destination
Chain OUTPUT (policy ACCEPT 17721 packets, 2732K bytes)
  pkts bytes target      prot opt in      out     source            destination
```

Then it is not iptables getting in the way.

---

## 6.2.5. Is the workstation sending the request?

Try watching the `/var/log/messages` file while the workstation is booting. You can do that with the following command:

```
tail -f /var/log/messages
```

This will 'follow' the log file as new records are written to it.

```
server dhcpd: DHCPDISCOVER from 00:50:56:81:00:01 via eth0
server dhcpd: no free leases on subnet WORKSTATIONS
server dhcpd: DHCPDISCOVER from 00:50:56:81:00:01 via eth0
server dhcpd: no free leases on subnet WORKSTATIONS
```

If you see messages like those above, saying 'no free leases', that indicates that **dhcpd** is running, but it doesn't know anything about the workstation that is requesting an IP address.

---

## 6.3. Trouble-shooting TFTP

Etherboot uses TFTP to retrieve a Linux kernel from the server. This is a fairly simple protocol, but sometimes there are problems trying to get it to work.

If you see a message similar to this:

### 6.2.4. Is ipchains or iptables blocking the request?

```
Loading 192.168.0.254:/lts/vmlinuz.tulip |
```

with the last character on the line (the '|'), rapidly changing between a '|', '\', '-' and '/', to form what looks like a rotating bar, then the kernel is being downloaded. Which normally indicates that TFTP is working properly.

If, instead, you don't see the rotating bar, there is a problem. Possible problems include:

## 6.3.1. tftpd is not running

On Redhat 7.1, tftp is run by **xinetd**. There is a startup script called `/etc/xinetd.d/tftp` that contains the information for running **tftpd**

## 6.3.2. Kernel not where tftpd expects to find it

The kernel needs to be in a location that the tftpd daemon can access it. If the '-s' option is specified when **tftpd** is run, then whatever the workstation is asking for must be relative to the `/tftpboot`. So, if the **filename** entry in the `/etc/dhcpd.conf` file is set to `/lts/vmlinuz.tulip`, then the kernel actually needs to be `/tftpboot/lts/vmlinuz.tulip`

## 6.4. Trouble-shooting NFS root filesystem

There are several things that can prevent a root filesystem from being mounted. Including the following:

### 6.4.1. No init found

If you get the following error:

```
Kernel panic: No init found. Try passing init= option to kernel.
```

Then it is most likely that either you are mounting the wrong directory as the root filesystem, or the `ltsroot` directory is empty.

### 6.4.2. Server returned error -13

If you get the following error:

```
Root-NFS: Server returned error -13 while mounting /tftpboot/lts/ltsroot
```

This indicates that either the `/tftpboot/lts/ltsroot` directory isn't listed in the `/etc/exports` file.

Take a look in the `/var/log/messages` file to see if there are any clues. An entry like this:

```
Jul 20 00:28:39 jamlap rpc.mountd: refused mount request from ws004
for /tftpboot/lts/ltsroot (//): no export entry
```

Then it confirms our suspicion that the entry in `/etc/exports` isn't correct.

### 6.4.3. NFS Daemon problems (portmap, nfsd & mountd)

NFS can be a complex and difficult service to trouble-shoot, but understanding what should be setup and what tools are available to diagnose the problems will surely help to make it easier.

There are three daemons that need to be running on the server for NFS to work properly. **portmap**, **nfsd** and **mountd**.

## 6.4.3.1. The Portmapper (portmap)

If you get the following messages:

```
Looking up port of RPC 100003/2 on 192.168.0.254
portmap: server 192.168.0.254 not responding, timed out
Root-NFS: Unable to get nfsd port number from server, using default
Looking up port of RPC 100005/2 on 192.168.0.254
portmap: server 192.168.0.254 not responding, timed out
Root-NFS: Unable to get mountd port number from server, using default
mount: server 192.168.0.254 not responding, timed out
Root-NFS: Server returned error -5 while mounting /tftpboot/lts/ltsroot
VFS: unable to mount root fs via NFS, trying floppy.
VFS: Cannot open root device "nfs" or 02:00
Please append a correct "root=" boot option
Kernel panic: VFS: Unable to mount root fs on 02:00
```

This most likely is caused by the **portmap** daemon not running. You can confirm whether or not the portmapper is running by using the **ps** command:

```
ps -e | grep portmap
```

If the portmapper is running, you should see output like this:

```
30455 ?          00:00:00 portmap
```

Another test is to use the **netstat**. The portmapper uses TCP and UDP ports 111. Try running this:

```
netstat -an | grep ":111 "
```

You should see the following output:

```
tcp    0    0 0.0.0.0:111          0.0.0.0:*            LISTEN
udp    0    0 0.0.0.0:111          0.0.0.0:*
```

If you don't see similar output, then the portmapper isn't running. You start the portmapper by running:

```
/etc/rc.d/init.d/portmap start
```

Then, you should make sure that the portmapper is setup to start when the server boots. Run **ntsysv** to make sure it is selected to run.

## 6.4.3.2. The NFS and MOUNT daemons (nfsd & mountd)

NFS has 2 daemons that need to be running. **nfsd** and **mountd**. They are both started by the `/etc/rc.d/init.d/nfs` script.

You can run the **ps** command to make sure that they are running.

```
ps -e | grep nfs
ps -e | grep mountd
```

If it shows that one or both of the daemons are not running, then you will need to start them.

Normally, you should be able to run the startup script with the **restart** argument to cause them both to startup, but for some reason, the `/etc/rc.d/init.d/nfs` script doesn't restart **nfsd** that way. It only restarts **mountd** (bug?). So, you should instead run the following sequence of commands:

```
/etc/rc.d/init.d/nfs stop
/etc/rc.d/init.d/nfs start
```

You may get errors on the **stop** command, but that is Ok. The **start** command should show **OK** as the status.

If the daemons are running, but NFS is still not working, you can verify that they have registered themselves with the portmapper by using the **rpcinfo** command.

```
rpcinfo -p localhost
```

You should see results similar to the following:

program	vers	proto	port	
100000	2	tcp	111	portmapper
100000	2	udp	111	portmapper
100011	1	udp	856	rquotad
100011	2	udp	856	rquotad
100005	1	udp	1104	mountd
100005	1	tcp	2531	mountd
100005	2	udp	1104	mountd
100005	2	tcp	2531	mountd
100003	2	udp	2049	nfs

This indicates that **nfs** (nfsd) and **mountd** are both running and have registered themselves with the portmapper.

## 6.5. Trouble-shooting the Xserver

Oh boy!, Probably the single most difficult part of setting up an LTSP workstation is getting the X server configured properly. If you are using a fairly new video card, and it is supported by XFree86, and a fairly new monitor that can handle a large range of frequencies and resolutions, then it is pretty straight forward. Usually, in that case, if it doesn't work, it is most likely the wrong X server for that card.

When an X server doesn't work with your card, it is usually pretty obvious. Either the X server won't start, or the display will be incorrect.

When the workstation is ready to start the X server, it calls the `/tmp/start_ws` script, which starts the X server on the local workstation, with a **-query** option pointing to a server, where a display manager, such as **XDM**, **GDM** or **KDM** is running.

Because the X server is started by the `start_ws` script, which is itself started by the **init** program, when it fails, **init** will attempt to run it again. **init** will continue this loop of trying to run the X server 10 times, then give up, because it thinks that it is respawning too quickly. After it finally gives up, the error message from the X server should be left on the screen.

Waiting for the X server to fail 10 times can be rather irritating, so a simple way to avoid the repeated failures is to start the workstation in runlevel 3, so that the X server is NOT started automatically. Instead, when you boot the workstation, you will get a **bash** prompt. From the bash prompt, you can start the X server manually with the following command:

```
sh /tmp/start_ws
```

The X server will attempt to start, then when it fails, it will return back to the bash prompt, so you can see what the reason for the failure is.

## 6.6. Trouble–shooting the Display manager

The display manager is the daemon that runs on the server, waiting for an X server to make contact with it. Once contact has been made, it will display a login dialog box on the screen, offering the user a chance to log into the server.

The three most common display managers are:

- XDM – It's been around forever. It is included with the standard X windows system.
- GDM – The 'Gnome Display Manager'. This is part of the Gnome package.
- KDM – The 'KDE Display Manager'. This is part of the K Desktop system.

Most recent GNU/Linux distributions include all three display managers.

---

### 6.6.1. Grey screen with large X cursor

This indicates that the X server is running, but it has not been able to make contact with a display manager. Some possible reasons for that are:

1. The display manager may not be running

On Redhat 7.1, the display manager is started from **init**. In the `/etc/inittab` file, there is a line that looks like this:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

The **prefdm** script will make the determination of which display manager to run.

The default display manager depends on which packages have been installed. If Gnome is installed, then GDM is the default display manager. If Gnome is not installed, then the `prefdm` script will check to see if KDE is installed. If it is, then KDM will be the default display manager. If KDE also is not installed, then GDM will be the default display manager.

Using the **netstat** command, you should be able to see if there is a display manager running.

```
netstat -ap | grep xdmcp
```

You should see results showing that there is a process listening on the xdmcp port (177).

```
udp      0      0 *:xdmcp          *: *              1493/gdm
```

This shows clearly that **gdm** is running with a PID of 1493, and it is listening on the xdmcp port.

If you see a line like the one shown above, indicating that there is definately a display manager listening, then you need to make sure that the workstation is sending the XDMCP query to the correct server.

In the `lts.conf` file, you can have an entry which specifies the IP address of the server that is running the display manager. the entry is optional, but if present, should look like this:

```
XDM_SERVER = 192.168.0.254
```

of coarse, the IP address for your network may be different than the example above.

If the 'XDM\_SERVER' entry is not present, it will then use the value of the 'SERVER' entry, if present. If that is not present, then it will use **192.168.0.254**.

Which ever way it is specified, you just need to make sure that the IP address is actually the correct address of the server running the display manager.

2. The display manager may be configured to ignore requests from remote hosts.

If you've determined that the display manager is running, then it is possible that it has been configured to ignore XDMCP requests from remote hosts. You will need to check the configuration files of the particular running display manager, to determine if it is configured properly.

## ◆ XDM

The default configuration for Redhat is to disable the ability for workstations to get login access from XDM. The **ltsp\_initialize** script will take care of enabling this for you, but if it's not working, you should check the `/etc/X11/xdm/xdm-config` file. Look for an entry that looks like this:

```
DisplayManager.requestPort:      0
```

This entry **MUST** be commented out in order for XDM to listen on port 177 for remote requests.

Another configuration file is also important for XDM to serve up remote login requests. There is a file called `/etc/X11/xdm/Xaccess` that **MUST** have a line that starts with an asterisk '\*'. the line is normally included in the file, but Redhat leaves the line commented out. the **ltsp\_initialize** script will fix the line for you, but if XDM doesn't seem to be working, you should check this file. A valid line should look like this:

```
*          #any host can get a login window
```

## ◆ KDM

Newer versions of KDM have a file called **kdmrc**. Different Linux distributions store that file in different locations. For Redhat 7.2, it is `/etc/kde/kdm/kdmrc`. For the other distros, you should run the **locate** to find out where it is stored.

The entry that controls whether remote workstations can get a login is in the **[Xdmcp]** section. Make sure that the **Enable** entry is set to **true**.

Older versions of KDM use the XDM configuration files, located in `/etc/X11/xdm`.

## ◆ GDM

GDM uses a different set of configuration files. They are located in the `/etc/X11/gdm` directory.

The main file to look at is the `gdm.conf` file. Look for the **[xdmcp]** section. you should see an entry within that section called 'Enable'. It must be set to '1'. Here is an example:

```
[xdmcp]
Enable=true
HonorIndirect=0
MaxPending=4
MaxPendingIndirect=4
MaxSessions=16
MaxWait=30
MaxWaitIndirect=30
Port=177
```

Notice the 'Enable=true' line. Older versions of GDM use '0' and '1' to signify whether to Disable or Enable the remote XDMCP. Newer versions use 'false' and 'true'.

---





# Chapter 7. Kernels

There are a few decisions that need to be made about the kernel that is run on the workstation. You need to decide whether you want to run one of the standard kernels available for download, or build your own. And, you need to decide if you want to display the graphic screen, complete with the progress bar which, which is made possible by the **Linux Progress Patch (LPP)**.

---

## 7.1. Standard LTSP supplied kernels

The kernel package that is available with LTSP actually includes two kernels. One has the Linux Progress Patch already applied and configured, and the other does not have the patch applied.

Both of the kernels already have the NFS Swap patch applied.

---

## 7.2. Build your own kernel

There are a few choices you can make, when deciding whether to use the standard which for the kernel

There are two ways to configure a kernel for LTSP. The default method is to use something called an 'Initial Ram Disk', or **initrd** for short. The initrd image is a small filesystem that appended to the kernel. The initrd filesystem image is loaded into memory, and once the kernel is booted, it will mount the ramdisk as its root filesystem. There are a couple of advantages of using an initrd image. First, we can compile the network drivers as modules and load the correct module during bootup. This allows a single kernel which will support virtually all network cards. The other advantage is that we can run the DHCP client as a "user-land" program rather than in kernel-space. Running the client in user-land provides better control over the options requested and received from the server. Also, it makes the kernel slightly smaller. The other way to configure the kernel is without the initrd. Building a kernel without an initrd requires that the specific network card driver be statically linked into the kernel, and it also requires that IP-Autoconfig and "Root filesystem on NFS" are set when building the kernel. The advantage of not using an initrd is that the kernel is slightly smaller, and it will boot slightly faster. Once the workstation is up and running, there is virtually no difference in how the workstation functions.

The standard kernel for LTSP includes an Initial Ramdisk (initrd) that takes care of detecting the network card, and making a user-space DHCP request. A major goal for the image was to make it as small as possible. So, we chose the uClinux libc replacement library, and busybox for the utilities that we need during the boot.

If you want to build your own kernels, you should download the `ltsp_initrd_kit` package. It contains the root filesystem hierarchy, and a script for building the image.

---

### 7.2.1. Obtaining the source for the kernel

When building a custom kernel, it's usually a good idea to start with fresh kernel sources direct from the **ftp.kernel.org** site. The reason for this is that the distributions, like Redhat, apply many patches to their kernel sources, leaving you with a set of source code that really doesn't match that of the official kernel.

Download the kernel source package of your choice, and save it in the `/usr/src` directory. The kernels are located in the `/pub/linux/kernel` directory of the `ftp.kernel.org` ftp server. You will need to grab a recent 2.4.x series kernel, because you need to include **devfs** support.

Also, if you want to include support for swapping over NFS or the Linux Progress Patch (LPP), you will need to make sure that you get the patches and kernel sources that are all the same. At the time of this writing, the 2.4.9 kernel is the latest to support those features.

For our example, we will use the 2.4.9 kernel. The entire path is  
`ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.9.tar.bz2`

Unpack the kernel sources in the `/usr/src` directory. You need to be careful, because when you un-tar the package, it will be in a directory called `linux`. You may already have a directory called `linux` with a different set of source code, and you don't want to clobber it. So, check for an existing directory, and if it is there, rename it to something else before unpacking the sources.

The source package we downloaded had been compressed with the **bzip2** compression utility. So, we need to uncompress it before we feed it to the **tar** program. You can use the following command to unpack it:

```
bunzip2 <linux-2.4.9.tar.bz2 | tar xf -
```

When the unpacking completes, you will have a directory called `linux` containing the entire source tree. At this point, I usually like to rename the directory to something more meaningful.

```
mv linux linux-2.4.9
```

Once the directory has been renamed, then change into the new directory:

```
cd linux-2.4.9
```

I usually like to modify the `Makefile` before I start configuring the new kernel. Near the top of the file is a variable named **EXTRAVERSION**. I set this to `'ltsp-1'`, so that the actual version number of the kernel will be `'2.4.9-ltsp-1'`, which makes it easier to identify the kernel later. The top of the `Makefile` should look like this when you are done:

```
VERSION = 2
PATCHLEVEL = 4
SUBLEVEL = 9
EXTRAVERSION = -ltsp-1

KERNELRELEASE=$(VERSION) . $(PATCHLEVEL) . $(SUBLEVEL) $(EXTRAVERSION)
```

---

### 7.2.2. Kernel Patches

After unpacking the kernel, you may have various patches you want to apply. For example, the NFS Swap patch or the Linux Progress Patch. These patches **MUST** be applied before configuring the kernel.

---

#### 7.2.2.1. NFS Swap patch

The NFS Swap patch will allow the workstation kernel to use a swapfile located on an NFS server. While it usually recommended to have enough memory in the workstation to not require swapping, it can sometimes be difficult to add more memory, especially on older computers. So, the ability to swap over NFS can make an otherwise unuseable computer actually useable.

If the current directory is `/usr/src/linux-2.4.9`, and the patch is in `/usr/src`, you can do the following to test the patch:

```
patch -p1 --dry-run <../linux-2.4.9-nfs-swap.diff
```

This will test the patch, to make sure it can be applied cleanly. If it finishes with no errors, then you can apply the patch without the **--dry-run** option.

```
patch -p1 <../linux-2.4.9-nfs-swap.diff
```

---

### 7.2.2.2. Linux Progress Patch (LPP)

The Linux Progress Patch (LPP) will allow you to configure a graphic logo to display during the boot process. The normal kernel boot messages are redirected to another tty screen, and special instructions are added to the boot scripts to cause the Progress bar to reflect how far along the boot process is.

Like the NFS Swap patch, you can test the LPP patch by issuing this command:

```
patch -p1 --dry-run <../lpp-2.4.9
```

If the test completes successfully, then you can apply the patch with:

```
patch -p1 <../lpp-2.4.9
```

---

## 7.2.3. Configuring kernel options

You can now run the kernel configuration program of your choice. Available choices are:

- make xconfig

This will invoke the X Windows version of the kernel configuration utility.

- make menuconfig

This will invoke the curses based version of the kernel configuration utility.

- make config

This will invoke the simple line-at-a-time version of the kernel configuration utility.

---

### 7.2.3.1. Kernel configuration for use with an initrd

Configuring the kernel for use with an initrd requires the following options to be set:

- File systems → /dev filesystem support

/dev file system support must be enabled. This is selected in the 'File systems' section. Do NOT specify 'Automatically mount at boot'. The mount will be performed by the /linuxrc script.

- Block devices → RAM disk support

LTSP workstations require that the kernel support a RAM disk. this is set in the 'Block devices' section.

- Block devices → Initial RAM disk (initrd) support

This must also be enabled.

- Processor type and features → Processor family

You need to make sure that the kernel you build can actually run on the CPU in the workstation. This is done in the 'Processor type and features' section. You should also turn off SMP support, unless you actually have multiple CPUs.

- File systems → Network file systems → NFS Client support

The workstation will be mounting its root filesystem via NFS, so NFS client support is required.

That should take care of the required options. You can also turn off many features of the kernel, to reduce the size of the kernel.

---

### 7.2.3.2. Kernel configuration for use without an initrd

Configuring the kernel for use without an initrd differs from a kernel with an initrd in a few ways:

- Block devices → RAM disk support

LTSP workstations require that the kernel support a RAM disk.

- Block devices → Initial RAM disk (initrd) support

This needs to be disabled.

- Networking options → IP:kernel level autoconfiguration

This needs to be enabled. This will instruct the kernel to automatically configure the eth0 ethernet interface, based on values passed on the kernel command line.

It is not necessary to specify the DHCP, BOOTP or RARP options because the Etherboot bootrom has already done a DHCP or BOOTP request, and it will make the IP parameters available on the kernel command line. This saves the kernel from going through the trouble of doing its own query.

- Network device support → Ethernet (10 or 100Mbit)

When not using an initrd, you must choose the specific network card driver that matches the network card. This **MUST** be statically linked into the kernel, because the ethernet interface is needed before mounting the root filesystem. This is a major difference between how a kernel with an initrd works.

- File systems → /dev filesystem support

As of LTSP version 2.09pre2, **devfs** support is needed. This is true, regardless of whether an initrd is used or not.

- File systems → Automatically mount at boot

When NOT using an initrd, the /dev filesystem must be mounted by the kernel, during the boot process. So, say 'Y' here.

- File systems → Network file systems → NFS Client support

The workstation will be mounting its root filesystem via NFS, so NFS client support is required.

---

### 7.2.4. Building the kernel

To make things easier, a copy of the `.config` file is included in the `ltsp_initrd_kit` package. You can copy that to the `/usr/src/linux-2.4.9` directory.

Once you are done selecting or de-selecting the kernel options, you need to build the kernel. The following commands need to be executed to build the kernel:

```
make dep
```

```
make clean
make bzImage
make modules
make modules_install
```

You can string them all together like this:

```
make dep && make clean && make bzImage && make modules && make modules_install
```

The double ampersand (&) means that if the first command completes successfully, then the second command will be executed. If the second command completes successfully, then the third command will be executed, and so on.

When the kernel compilation is finished, the new kernel will be sitting in  
`/usr/src/linux-2.4.7/arch/i386/boot/bzImage`.

---

### 7.2.5. Tagging the kernel for Etherboot

For Etherboot to handle a Linux kernel, it needs to be prepared. This is called 'Tagging' the kernel. This process will add some additional code to the kernel that is executed before control is passed to the kernel. The tool for tagging a kernel is called '**mkbni-linux**'.

The `ltsp_initrd_kit` includes a shell script called **buildk** that includes all of the commands that you need to prepare the kernel image for network booting.

---



## Chapter 8. Its.conf entries

When we designed the LTSP, one of the issues that we knew that we would have to deal with varying hardware configurations for the workstations. Certainly, whatever combination of processor, network card and video card available today would not be available in 3 months, when we want to add more workstations to the network.

So, we devised a way of specifying the configuration of each workstation. The configuration file is called `lts.conf` and it lives in the `ltsroot/etc` directory.

The format of the `lts.conf` allows for 'default' settings and individual workstation settings. If all of your workstations are identical, you could specify all of the configuration settings in the '[Default]' section.

---

### 8.1. Sample Its.conf file

Here is an example of the `lts.conf` file:

```
[Default]
    SERVER                = 192.168.0.254
    X_MOUSE_PROTOCOL      = "PS/2"
    X_MOUSE_DEVICE        = "/dev/psaux"
    X_MOUSE_RESOLUTION    = 400
    X_MOUSE_BUTTONS       = 3
    USE_XFS                = N
    RUNLEVEL              = 5

[ws001]
    XSERVER               = auto
    X_MOUSE_PROTOCOL      = "Microsoft"
    X_MOUSE_DEVICE        = "/dev/ttyS1"
    X_MOUSE_RESOLUTION    = 50
    X_MOUSE_BUTTONS       = 3
    X_MOUSE_BAUD          = 1200

[ws002]
    XSERVER               = XF86_Mach64

[ws003]
    RUNLEVEL              = 3
```

---

### 8.2. Available Its.conf parameters

#### 8.2.1. General parameters

##### *Comments*

Comments start with the hash '#' sign and continue through the end of the line.

##### ***LTSP\_BASEDIR***

This indicates where the LTSP root filesystems are located. The default value is `/opt/ltsp`

##### ***SERVER***

This is the server that is used for the `XDM_SERVER`, `TELNET_HOST`, `XFS_SERVER` and `SYSLOG_HOST`, if any of those are not specified explicitly. If you have one machine that is acting as the server for everything, then you can just specify the address here and omit the other server parameters. If this value is not set, **192.168.0.254** will be used.

##### ***SYSLOG\_HOST***

If you want to send logging messages to a machine other than the default server, then you can specify the machine here. If this parameter is NOT specified, then it will use the 'SERVER' parameter described above.

### ***NFS\_SERVER***

This specifies the IP address of the NFS server used when the /home filesystem is mounted. The default is whatever the **SERVER** is set to.

### ***USE\_NFS\_SWAP***

Set this to **Y** if you want to turn on NFS swap. The default is **N**

### ***SWAPFILE\_SIZE***

This is how you can control the size of the swapfile. The default is **64m**.

### ***SWAP\_SERVER***

The swapfile can exist on any server on the network that is capable of handling it. You can specify the IP address of that server. The default is whatever the value of **NFS\_SERVER** set to.

### ***NFS\_SWAPDIR***

The directory on the server that is being exported via NFS. The default is /var/opt/ltsp/swapfiles. Make sure that the directory is being exported in the /etc/exports file.

### ***TELNET\_HOST***

If the workstation is setup to have a character based interface, then the value of this parameter will be used as the host to telnet into. If this value is NOT set, then it will use the value of **SERVER** above.

### ***DNS\_SERVER***

Used to build the resolv.conf file.

### ***SEARCH\_DOMAIN***

Used to build the resolv.conf file.

### ***MODULE\_01 thru MODULE\_10***

Upto 10 kernel modules can be loaded by using these configuration entries. The entire command line that you would use when running insmod can be specified here. For example:

```
MODULE_01 = uart401.o
MODULE_02 = sb.o io=0x220 irq=5 dma=1
MODULE_03 = opl3.o
```

If the value of this parameter is an absolute pathname, then **insmod** will be used to load the module. Otherwise, **modprobe** will be used.

### ***RAMDISK\_SIZE***

When the workstation boots, it creates a ramdisk and mounts it on the /tmp directory. You can control the size of the filesystem with this parameter. Specify it in units of kbytes (1024 bytes). To create a ramdisk of 1 megabyte, specify **RAMDISK\_SIZE = 1024**

If you change the size of the ramdisk here, you will also need to change the size of the ramdisk within the kernel. This can be compiled in, or if you are using Etherboot or Netboot, you tell the kernel the ramdisk size when you tag the kernel with mknbi-linux.

The default value for this is 1024 ( 1 mb )

### ***RCFILE\_01 thru RCFILE\_10***

Additional RC scripts can be executed by the rc.local script. Just put the script in the /etc/rc.d directory, and specify the name of the script in one of these entries.



**SOUND**

If the LTSP Sound package is installed, you need to set this entry to **Y** and it will execute the **rc.sound** script to setup the sound card and daemon. The default is **N**.

---

**8.2.2. X–Windows parameters****XDM\_SERVER**

If you want to point XDM to a machine other than the default server, then you can specify the server here. If this parameter is NOT specified, then it will use the 'SERVER' parameter described above.

**XSERVER**

This defines which X server the workstation will run. For PCI and AGP video cards, this parameter should not be required. The rc.local script should be able to auto–detect the card. You can also set this value to **auto** to indicate that it should attempt to auto–detect the card.

For ISA video cards, or to force a specific X server, you can specify the name of the driver or Xserver in this entry.

If the value begins with **XF86\_**, then XFree86 3.3.6 will be used. Otherwise, XFree86 4.1.x will be used. The default value for this is **auto**.

**X\_MODE\_0 through X\_MODE\_2**

Up to 3 Modelines or resolutions can be configured for the workstation. This entry can take two different types of values. It can take either a resolution, or a complete modeline.

```
X_MODE_0 = 800x600

or

X_MODE_0 = 800x600 60.75 800 864 928 1088 600 616 621 657 -HSync -VSync
```

If none of the X\_MODE\_x entries are not specified, then it will use the built–in modelines, and the resolutions of 1024x768, 800x600 and 640x480.

If one or more X\_MODE\_x entries are specified, they will completely override any built–in modelines.

**X\_MOUSE\_PROTOCOL**

Any value that will work for the XFree86 Pointer Protocol keyword can be put here. Typical values include "Microsoft" and "PS/2". The default value for this is **"PS/2"**.

**X\_MOUSE\_DEVICE**

This is the device node that the mouse is connected to. If it is a serial mouse, this would be a serial port, such as **/dev/ttyS0** or **/dev/ttyS1**. If it is a PS/2 keyboard mouse, this value would be **/dev/psaux**. The default value for this is **/dev/psaux**.

**X\_MOUSE\_RESOLUTION**

This is the 'Resolution' value in the **XF86Config** file. A typical value for a serial mouse is **50** and a typical value for a PS/2 mouse would be **400**. The default value for this is **400**.

**X\_BUTTONS**

This tells the system how many buttons the mouse has. Usually set to **2** or **3**. The default value for this is **3**.

**X\_MOUSE\_EMULATE3BTN**

This tells the X server to emulate a 3–button mouse by accepting a click of both the right and left mouse buttons simultaneously. The default value is **N**.

### ***X\_MOUSE\_BAUD***

For serial mice, this defines the baud rate. The default value for this is **1200**.

### ***X\_COLOR\_DEPTH***

This is the number of bits to use for the color depth. Possible values are **8, 15, 16, 24** and **32**. 8 bits will give 256 colors, 16 will give 65536 colors, 24 will give 16 million colors and 32 bits will give 4.2 billion colors! Not all X servers support all of these values. The default value for this is **16**

### ***USE\_XFS***

You have a choice of running the X Font Server (XFS) or reading the fonts through the NFS filesystem. The font server should provide a simple way of keeping all of the fonts in one place, but there has been some problems when the number of workstations grows past about 40. The 2 values for this option are **Y** and **N**. The default value is **N**. If you do want to use a font server, then you can use the **XFS\_SERVER** entry to specify which host will act as the font server.

### ***XFS\_SERVER***

If you are using an X Font Server to serve fonts, then you can use this entry to specify the IP address of the host that is acting as the font server. If this is not specified, it will use the default server, which is specified with the **SERVER** entry described above.

### ***X\_HORZSYNC***

This sets the XFree86 **HorizSync** configuration parameter. It defaults to "**31–62**".

### ***X\_VERTREFRESH***

This sets the XFree86 **VertRefresh** configuration parameter. It defaults to "**55–90**".

### ***XF86CONFIG\_FILE***

If you want to create your own complete XF86Config file you can do so and place it in the **/tftpboot/lts/ltsroot/etc** directory. Then, whatever you decide to call it needs to be entered as a value for this configuration variable. For example:

```
XF86CONFIG_FILE = XF86Config.ws004
```

---

## **8.2.3. Touch screen parameters**

### ***USE\_TOUCH***

If you are connecting a touch screen to the workstation, you can enable it by setting this entry to **Y**. If enabled, additional configuration entries will configure specific aspects of the touch screen. The default value is **N**.

### ***X\_TOUCH\_DEVICE***

A touch screen works like a mouse and usually is interfaced with the workstation through a serial port. You can specify which serial port with this entry. For example, you could set it equal to **/dev/ttyS0**. There is no default value for this entry.

### ***X\_TOUCH\_MINX***

Calibration entry for an EloTouch touch screen. Defaults to **433**.

### ***X\_TOUCH\_MAXX***

Calibration entry for an EloTouch touch screen. Defaults to **3588**.

### ***X\_TOUCH\_MINY***

Calibration entry for an EloTouch touch screen. Defaults to **569**.

### ***X\_TOUCH\_MAXY***

Calibration entry for an EloTouch touch screen. Defaults to **3526**.

***X\_TOUCH\_UNDELAY***

Calibration entry for an EloTouch touch screen. Defaults to **10**.

***X\_TOUCH\_RPTDELAY***

Calibration entry for an EloTouch touch screen. Defaults to **10**.

---

## 8.2.4. Local apps parameters

***LOCAL\_APPS***

If you want the ability to run applications locally on a workstation, set this variable to **Y**. Several additional steps must be taken on the server to enable local apps. See the 'Local Apps' section in the LTSP manual for more information. The default value is **N**.

***NIS\_DOMAIN***

If you do setup LOCAL\_APPS, then you need to have an NIS server on the network. The NIS\_DOMAIN entry is where you specify the NIS domain name. It needs to match a domain name that has been defined on the NIS server. This is NOT the same thing as an internet DOMAIN. The default value is **ltsp**.

***NIS\_SERVER***

Set this to the IP address of your NIS server if you don't want it to send a broadcast looking for an NIS server.

---

## 8.2.5. Keyboard parameters

All of the keyboard support files are now copied into the ltsroot hierarchy so configuring international keyboard support is now a matter of configuring XFree86. Several configuration parameters are available to make this possible.

The values for the above parameters are from the XFree86 documentation. Whatever is valid for XFree86 is valid for these parameters.

We would like to add documentation to show what values are needed for each type of international keyboard. If you work with this and can configure your international keyboards, feedback to the ltsp core group would be greatly appreciated.

***XkbTypes***

The default value for this is the word '**default**'.

***XkbCompat***

The default value for this is the word '**default**'.

***XkbSymbols***

The default value for this is '**us(pc101)**'.

***XkbModel***

The default value for this is '**pc101**'.

***XkbLayout***

The default value for this is '**us**'.

---

## 8.2.6. Printer configuration parameters

Upto three printers can be connected to a diskless workstation. A combination of serial and parallel printers can be configured via the following entries in the **lts.conf** file:

***PRINTER\_0\_DEVICE***

The device name of the first printer. Names such as ***/dev/lp0***, ***/dev/ttyS0*** or ***/dev/ttyS1*** are allowed.

***PRINTER\_0\_TYPE***

The type of the printer. Valid choices are ***'P'*** or for Parallel, and ***'S'*** for Serial.

***PRINTER\_0\_PORT***

The TCP/IP Port number to use. By default, it will use ***'9100'***

***PRINTER\_0\_SPEED***

If the printer is serial, this is the setting that will select the baud rate. By default, ***'9600'*** will be used.

***PRINTER\_0\_FLOWCTRL***

For serial printers, the flow control can be specified. Either ***'S'*** for Software (XON/XOFF) flow control, or ***'H'*** for Hardware (CTS/RTS) flow control. If neither is specified, ***'S'*** will be used.

***PRINTER\_0\_PARITY***

For serial printers, the Parity can be specified. The choices are: ***'E'***–Even, ***'O'***–Odd or ***'N'***–None. If not specified, ***'N'*** will be used.

***PRINTER\_0\_DATABITS***

For serial printers, the number of data bits can be specified. The choices are: ***'5'***, ***'6'***, ***'7'*** and ***'8'***. If not specified, ***'8'*** will be used.

***PRINTER\_1\_DEVICE***

Second printer device name

***PRINTER\_1\_TYPE***

Second printer device type

***PRINTER\_1\_PORT***

Second printer device TCP/IP port

***PRINTER\_1\_SPEED***

Second printer baud rate (serial)

***PRINTER\_1\_FLOWCTRL***

Second printer flow control (serial)

***PRINTER\_1\_PARITY***

Second printer parity (serial)

***PRINTER\_1\_DATABITS***

Second printer data bits (serial)

***PRINTER\_2\_DEVICE***

Third printer device name

***PRINTER\_2\_TYPE***

Third printer device type

***PRINTER\_2\_PORT***

Third printer device TCP/IP port

***PRINTER\_2\_SPEED***

Third printer baud rate (serial)

***PRINTER\_2\_FLOWCTRL***

Third printer flow control (serial)

***PRINTER\_2\_PARITY***

Third printer parity (serial)

***PRINTER\_2\_DATABITS***

Third printer data bits (serial)

---



# Chapter 9. Local Applications

In an LTSP environment, you have a choice of running the applications locally on the workstation, or remotely on the server.

By far, the easiest way to setup an LTSP environment is to run the apps on the server. That is, the client application runs on the server, using the servers' CPU and memory, while it displays its output on the workstation and uses the workstations' keyboard and mouse.

This is a fundamental capability of X Windows. The workstation works just like a standard X Windows terminal.

In order for a user to run an application on the workstation, the workstation needs to know some information about the user. Info such as the following:

- User id
- Primary group that the user belongs to
- The users home directory

LTSP relies on the Network Information Service – NIS, (formerly called *Yellow Pages*) to make the user and group information available to the workstations.

---

## 9.1. Benefits of running apps locally

There are benefits to running applications on the workstation.

- Reduces the load on the server. In large networks with memory intensive applications, such as Netscape, running the app on the workstation can provide better performance, as long as the workstation is powerful enough to handle it.
- Runaway apps will not affect other users.
- Sound support is much easier to configure when the application that plays the sound is running on the workstation.

---

## 9.2. Issues with setting up support for local apps

Setting up the ability to run applications locally requires much more.

- Greater demand on the workstation. It needs more RAM and a more powerful CPU. 64mb of ram on the workstation is a pretty good starting point.
- NIS – To run the apps on the workstation, you first must identify yourself to the workstation. That is, the workstation needs to know who you are. This requires some form of password authentication. NIS has been chosen as the method of authenticating users over the network.
- Additional directories need to be exported from the workstation to mount via NFS.
- Slower startup of applications, because they need to be read via NFS, causing increased network activity. Also, because each copy of the program is running on its own CPU, you won't get the advantage of the ability of Linux/Unix to share code segments between multiple instances of the same program, which would reduce the time it takes for second and succeeding invocations of the program.

---

## 9.3. Server Configuration for Local Apps

### 9.3.1. Its.conf entries

A few entries need to be setup in the `lts.conf` file:

**LOCAL\_APPS**

This must be set to **Y**. It will cause the following to happen during the workstation boot process:

1. The `/home` directory on the server will be mounted via NFS.
2. The `/var/yp/nicknames` will be created on the workstation.
3. The **portmapper** will be started on the workstation.
4. **xinetd** will be started on the workstation.
5. The `/etc/yp.conf` file will be created on the workstation.
6. The **domainname** command will be run with the value of the **NIS\_DOMAIN** `lts.conf` entry.
7. The **ypbind** will be run on the workstation.

### **NIS\_DOMAIN**

With NIS, all of the nodes on the network that want to be associated with a specific NIS server need to belong to the same NIS domain (this is not related in any way to a DNS Domain). You use the **NIS\_DOMAIN** entry to specify the name of the NIS domain that the workstation will belong to.

### **NIS\_SERVER**

NIS will either attempt to bind with a specific NIS server, or it will send a broadcast out to the network, looking for a server. If you want to choose a specific server, then enter the IP address of that server in the **NIS\_SERVER** entry.

---

## 9.3.2. Network Information Service – NIS

NIS is a Client/Server type of service. On the server, there is a daemon running, that will accept requests from the clients (workstations). The daemon on the server is called **ypserv**.

On the workstation, there is a process called **ypbind**. When the workstation needs to lookup information about a user, such as verifying a password or finding the users home directory, it will use **ypbind** to establish a connection to **ypserv**, on the server.

If you are already running NIS in your network environment, then there is no need to configure the LTSP server to also run **ypserv**. You can just configure the **NIS\_DOMAINNAME** and **NIS\_SERVER** entries in `lts.conf` to match your current NIS scheme.

If you are NOT already running NIS in your network, then you will need to configure the server to run **ypserv**.

For complete information on setting up an NIS server, refer to the HOWTO document on the LDP site called ***The Linux NIS(YP)/NYS/NIS+ HOWTO***. Refer to the list of other sources of information at the end of this document.

---

## 9.4. Application Configuration

To setup an application to run on the workstation, you need to put all of the components of the application in a place where the workstation can see it.

With older versions of LTSP (2.08 and earlier), lots of directories were exported on the server and mounted by the workstation. Directories such as `/bin`, `/usr/bin`, `/lib` and `/usr` were exposed to the workstation.

The problem with that scheme is that it only works if the workstation and the server are the same architecture. In fact, even differences, like the server being a Pentium II (i686) and the workstation being a classic Pentium (i586) can be a problem, because the server will likely have the i686 libraries and not the i386, i486 or i586 libraries.

So, the cleanest way to handle this is to have a complete tree with all of the binaries and libraries that the workstation will need, independent of the server binaries and libraries.

Configuring an application for local execution requires putting all of the required pieces in the tree. One of the packages available for download from the LTSP site is the Local netscape package, installs alot of files into the `/opt/ltsp/i386/usr/local/netcape` directory. Things like java classes, help files, executable binary files



and scripts are put there.

Netscape doesn't need any additional system libraries, so there is nothing to add to the `/opt/ltsp/i386/lib` directory. Many applications do require additional libraries.

So, how can you determine what libraries are needed? That's where the **ldd** command comes in handy.

Lets assume that you want to setup a certain application to run locally. We'll pick **gaim** as an example. **gaim** is an AOL Instant Messenger client, that will allow you to communicate with other people on AOL forums.

The first thing you need to do is find the **gaim** executable binary file. On a Redhat 7.2 system, it is located in the `/usr/bin` directory.

Once you locate the **gaim** binary, you can run **ldd** against it:

```
[jam@server /]$ ldd /usr/bin/gaim
        libaudiofile.so.0      => /usr/lib/libaudiofile.so.0 (0x40033000)
        libm.so.6              => /lib/i686/libm.so.6 (0x40051000)
        libnsl.so.1            => /lib/libnsl.so.1 (0x40074000)
        libgnomeui.so.32       => /usr/lib/libgnomeui.so.32 (0x4008a000)
        libart_lgpl.so.2       => /usr/lib/libart_lgpl.so.2 (0x4015d000)
        libgdk_imlib.so.1      => /usr/lib/libgdk_imlib.so.1 (0x4016c000)
        libSM.so.6             => /usr/X11R6/lib/libSM.so.6 (0x40191000)
        libICE.so.6            => /usr/X11R6/lib/libICE.so.6 (0x4019a000)
        libgtk-1.2.so.0        => /usr/lib/libgtk-1.2.so.0 (0x401b1000)
        libdl.so.2             => /lib/libdl.so.2 (0x402df000)
        libgdk-1.2.so.0        => /usr/lib/libgdk-1.2.so.0 (0x402e3000)
        libgmodule-1.2.so.0    => /usr/lib/libgmodule-1.2.so.0 (0x40319000)
        libXi.so.6             => /usr/X11R6/lib/libXi.so.6 (0x4031d000)
        libXext.so.6           => /usr/X11R6/lib/libXext.so.6 (0x40325000)
        libX11.so.6            => /usr/X11R6/lib/libX11.so.6 (0x40333000)
        libgnome.so.32         => /usr/lib/libgnome.so.32 (0x40411000)
        libgnomesupport.so.0   => /usr/lib/libgnomesupport.so.0 (0x40429000)
        libesd.so.0            => /usr/lib/libesd.so.0 (0x4042e000)
        libdb.so.2             => /usr/lib/libdb.so.2 (0x40436000)
        libglib-1.2.so.0       => /usr/lib/libglib-1.2.so.0 (0x40444000)
        libcrypt.so.1          => /lib/libcrypt.so.1 (0x40468000)
        libc.so.6              => /lib/i686/libc.so.6 (0x40495000)
        libz.so.1              => /usr/lib/libz.so.1 (0x405d1000)
        /lib/ld-linux.so.2     => /lib/ld-linux.so.2 (0x40000000)
```

The listing above shows all of the libraries that the **gaim** program is dynamically linked against.

Most programs that use shared libs, rely on the dynamic loader **ld-linux** to locate and load each of the shared libraries. Some programs, however, load the libraries manually with the **dlopen()** function call. For those applications, **ldd** will not show the libraries. In that case, **strace** can be used to trace the execution of the program, and you will see the **dlopen()** calls, with the name of the library listed in the arguments.

Once the list of libraries has been collected, the required libraries will need to be copied to the appropriate places in the `/opt/ltsp/i386` tree.

## 9.5. Launching local applications

In X Windows, programs typically run relative to where the window manager runs. That is, if the window manager is running on the server, displaying its output on the workstation, then any programs that are launched, will also run on the server, sending their output to the workstation.

The trick is having the server tell the workstation to launch the program. This is typically done with the **rsh** command.

Here's an example of how to run the **gaim** program on the workstation:

```
HOST=`echo $DISPLAY | awk -F: '{ print $1 }'`  
rsh ${HOST} /usr/bin/gaim -display ${DISPLAY}
```

The above example can be entered in an **xterm** window, or it can be put into a shell script, and be launched by an icon on the desktop.

Launching local Netscape is done in a similar way, but an additional environment variable needs to be set, before running the program.

```
HOST=`echo $DISPLAY | awk -F: '{ print $1 }'`  
rsh ${HOST} MOZILLA_HOME=/usr/local/netscape \  
    /usr/local/netscape/netscape -display ${DISPLAY}
```

---

# Chapter 10. Other sources of information

## 10.1. Online references

1. The LTSP home page

[www.LTSP.org](http://www.LTSP.org)

2. Diskless-Nodes HOW-TO document for Linux

[www.linuxdoc.org/HOWTO/Diskless-HOWTO.html](http://www.linuxdoc.org/HOWTO/Diskless-HOWTO.html)

3. Etherboot Home Page

[etherboot.sourceforge.net](http://etherboot.sourceforge.net)

4. The Rom-O-Matic site

[www.Rom-O-Matic.net](http://www.Rom-O-Matic.net)

5. XFree86-Video-Timings-HOWTO

[www.linuxdoc.org/HOWTO/XFree86-Video-Timings-HOWTO.html](http://www.linuxdoc.org/HOWTO/XFree86-Video-Timings-HOWTO.html)

6. The Linux NIS(YP)/NYS/NIS+ HOWTO

[www.linuxdoc.org/HOWTO/NIS-HOWTO.html](http://www.linuxdoc.org/HOWTO/NIS-HOWTO.html)

---

## 10.2. Print publications

- 1.

Managing NFS and NIS  
Hal Stern  
O'Reilly & Associates, Inc.  
1991  
ISBN 0-937175-75-7

- 2.

TCP/IP Illustrated, Volume 1  
W. Richard Stevens  
Addison-Wesley  
1994  
ISBN 0-201-63346-9

- 3.

X Window System Administrator's Guide  
Linda Mui and Eric Pearce  
O'Reilly & Associates, Inc.  
1993  
ISBN 0-937175-83-8  
(Volume 8 of the The Definitive Guides to the X Window System)

