Diskless Nodes HOW–TO document for Linux

Table of Contents

Table of Contents

Diskless Nodes HOW−TO document for Linux

Robert Nemkin buci@math.klte.hu , Al Dev (Alavoor Vasudevan) − Maintainer of this HOWTO alavoor@yahoo.com , Markus Gutschke markus+etherboot@gutschke.com , Ken Yap ken.yap@acm.org , Zurück zu Gero gero@gkminix.han.de

v6.0, 22 Jan 2000

This document describes how to set up a diskless Linux box. As technology is advancing rapidly, network−cards are becoming cheaper and much faster − 100 MBits ethernet is standard now and in about 1 to 2 years 1000 MBits i.e. 1GigBits ethernet cards will become a industry standard. With high−speed network cards, remote access will become as fast as the local disk access which will make diskless nodes a viable alternative to workstations in local LAN. Also diskless nodes eliminates the cost of software upgrades and system administration costs like backup, recovery which will be centralized on the server side. Diskless nodes also enable "sharing/optimization" of centralised server CPU, memory, hard−disk, tape and cdrom resources. Diskless nodes provides mobility for the users i.e., users can log on from any one of diskless nodes and are not tied to one workstation. Diskless Linux box completely eliminates the need for local floppy disk, cdrom drive, tape drive and hard−disk. Diskless nodes JUST has a network card, 8MB RAM, a low−end cpu and a very simple mother−board which does not have any interface sockets/slots for harddisks, modem, cdrom, floppy etc.. With Diskless linux nodes you can run programs on remote Linux 64 CPU SMP box or even on Linux super−computer! Diskless nodes lowers the "Total Cost of Ownership" of the computer system. This document is copy-righted by Robert Nemkin and other authors as listed above. Copyright policy is GPL. Thanks to Bela Kis bkis@cartan.math.klte.hu for translating this initial document v0.0.3 (which was a mini−howto) to English.

---

1. [Buying is cheaper than building!](#)

2. [Diskless Computer for Microsoft Windows 95/NT !!](#)

- [2.1 VMWare package](#)
- [2.2 VNC package from AT and T](#)

1. Buying is cheaper than building!

Sometimes, buying a diskless linux computer will be cheaper than building!! Checkout the following commercial sites, which are selling diskless linux network−cards and diskless computers. These companies do **mass production** of Linux Diskless computers selling millions of units and thereby reducing the cost per unit. Each and every fortune 1000 companies in USA will be replacing the MS Windows PCs with diskless computers in near future as diskless linux computers can run both Linux and MS Windows 95 programs (via VMWare BIOS software). VMWare is NOT a emulator but has BIOS which allows you to install Windows 98/NT as guest OS to linux. You can use the 'xhost' command and DISPLAY environment from diskless node to run Windows95/Linux programs. See 'man xhost' on linux. You can also use Virtual Network Computing (VNC) to run Windows95/NT programs on linux diskless nodes. Get VNC from http://www.uk.research.att.com/vnc

- Linux Systems Labs Inc., USA http://www.lsl.com Click on "Shop On−line" and than click on "HardWare" where all the Diskless computers will be listed. Phone 1−888−LINUX−88.

- Diskless Workstations Corporation, USA http://www.disklessworkstations.com

- Unique Systems of Holland Inc., Ohio, USA http://www.uniqsys.com

Even if you buy diskless linux computer, you may be very much interested in reading this entire document.

Next Previous ContentsNextPreviousContents

10. Netboot

Netboot was written by Zurück zu Gero. The main site is at http://www.han.de/~gero/netboot.html.

10.1 Introduction

The following list shows just a few examples of what Netboot can be used for:

- Printer spooler
- Terminal server
- X11 terminal
- Data logging system
- Network−Computer (NC)
- Some more ....

For the bootrom to find the kernel image it uses the BOOTP protocol as defined in RFC 951 and RFC 1533 to get the necessary boot information, and then loads the actual image using the TFTP protocol as defined in RFC 1350 .

The exact specifications for this netboot process can be found http://www.han.de/~gero/netboot/english/spec.html.

10.2 Mailing list

There exists a mailing list devoted to network booting. To subscribe simply send a mail with the line

subscribe netboot

in it's body to majordomo@baghira.han.de

The subject in the mail header doesn't matter. After subscribing to it, you can send messages into the list by writing a mail to netboot@baghira.han.de.

10.3 Netboot useful links

Netboot mailing list archive is at http://www.han.de/~gero/netboot/archive/maillist.html

- 3com drivers at http://support.3com.com/infodeli/tools/nic
- Accton drivers at here
- Artisoft
- CNET
- Compaq
- D−Link
- Microdyne
- Many NE2000 PCI cards are based on Realtek chipsets. Get drivers here
- Standard Microsystems Corp
- Surecom
- Thomas Conrad corp
- Winbond
- Xircom

- Webopaedia page on network cards
- Jargon's driver page with many drivers for older network cards.
- Etherboot This is a project similar to Netbot but based on the BSD bootrom code.
- How to make an X Window Terminal out of your old or outdated PC.
- List of jumper settings for various network cards. This page also contains many other good links.
- Freefire is the home page of the Freefire project, which lists many resources for network security issues.

11. Diskless Root NFS howto

See also the 'Diskless−Root−NFS−HOWTO' at
http://metalab.unc.edu/LDP/HOWTO/Diskless−Root−NFS−HOWTO.html

12. Appendix A − Install Instructions

```
                    I N S T A L L A T I O N
```

```
Overview of the installation process
====================================
```

Due to it's nature this package requires at least two computer systems. One
acts as a server, and at least one other will be setup as a diskless client.
Therefore this installation guide is divided into four sections:

```
        1.) Compilation and installation of utility programs on the server
        2.) Create a netbootable image of the target operating system
        3.) Setup of the server
        4.) Setup of the client including building the bootrom
```

The server has to support TCP/IP and certain protocols based on this network
standard. Most likely this will be a Unix-type server. Though it's probably
possible to also use servers running OS/2 or Windows-NT, for example, all
server related programs in this package can currently only be compiled on
a Unix-type host. This requirement is independed of the operating system
which is later booted on the diskless client. Therefore even if you want
to boot MS-DOS on your client(s) you need at least one Unix-type computer
for program compilation and generation of all boot files. Lateron when all
necessary files are built you can use any server you want.

This package contains two main parts:

```
        1.) The bootrom source and binaries. This part gets installed on
            the diskless client. All binaries except for utility programs
            are already precompiled. There are no further user changeable
            or adjustable options in the sources so you don't have to have
            access to the 16 bit development tools in order to use the boot-
            rom. You can just use the binaries provided.
            In order for the bootrom to access the network card in your
            diskless client you need a driver. Currently the bootrom only
            supports so called packet drivers, which are normally used on
            MS-DOS systems to interface a network stack with the hardware.
            With this package only the packet driver binaries are required,
            so you don't need to recompile anything here as well. You can
            find precompiled packet drivers for many popular network cards
            on any SimTel FTP mirror (it's called Crynwr packet driver col-
            lection), and for those of you without internet access some of
            those packet driver binaries are included with this package.
            Another good source for a packet driver for your network card
            might be it's manufacturer. At least the well known manufacturers
            (3Com and SMC for example) provide packet drivers for their
            complete product line. Those manufacturer provided packet drivers
            are usually faster and easier to install than those from the
            Crynwr collection, and can sometimes determine the hardware
            configuration at runtime, which the Crynwr drivers can't. However,
            there is a limitation in that you can only use packet drivers
            which are COM-type executables. EXE-type programs are not
            supported yet.

        2.) A set of programs to generate netbootable images on the server.
            These programs are called mknbi-<os>, where <os> identifies the
            operating system which is lateron running on the diskless client.
            Currently only Linux and MS-DOS are supported.
```

There is another requirement which should not leave unnoted. Although you
can build a bootrom with slightly limited functionality which is less than
16kB in size, the usual size for a bootrom will be between 16kB and 32kB.
Therefore when you go shopping for a network card you should try to get
one which is able to support 32kB EPROM's. This is standard on almost all
cards from major manufacturers, but most cheap NE2000 are known to allow

only a maximum of 16kB. Also note that some network cards from 3Com and SMC
allow you to select ROM sizes of 32kB and more with their configuration
programs, but can physically support only 16kB!

```
Compilation and installation of utility programs on the server
==============================================================
```

This package uses GNU's autoconf to configure the compilation process
of the utility programs. You shouldn't have any problems to compile
these programs on any Unix-type system.

        1.) Cd into the netboot directory and run ./configure. It's
            a configuration script generated by autoconf and checks
            for header files and system specific details. The mknbi
            utility programs contain some Intel assembler modules which
            lateron run on the diskless client. If you want to assemble
            these modules you need as86 and ld86, which you can get for
            free for Unix systems. However, there are preassembled files
            available so you actually don't need these two programs.
            configure checks for their existence and creates the Makefiles
            accordingly.
            For an explanation of the switches available to configure
            just run it with the --help option. Some additional switches
            are available:

                --disable-mknbi-linux
                --disable-mknbi-dos

            Choose these options if you don't want to create any of the
            corresponding mknbi utility programs. There is also another
            configure option:

                --enable-bootrom

            Use this option only if you want to recompile the bootrom
            itself. If you want to use the precompiled binaries, you don't
            need to specify this switch. See the file INSTALL.bootrom
            about how to recompile the bootrom.

        2.) Check that all generated Makefiles and the config.h are correct
            for your system.

        3.) Compile all programs with

                make clean
                make

            This will compile all programs without those which you disabled
            during the configuration stage. IMPORTANT NOTE: Some Makefiles
            use ifdef, which not every make program understands. If you
            get an error from make (usually in the form: "missing delimiter")
            then get and install GNU make on your system! Especially System V
            systems are known to have this deficiency.

        4.) If you want to permanently install the utility programs on
            your server you can run

10.3 Netboot useful links   7

```
        make install
```

        This will also install the corresponding man pages for later
        reference. However, it's perfectly ok to skip this step and
        run the mknbi program from their source directories. But please
        note that they are just called "mknbi" within their source
        directories. Therefore if you read further down to run mknbi-dos,
        you have to use "./mknbi-dos/mknbi" instead if you didn't install
        the programs using 'make install'.

Create a netbootable image of the target operating system
===

This step of the installation process depends on which operating you
want to boot on your diskless clients. Everything described in this
chapter does not depend on working on a Linux system. You can use any
UNIX type system to create the netbootable images.

Linux: With Linux you have far too many options to list them all in
        this text. Please refer to the mknbi-linux man page for all
        details. I will only describe the most common ways to setup a
        diskless Linux client here.
        First you have to decide where the Linux client is going to
        mount it's root filesystem from. This can either be a directory
        on an NFS server or a ram disk. Setup your Linux kernel accordingly.
        To use a root filesystem on an NFS server you should include TCP/IP
        network support into the kernel together with support for NFS file-
        systems. You cannot load this NFS support using a module as it has
        to be available at bootup. Additionally you also have to select
        NFSROOT support during kernel configuration. However, you don't
        need BOOTP or RARP support. Accordingly if you want to use ramdisk
        support the filesystem type you are going to use on the ramdisk has
        to be permanently compiled into the kernel. Also initrd has to be
        included in that case.

        1.) Configuring for NFS root filesystem.

        Next copy your Linux kernel into the current directory and run
        mknbi-linux:

                mknbi-linux -d rom -i rom -k zImage -o bootImage

        This supposes that your kernel image is called zImage, and gives
        you a netbootable image named bootImage.

        2.) Configuring for root filesystem on ramdisk

        If you want to use a ramdisk as a root device you have to create
        a ramdisk image first. Probably the easiest way to setup such an
        image is to use a floppy, though you can also use the loopback
        device if you are working on a Linux host. First format the floppy
        and make a filesystem on it. Next copy all programs and files onto
        it which you want to have on the root filesystem of the diskless
        client lateron. You should then test your root floppy. To do this
        copy your kernel onto another floppy with dd and set it's root device

to floppy using rdev:

```
dd if=zImage of=/dev/fd0
rdev /dev/fd0 /dev/fd0
```

Now boot your diskless client using this boot disk. After the kernel
started up, it will ask you to insert the root floppy and to press
enter. Your root floppy will be mounted.
If everything works as you intended, you can now create a netbootable
image. Re-insert the root floppy into your server system (or whereever
this netboot directory is located), and type:

```
dd if=/dev/fd0 of=ramImage
gzip -9 ramImage
mknbi-linux -d ram -i rom -r ramImage.gz -k zImage -o bootImage
```

Like above this will now give you a file bootImage with the netbootable
Linux kernel image in it.

MS-DOS: To boot DOS on your diskless client you have to have MS-DOS Version
        5.0 or higher. Windows-95 has an internal DOS called version 7.0, so
        it should be no problem to use it as well. Older MS-DOS versions
        will definitely not work. I haven't had a chance to test any other
        DOS like Novell-DOS or DR-DOS. Give them a try, and tell me.

        First you have to create a directory which contains all the files
        the client will see on it's boot drive (either A: or C:). This
        can either be the root directory on a DOS floppy or any directory
        on the system on which you installed mknbi-dos. In the first case
        it has to be a floppy which contains a bootable DOS system, i.e.
        which has been created with

```
format a: /s
```

on a DOS system. If the directory resides on a UNIX system, you
have to copy the two system files msdos.sys and io.sys, which are
part of MS-DOS, into it by yourself. To do this I recommend using
mread of the MTools, which are freely available for almost every
UNIX system.

After you created the directory or floppy which lateron becomes
the clients boot drive, you should copy all other necessary files
into it. This will probably include programs to setup a network
environment on the client. When editing text files for the client
please note that they usually have to be in MS-DOS format with
lines ending in Carriage-Return/Linefeed instead of just Linefeed
as it is common on UNIX systems. When you are finished setting up
the clients boot directory, first get a copy of the floppy disk
image, and then run mknbi-dos to create a netbootable image:

```
dd if=/dev/fd0 of=fdImage
mknbi-dos -r fdImage -o bootImage
```

This assumes that you inserted the boot floppy into the fd0 drive
of your UNIX system, and will create a file named bootImage. If you
used a UNIX directory, substitute fdImage with it's name. mknbi-dos
will automatically detect wether it is a directory, an ordinary
file or a block device.

By default mknbi-dos creates a netbootable image, which lateron
mounts the ram disk as the A: drive on your client. If you want

> to mount the ram disk as C: instead, you should include the '-c'
> switch to the call of mknbi-dos.
> The difference between mounting the ram disk as a floppy (A:) or
> hard disk (C:) is, that with the floppy option the ram disk can
> be removed lateron, maybe after a network redirector has been
> loaded, which makes the ram disk obsolete. This is not possible
> with a virtual hard disk drive. On the other hand side, when using
> the ram disk as C: you can specify a different ramdisk size with
> the '-s' option. Please refer to the man page for mknbi-dos for
> further information.

```
Setup of the server
===================
```

> Setup of the server depends on the kind of server you are using. There-
> fore all further explanations in this chapter can only serve as a general
> guide. You should consult your server's documentation as the final autho-
> rity.

> When the bootrom starts on the client it first tries to query a bootp
> server for information like IP numbers and the name of the boot image
> file. Such a bootp server program is usually called bootpd. Most sun
> servers use a program called bootparamd instead. Note that you _cannot_
> use bootparamd as a substitute for bootpd as both programs use different
> protocols. Install a publicly available bootpd instead on your sun.
> Next you should copy the bootImage file, which you have created in the
> previous step above, into a publicly accessible directory (called /boot
> for example). If you want to boot more than one diskless client you can
> use the same bootImage file for every client. However, if you configured
> for a ramdisk (with Linux or DOS) and the ramdisk image contains different
> files or information for every client, you will obviously also need a
> different bootImage file for each client.
> Then you need to setup a boot description file for bootpd, which is
> usually called /etc/bootptab. Consult your server's documentation for
> further information. However, the entries in this file will usually
> look something like this for every diskless client:

```
client1:hd=/boot:vm=auto:ip=192.109.225.66:\
        :ht=ethernet:ha=004001417173:\
        :bf=bootImage-client1:rp=/boot/client1/root
```

> 'hd' specifies the home directory and 'bf' is the name of the bootImage file,
> which you created in the previous step. Therefore the full pathname for
> the bootImage file for the diskless system called "client1" will be

```
        /boot/bootImage-client1
```

> with this sample entry. The 'ip' tag specifies the IP address of the client,
> 'ht' the type of the network the client is attached to, and 'ha' it's hard-
> ware address. The 'vm=auto' tag tells bootpd to use the same vendor encoding
> as the bootrom. If your diskless client is going to use it's root filesystem
> via NFS you should also specify the directory on the server which gets mounted
> lateron with the 'rp' tag. However, if your diskless client uses a ramdisk,
> you can omit 'rp'. When you choose to use the standard bootrom with ANSI
> display driver (see below for further information) you could also setup
> a menu for letting the user select different boot image files. See the

additional file INSTALL.menu about how to use this feature. But I recommend
to first use the standard way of setting up the bootptab file as described
above. You can always add a user menu lateron.
Of course you should also remember to get bootpd running on the server,
either on bootup from /etc/rc or some similar mechanism, or from inetd.
Again, see your server's documentation about how to do this.

The next step preformed by the bootrom after querying the bootp server is
to load in the boot image file specified by the 'hd' and 'bf' tags in
/etc/bootptab. To do this a protocol named tftp is used. Therefore you
will next have to setup a daemon process for this protocol on your server.
Such a daemon is usually called tftpd, and you should again remember to
get tftpd running, usually via inetd. Since the TFTP protocol is very
insecure access to the tftpd server is usually restricted, either within
tftpd itself, or with a TCP/IP wrapper like tcpd. tcpd for example uses
host access control tables which are stored in /etc/hosts.allow and
/etc/hosts.deny. See tftpd(8), tcpd(8) and hosts_access(5) as well as
your server's documentation for further information.

If you selected a ramdisk for the diskless client's root directory you are
now finished with the server setup. But if your client is going to use NFS
(either directly like with booting Linux, or by using programs included on
the ram disk) you should now setup everything which is necessary for moun-
ting an NFS directory on the server. This usually involves running several
programs: portmap, mountd, nfsd and optionally ugidd. portmap usually doesn't
require editing any configuration files. But for mountd and nfsd you need
to specify the permissions which allow the client to access the required
directories on the server. These permissions are usually set with a file
called /etc/exports. Typically it looks like this for our sample client:

#
Export directories for client1 (diskless workstation)
#
/boot/client1/root   client1(rw,link_absolute)
/boot/client1/usr   client1(rw,link_absolute)

If you use 'map-daemon' to map UID and GID numbers on the server you
should remember to also configure and run ugidd on the server. Please
consult your server's documentation for further information regarding
setup of NFS exports. You might also want to check out the portmap(8),
nfsd(8), mountd(8) and ugidd(8) man pages. Also remember that access
to any of these services might be restricted with tcpd on your server.

Another important step is to fill up the root directory for the disk-
less client. It has to contain all files necessary for the client to
startup and mount further directories via NFS (like a /usr filesystem
as specified in the /etc/exports example above). How to setup this
root directory is far beyond the scope of this documentation. Just one
hint: if your server is _not_ running Linux, you should be aware of
major/minor number assignments in the /boot/client1/root/dev directory.
For example, simply using mknod on an AIX server will eventually give
you wrong major/minor number when the directory is later exported to
a Linux diskless client. With some configurations AIX will add a certain
offset to all major numbers which makes them unusable for Linux. Refer
to your server's manuals for further information. You might also find
some useful hints in the file Documentation/nfsroot.txt in the Linux
source tree, if your diskless client is booting Linux.

10.3 Netboot useful links   11

```
Setup of the client including building the bootrom
=================================================
```

Until now you only had to work on the server (with the exception of maybe
booting your diskless client from a diskette to check the correctness of
the root filesystem). As the last step we can now go on and setup the
diskless client itself.

The first step is to configure the network card in the diskless client. For
this refer to the manual which came with the network card. Some cards require
setting of jumpers. Others have setup programs which have to be run. After
configuring the network interface write down all necessary hardware parameters
like I/O addresses, memory addresses, interrupt line number or DMA channel
numbers, as you might need this information lateron in the configuration
process.

Next change into the netboot directory on your UNIX system (where this
documentation file is in) and type

```
        make bootrom
```

This will compile all necessary utility programs and then run the
configuration program. It will first ask you which bootrom kernel you
want to use. The minimal kernel is necessary for network cards which
only allow up to 16 kB ROM size, and kernel86 can be used to boot on
16−bit systems (older than 386), for example for booting MS−DOS. Unless
you have any special requirements you should choose the standard kernel.
Then you have to specify the packet driver to use for your network card.
You can either choose one of the supplied drivers, or provide your own.
If you want to provide your own driver you have to give the full path
name of the packet driver binary on your server, and also specify all
necessary options to run it. Don't specify any options here which switch
the packet driver into windows mode or which allow it to work for disk-
less systems. Those options are for Novell network bootroms only, and
are not necessary for this bootrom.
If you use one of the drivers in the list shown, the configuration
program will ask you about all necessary hardware information to run
the packet driver which you selected. This usually includes the I/O
address of the network card, it's interrupt number and a DMA channel
number. Note that only that information is requested which is really
necessary. You should have your network card information handy when
entering this information. Some packet drivers are able to determine
hardware related information at runtime and therefore don't require
any further information.

If you did not select the minimal kernel, the configuration program
is next going to ask you wether you want to include some additional
drivers. First it lets you select the ANSI display driver. This will
allow you to draw nice menus on the screen with the standard bootrom
kernel. You can then select the packet driver debugging program. It's
an additional module to trace network problems and is usually not re-
quired. It shows you the first couple bytes of all packets (where
the UDP/IP headers are encoded) going through the packet driver
during boot time of the diskless client. Only select this debugging
module if you run into problems during the initial network boot process
of the bootrom _and_ you know how to decode the UDP/IP header infor-
mation. The configuration program will also ask you about any additional
modules you want to install into the bootrom. These modules have to
be standard DOS COM−type programs, and can, for example, preset
the network card to a special state before the packet driver starts,

or setup a serial line to support booting over a PPP or SLIP connec-
tion (the Crynwr packet driver collection also contains a SLIP packet
driver which is not provided in this package). However note that the
total size of the resulting bootrom image can't be larger than 64kB.

After you answered all questions the configuration program is creating
the bootrom according to your specifications. It first combines the
bootrom kernel with all selected modules, then compresses the resulting
file and adds the bootrom startup code. When the configuration program
has finished you will find two new files in the current directory:

        image.flo – this file can be written onto a floppy using dd
        image.rom – image to be burned into an EPROM

You should now copy image.flo onto a floppy using

        dd if=image.flo of=/dev/fd0

and then boot your diskless client using this floppy. If you have setup
everything (including your network card) you will see the bootrom code
starting, querying the bootp server and loading the boot image file. When
everything works as required you can then go on and burn the file image.rom
into an EPROM. Please consult the manual of your EPROM burner how to do
this. It usually requires converting the image file into a special format
(Intel or Motorola hex format for example). Insert the EPROM into the
socket on your network card and turn on the diskless system. You should
now see the bootrom coming up.
Another way of getting the bootrom code into your client is using the
Flash-EPROM card (called FlashCard), for which you can find a schematic
and PCB layout in this package. You can use image.rom directly to burn
it into FlashCard – there is no hex conversion necessary. About how to
use and program the FlashCard see the documentation in the FlashCard
directory.

In case you want to create new bootroms without always having the sources
around, you can now install the binaries created during the configuration
step with the command

        make bootrom_install

This will copy all necessary binaries for creating new bootroms into the
directory $prefix/lib/netboot where $prefix is either /usr/local or the
prefix you specified with running GNU configure. The typical path would
be /usr/local/lib/netboot. It will also install the makerom script into
$prefix/bin, so you just have to type makerom to create a new bootrom.

Appendix: Recompiling the bootrom
========

If you want to recompile the bootrom for some reason, checkout the file
INSTALL.bootrom for further information. However, you don't need to re-
compile the bootrom in order to just use it!

13. Appendix B – Troubleshoot Problems

```
                 T R O U B L E S H O O T I N G

If you run into any problem during installation or when using this
package, please first read the following text and all other relevant
documentation. Especially you should consult your server's documen-
tation if you run into problems setting up your server. Also refer
to your network card's user manual or the documentation for the
operating systems of the diskless clients accordingly. However, if
you still can't solve the problem on your own, you can send me an
email to

            gero@gkminix.han.de

Users able to speak German can send me the mail in german. Otherwise
please write in english. I already received some emails in so poor
english that I haven't been able to even understand the problem. I
can't help you in that case. And please excuse me that I can't answer
questions sent to me by standard mail or telephone calls. I just don't
have the time for dealing with that.
If you decided to send me an email please describe your problem as
exactly as possible. It usually helps to send me relevant portions
of configuration files (I have to pay for my internet access by myself
so please keep quotings as short as possible). Especially with problems
with the bootrom it usually helps to _exactly_ write down the screen
output, not only but including any error messages. Also state as exact
as possible how you created the problem so that I can try to simulate
it on my own hardware.
Additionally please note that I can't help you with every problem with
your server, as there are so many different systems on the market. The
same is true for problems with network cards. I just don't have the
financial capabilities to buy any card on the market for testing. Per-
sonally I'm using NE2000 and WD8013 cards, so I can probably help you
with those.
If you find a problem which looks like a bug in the code I really
appreciate a short notice from you. And if you have a fix for the bug
I would even more appreciate your message.
Besides contacting me directly there also exists a mailing list related
to network booting which you can subscribe to. Write a mail with the
message 'subscribe netboot' in it's body to majordomo@baghira.han.de
(the subject of the mail doesn't matter). The readers of the mailing
list should also be able to help you with any problem you might have
while setting up a diskless client. And besides that I'm also going
to announce any new version of this netboot package to the mailing
list.



    Problem: My operating system OS/XY is not supported by netboot
```

I would gladly provide support for every operating system on the
market, but I don't have the resources for doing this. However,
if you want a particular operating system to be supported, you
should get in contact with me. In any case you will have to provide
me with a valid and licensed copy of that operating system. You are
also invited to write your own boot loader, and send it to me for
inclusion into netboot under the terms of the GNU GPL.

Problem: While trying to build a bootrom I get a compiler error

The installation scripts require to compile a couple of utility
programs which are only required during building the bootrom.
They should compile on any Unix-type system, so if you get an
error please report it to me, even when you are able to fix it
yourself, so that I can include a patch for future releases.

Problem: I get a an error from make saying something like "missing delimiter"

Some of the Makefiles use ifdef's, which older make programs don't
understand. Even some more "modern" systems like SCO Open-Server 5
have this problem. In that case you will have to get and install GNU
make on your system (which is the better choice anyway).

Problem: The bootrom doesn't startup at all

Either you have a floppy in your diskette drive or you have
a hard disk installed with a partition marked as active, and the
bootrom has been built so that it lets the BIOS look for active
partitions first. Both conditions let the system boot from the
bootable media instead of using the bootrom. Just remove the
floppy or use fdisk to mark all partitions as unbootable (e.g.
inactive). Alternatively you can also build the bootrom so that
it does not allow the BIOS to look for bootable partitions. The
program which actually creates the bootrom ('makerom', it gets
called when you run 'make bootrom') will ask you about this right
after selecting the bootrom kernel image.

Problem: The bootrom behaves strange during startup, and may even hangup
        the whole system

If you compiled the mknbi programs on a system with big endian
byte order (like Motorola or PPC systems) this might indicate
that the configuration program couldn't find the correct byte
order. It might also be that there is a bug in the byte ordering
code. Some systems like SPARCs also do not allow data accesses at
misaligned addresses. 'configure' should usually find out about
these conditions. In any case, if 'configure' is not able to pro-
perly detect what kind of system you are using, edit the file
config.h by hand and try it again. Please report this condition,
and also note which system you used for installation.

Problem: The packet driver is not able to start properly

First check what error message the packet driver prints. Usually
this problem is a result of an incorrect setup of the network
card, so check that it uses an I/O address, interrupt line and DMA
channel (if applicable) of it's own, and that the packet driver
uses the correct values. Another common problem with ethernet
cards which use shared memory (like WD80?3 cards) is an overlap-
ping of this shared memory with the rom area used by the bootrom.
Select a different shared memory address in that case. If that's
ok you should next check that you configured the packet driver
correctly with the bootrom configuration program. Usually the
packet driver prints out what it expects the hardware to look
like so you can use this information to check up your setup.

Problem: The bootrom tells me that there is not enough memory but I have
        xx megabytes installed

This problem is a result of the fact that the BIOS starts the
bootrom in the processor's real mode. The bootrom is therefore
only able to access the lower 1 megabyte of memory, regardless
of how much you installed. And 384kB of this is reserved for
ROM's and the video memory, so there is only 640kB left. Unfor-
tunately some systems even reserve memory from these lower 640kB
for internal BIOS data. This is called extended BIOS data area,
and known to be used on most PS/2 systems. But also some other
BIOSes use such an extended BIOS data area, which is usually
selectable in the system's setup. Therefore you should try to
deselect such a feature. If that's not possible you are out
of luck − sorry.

Problem: The bootrom doesn't receive a bootp answer and just hangs printing
        dots

First you should check if bootpd runs on your server or is started
properly from inetd. Then check that the server's /etc/bootptab is
setup correctly. Especially the hardware address and the client's
IP address and name have to be correct.
Most bootp servers have the ability to write debugging information
into a log file. Use that feature to verify that your server really
receives bootp requests from the client's bootrom and sends out a
valid answer. Also check for error messages in the log file. Even
if your bootpd doesn't write into a seperate log file it might use
syslog on your system, so find the log file name from your syslogd
configuration file and check for errors.
If you are able to use a network tracing program like tcpdump you
can check if the bootrom sends out correct requests and that the
server is answering correctly. In that case it is more likely to
be a problem in the bootrom, so you should create a new bootrom
image with the packet driver debugging module included. You should
then see the bootrom's request packets going out, and the server's
answers coming in. If there are no packets coming in although you
verified that the server is sending out correct replies there might
be a problem with your network card. Did you set it up correctly,
is a cable connected (no kidding, those things really happen)?
If everything fails try to boot the diskless client with the
intended operating system and try to access the network card
using that operating system's tools.
If the server is not sending out answer packets, but the bootpd

13. Appendix B − Troubleshoot Problems   16

logfiles indicates correct answers, it might be a problem with
the arp setup on your server. Normally arp shouldn't be a concern
for you. However, some older versions of bootpd for Linux had
problems here, which could be solved by setting the kernel arp
table manually.

Problem: The bootrom did get a bootp answer but is not able to load the
         bootimage file

         This is likely to be a problem with the tftpd setup on the server.
         Does tftpd run when you startup the bootrom code? If not check
         that inetd is configured correctly. Also there might be a TCP/IP
         wrapper running on your server which might prohibit access to
         the tftp service (which is known to be very insecure and therefore
         a candidate for getting started by an internet security wrapper
         like tcpd). Check any access configuration files for tcpd.
         Furthermore tftpd has to be able to access the bootimage file. It
         usually runs as a user with very low priviliges because of security
         reasons and might not be allowed to read the bootimage file, so
         you should check and set the bootimage file's permissions correctly.

Problem: The boot image loader reports an error

         Congratulations! You just discovered a bug in the boot loader.
         Please report it to me.

Problem: When I'm using the bootrom menu to load a Unix system off the local
         hard disk, it reports some weird error messages to me (especially,
         SCO Unix says that it's not able to open boot device). However,
         booting without the bootrom works without a problem.

         Some operating systems, especially Unix like systems, read the
         partition table after booting and try to find their own boot par-
         tition. When using the bootrom, it's not necessary to mark the
         Unix partition as bootable, so the Unix startup loader fails.
         To solve this problem, mark the Unix partition active with some
         fdisk program. To avoid that it starts running instead of the
         bootrom, create the bootrom so that it does not allow the BIOS
         to search for boot partitions on the installed hard disks (the
         'makerom' program, which gets run when you do a 'make bootrom',
         will ask you about this right after selecting a kernel image).

Problem: I'm loading Linux onto my diskless client and the kernel tells
         me to insert a root floppy and press enter

         First you should check that you built your kernel correctly. It
         should have support for the root filesystem built in. If you want
         to use an NFS mounted directory as root the kernel should have
         TCP/IP support installed. Also it has to have a driver for your
         network card built in, and NFS and NFSROOT have to be both speci-
         fied. When using a ramdisk it's support has to be compiled in
         as well as support for the filesystem with which you formatted
         the ramdisk image. Please note that the loaded kernel is not
         able to use modules at bootup time (only _after_ the root file-

13. Appendix B – Troubleshoot Problems   17

system has been mounted, but not before), so everything has to
be compiled in.

If the kernel is not able mount it's root via NFS, this might
have many different reasons. It requires all addresses in the
/etc/bootptab file to be correct, and the access rights on the
server have to be set correctly – not only in /etc/exports but
also the permissions for the directory to get mounted. If that's
correct check that a portmapper is running on the server, and
that it registered the mountd and nfsd services correctly. You
can usually do this by running the command

                rpcinfo –p

Note that services are only listed here if their associated server
process is really running. The rpcinfo output should then look
something like this:

```
        program vers proto    port
         100000   2    tcp     111  portmapper
         100000   2    udp     111  portmapper
         100003   2    udp    2049  nfs
         100003   2    tcp    2049  nfs
         100005   1    udp     663  mountd
         100005   1    tcp     665  mountd
```

However, the port numbers might be different.

When the kernel starts mounting the NFS root directory it prints
out the name of that directory on the server. It should be the
same as the one configured in /etc/bootptab. Check that it's
correct. If not you can try to use the –d option with mknbi-linux
to specify the name explicitly.

If the kernel gets an error from the server's nfsd, it prints
a number which is defined according to the NFS protocol. The
most commonly occurring numbers are:

```
         1  –  permission denied to access directory
         2  –  directory doesn't exist
         5  –  I/O error on server filesystem
        13  –  nfsd is unable to access directory
        20  –  path name is not a directory
        63  –  path name is too long
```

Note that some nfsd and mountd programs only read /etc/exports
on startup. If you changed this file afterwards, you will have
to restart both daemons. Additionally, with nfsd versions for
Linux earlier than 2.1 you will have problems with special files
like UNIX domain sockets or block/character special files on
your NFS partitions. You should therefore use the latest avai-
lable versions.

Problem: The Linux kernel mounts it's root correctly but doesn't give me
          a login prompt.

1.)   This might be the result of an incorrect setup of the root file-
          system (see No. 2 below). However, it's also possible that your
          server reported the wrong major/minor numbers for the console device
          even though you specified them correctly in the NFS mounted root

13. Appendix B – Troubleshoot Problems   18

directory. I know of this problem with AIX and HP−UX servers,
but there might exist others as well which don't transfer special
devices via NFS as Linux requires it. One solution to solve this
problem is to boot the diskless client with a ramdisk image as
it's root, and then mount the should-be-root directory on the
server using NFS. Then you can create the special files in the
dev directory using Linux's mknod program, and use the NFS root
mounting bootimage again.
Another way is to try to find out, how the server operating system
encodes major/minor numbers on it's own filesystem. For example,
HP−UX uses a 32 bit device number, with the 8 highest bits being
the major number, and the lower 24 bits being the minor device
number:

        major << 24 | minor   ==>   aaaaaaaabbbbbbbbbbbbbbbbbbbbbbbb

In this representation (a) means a bit of the major number, and
(b) means a bit of the minor number. Linux uses the following
scheme instead:

        major << 8 | minor   ==>   0000000000000000aaaaaaaabbbbbbbb

The NFS protocol now transfers these 32 bits just as they are,
without any further interpretation regarding major/minor numbers.
That means, that all relevant bits in the Linux representation
fit into the minor number on HP−UX. Therefore, if you create a
device on the HP−UX server, you have to alway give it a major
number of zero and compute the minor number the way mentioned
above for Linux. For example, to let Linux see a device 5/2 in
it's NFS-mounted /dev directory, you can compute the minor device
number on HP−UX as

        5 << 8 | 2   ==>   1282

So the device to create on the HP−UX server is 0/1282. This will
let Linux see 5/2 after the filesystem is mounted with NFS.

2.)   Another reason for this problem might be that the init process
        doesn't get started at all. This can be a result of incorrect
        shared libraries, which the client might see but without a proper
        ld.so.cache file. Or the shared libraries are not reachable by
        the client at all. Bruce Janson and Markus Gutschke collected a
        good list of possibilities, which you should check out:

            − you do not have a private copy of the /, /etc, /var, ...
              directories

            − your /dev directory is missing entries for /dev/zero and/or
              /dev/null or is sharing device entries from a server that uses
              different major and minor numbers (i.e. a server that is not
              running Linux − see above).

            − your /lib directory is missing libraries (most notably libc*
              and/or libm*) or does not have the loader files ld*.so*

            − you neglected to run ldconfig to update /etc/ldconfig.cache
              or you do not have a configuration file for ldconfig.

            − your /etc/inittab and/or /etc/rc.d/* files have not been
              customized for the clients.

            − your kernel is missing some crucial compile-time feature

13. Appendix B − Troubleshoot Problems   19

```
                          (such as NFS filesystem support, booting from the net, trans-
                          name (optional), ELF file support, networking support, driver
                          for your ethernet card).

                     - missing init executable (in one of the directories
                       known by the kernel: /etc, /sbin, ?)

                     - missing /etc/inittab

                     - missing /dev/tty?

                     - missing /bin/sh

                     - system programs that insist on creating/writing to files
                       outside of /var (mount and /etc/mtab* is the canonical
                       example)



        Problem: Can't compile the bootrom

                  Please get in touch with me if you encounter any problems
                  while recompiling the bootrom.
```

14. Appendix C – RFC 951

This section is for academic interest only – for universities or research institutes.

```
        Network Working Group                   Bill Croft (Stanford University)
        Request for Comments: 951               John Gilmore (Sun Microsystems)
                                                             September 1985

                          BOOTSTRAP PROTOCOL (BOOTP)


        1. Status of this Memo

           This RFC suggests a proposed protocol for the ARPA-Internet
           community, and requests discussion and suggestions for improvements.
           Distribution of this memo is unlimited.

        2. Overview

           This RFC describes an IP/UDP bootstrap protocol (BOOTP) which allows
           a diskless client machine to discover its own IP address, the address
           of a server host, and the name of a file to be loaded into memory and
           executed.  The bootstrap operation can be thought of as consisting of
           TWO PHASES.  This RFC describes the first phase, which could be
           labeled 'address determination and bootfile selection'.  After this
           address and filename information is obtained, control passes to the
           second phase of the bootstrap where a file transfer occurs.  The file
           transfer will typically use the TFTP protocol [9], since it is
```

intended that both phases reside in PROM on the client. However
BOOTP could also work with other protocols such as SFTP [3] or
FTP [6].

We suggest that the client's PROM software provide a way to do a
complete bootstrap without 'user' interaction. This is the type of
boot that would occur during an unattended power-up. A mechanism
should be provided for the user to manually supply the necessary
address and filename information to bypass the BOOTP protocol and
enter the file transfer phase directly. If non-volatile storage is
available, we suggest keeping default settings there and bypassing
the BOOTP protocol unless these settings cause the file transfer
phase to fail. If the cached information fails, the bootstrap should
fall back to phase 1 and use BOOTP.

Here is a brief outline of the protocol:

   1. A single packet exchange is performed. Timeouts are used to
   retransmit until a reply is received. The same packet field
   layout is used in both directions. Fixed length fields of maximum
   reasonable length are used to simplify structure definition and
   parsing.

   2. An 'opcode' field exists with two values. The client
   broadcasts a 'bootrequest' packet. The server then answers with a
   'bootreply' packet. The bootrequest contains the client's
   hardware address and its IP address, if known.

Croft & Gilmore   [Page 1]

RFC 951   September 1985
Bootstrap Protocol

   3. The request can optionally contain the name of the server the
   client wishes to respond. This is so the client can force the
   boot to occur from a specific host (e.g. if multiple versions of
   the same bootfile exist or if the server is in a far distant
   net/domain). The client does not have to deal with name / domain
   services; instead this function is pushed off to the BOOTP server.

   4. The request can optionally contain the 'generic' filename to be
   booted. For example 'unix' or 'ethertip'. When the server sends
   the bootreply, it replaces this field with the fully qualified
   path name of the appropriate boot file. In determining this name,
   the server may consult his own database correlating the client's
   address and filename request, with a particular boot file
   customized for that client. If the bootrequest filename is a null
   string, then the server returns a filename field indicating the
   'default' file to be loaded for that client.

   5. In the case of clients who do not know their IP addresses, the
   server must also have a database relating hardware address to IP
   address. This client IP address is then placed into a field in
   the bootreply.

   6. Certain network topologies (such as Stanford's) may be such
   that a given physical cable does not have a TFTP server directly
   attached to it (e.g. all the gateways and hosts on a certain cable
   may be diskless). With the cooperation of neighboring gateways,

14. Appendix C – RFC 951   21

BOOTP can allow clients to boot off of servers several hops away, through these gateways. See the section 'Booting Through Gateways' below. This part of the protocol requires no special action on the part of the client. Implementation is optional and requires a small amount of additional code in gateways and servers.

3. Packet Format

All numbers shown are decimal, unless indicated otherwise. The BOOTP packet is enclosed in a standard IP [8] UDP [7] datagram. For simplicity it is assumed that the BOOTP packet is never fragmented. Any numeric fields shown are packed in 'standard network byte order', i.e. high order bits are sent first.

In the IP header of a bootrequest, the client fills in its own IP source address if known, otherwise zero. When the server address is unknown, the IP destination address will be the 'broadcast address' 255.255.255.255. This address means 'broadcast on the local cable, (I don't know my net number)' [4].

Croft & Gilmore   [Page 2]

RFC 951   September 1985
Bootstrap Protocol

The UDP header contains source and destination port numbers. The BOOTP protocol uses two reserved port numbers, 'BOOTP client' (68) and 'BOOTP server' (67). The client sends requests using 'BOOTP server' as the destination port; this is usually a broadcast. The server sends replies using 'BOOTP client' as the destination port; depending on the kernel or driver facilities in the server, this may or may not be a broadcast (this is explained further in the section titled 'Chicken/Egg issues' below). The reason TWO reserved ports are used, is to avoid 'waking up' and scheduling the BOOTP server daemons, when a bootreply must be broadcast to a client. Since the server and other hosts won't be listening on the 'BOOTP client' port, any such incoming broadcasts will be filtered out at the kernel level. We could not simply allow the client to pick a 'random' port number for the UDP source port field; since the server reply may be broadcast, a randomly chosen port number could confuse other hosts that happened to be listening on that port.

The UDP length field is set to the length of the UDP plus BOOTP portions of the packet. The UDP checksum field can be set to zero by the client (or server) if desired, to avoid this extra overhead in a PROM implementation. In the 'Packet Processing' section below the phrase '[UDP checksum.]' is used whenever the checksum might be verified/computed.

```
     FIELD   BYTES   DESCRIPTION
     -----   -----   -----------

       op      1       packet op code / message type.
                       1 = BOOTREQUEST, 2 = BOOTREPLY

       htype   1       hardware address type,
                       see ARP section in "Assigned Numbers" RFC.
```

```
                        '1' = 10mb ethernet


        hlen    1       hardware address length
                        (eg '6' for 10mb ethernet).


        hops    1       client sets to zero,
                        optionally used by gateways
                        in cross-gateway booting.


        xid     4       transaction ID, a random number,
                        used to match this boot request with the
                        responses it generates.


        secs    2       filled in by client, seconds elapsed since
                        client started trying to boot.
```

```
Croft & Gilmore                                             [Page 3]
```

```
RFC 951                                                September 1985
Bootstrap Protocol


        --      2       unused

        ciaddr  4       client IP address;
                        filled in by client in bootrequest if known.


        yiaddr  4       'your' (client) IP address;
                        filled by server if client doesn't
                        know its own address (ciaddr was 0).


        siaddr  4       server IP address;
                        returned in bootreply by server.


        giaddr  4       gateway IP address,
                        used in optional cross-gateway booting.


        chaddr  16      client hardware address,
                        filled in by client.


        sname   64      optional server host name,
                        null terminated string.


        file    128     boot file name, null terminated string;
                        'generic' name or null in bootrequest,
                        fully qualified directory-path
                        name in bootreply.


        vend    64      optional vendor-specific area,
                        e.g. could be hardware type/serial on request,
                        or 'capability' / remote file system handle
                        on reply.  This info may be set aside for use
                        by a third phase bootstrap or kernel.

    4. Chicken / Egg Issues

       How can the server send an IP datagram to the client, if the client
       doesnt know its own IP address (yet)?  Whenever a bootreply is being
       sent, the transmitting machine performs the following operations:
```

1. If the client knows its own IP address ('ciaddr' field is
nonzero), then the IP can be sent 'as normal', since the client
will respond to ARPs [5].

2. If the client does not yet know its IP address (ciaddr zero),
then the client cannot respond to ARPs sent by the transmitter of
the bootreply. There are two options:

a. If the transmitter has the necessary kernel or driver hooks

Croft & Gilmore   [Page 4]

RFC 951   September 1985
Bootstrap Protocol

to 'manually' construct an ARP address cache entry, then it can
fill in an entry using the 'chaddr' and 'yiaddr' fields. Of
course, this entry should have a timeout on it, just like any
other entry made by the normal ARP code itself. The
transmitter of the bootreply can then simply send the bootreply
to the client's IP address. UNIX (4.2 BSD) has this
capability.

b. If the transmitter lacks these kernel hooks, it can simply
send the bootreply to the IP broadcast address on the
appropriate interface. This is only one additional broadcast
over the previous case.

5. Client Use of ARP

The client PROM must contain a simple implementation of ARP, e.g. the
address cache could be just one entry in size. This will allow a
second-phase-only boot (TFTP) to be performed when the client knows
the IP addresses and bootfile name.

Any time the client is expecting to receive a TFTP or BOOTP reply, it
should be prepared to answer an ARP request for its own IP to
hardware address mapping (if known).

Since the bootreply will contain (in the hardware encapsulation) the
hardware source address of the server/gateway, the client MAY be able
to avoid sending an ARP request for the server/gateway IP address to
be used in the following TFTP phase. However this should be treated
only as a special case, since it is desirable to still allow a
second-phase-only boot as described above.

6. Comparison to RARP

An earlier protocol, Reverse Address Resolution Protocol (RARP) [1]
was proposed to allow a client to determine its IP address, given
that it knew its hardware address. However RARP had the disadvantage
that it was a hardware link level protocol (not IP/UDP based). This
means that RARP could only be implemented on hosts containing special
kernel or driver modifications to access these 'raw' packets. Since
there are many network kernels existent now, with each source
maintained by different organizations, a boot protocol that does not
require kernel modifications is a decided advantage.

BOOTP provides this hardware to IP address lookup function, in

14. Appendix C – RFC 951   24

addition to the other useful features described in the sections above.

Croft & Gilmore   [Page 5]

RFC 951   September 1985
Bootstrap Protocol

7. Packet Processing

   7.1. Client Transmission

      Before setting up the packet for the first time, it is a good idea
      to clear the entire packet buffer to all zeros; this will place
      all fields in their default state. The client then creates a
      packet with the following fields.

      The IP destination address is set to 255.255.255.255. (the
      broadcast address) or to the server's IP address (if known). The
      IP source address and 'ciaddr' are set to the client's IP address
      if known, else 0. The UDP header is set with the proper length;
      source port = 'BOOTP client' port destination port = 'BOOTP
      server' port.

      'op' is set to '1', BOOTREQUEST. 'htype' is set to the hardware
      address type as assigned in the ARP section of the "Assigned
      Numbers" RFC. 'hlen' is set to the length of the hardware address,
      e.g. '6' for 10mb ethernet.

      'xid' is set to a 'random' transaction id. 'secs' is set to the
      number of seconds that have elapsed since the client has started
      booting. This will let the servers know how long a client has
      been trying. As the number gets larger, certain servers may feel
      more 'sympathetic' towards a client they don't normally service.
      If a client lacks a suitable clock, it could construct a rough
      estimate using a loop timer. Or it could choose to simply send
      this field as always a fixed value, say 100 seconds.

      If the client knows its IP address, 'ciaddr' (and the IP source
      address) are set to this value. 'chaddr' is filled in with the
      client's hardware address.

      If the client wishes to restrict booting to a particular server
      name, it may place a null-terminated string in 'sname'. The name
      used should be any of the allowable names or nicknames of the
      desired host.

      The client has several options for filling the 'file' name field.
      If left null, the meaning is 'I want to boot the default file for
      my machine'. A null file name can also mean 'I am only interested
      in finding out client/server/gateway IP addresses, I dont care
      about file names'.

      The field can also be a 'generic' name such as 'unix' or

Croft & Gilmore   [Page 6]

14. Appendix C – RFC 951   25

RFC 951   September 1985
Bootstrap Protocol

        'gateway'; this means 'boot the named program configured for my
        machine'. Finally the field can be a fully directory qualified
        path name.

        The 'vend' field can be filled in by the client with
        vendor-specific strings or structures. For example the machine
        hardware type or serial number may be placed here. However the
        operation of the BOOTP server should not DEPEND on this
        information existing.

        If the 'vend' field is used, it is recommended that a 4 byte
        'magic number' be the first item within 'vend'. This lets a
        server determine what kind of information it is seeing in this
        field. Numbers can be assigned by the usual 'magic number'
        process --you pick one and it's magic. A different magic number
        could be used for bootreply's than bootrequest's to allow the
        client to take special action with the reply information.

        [UDP checksum.]

    7.2. Client Retransmission Strategy

       If no reply is received for a certain length of time, the client
       should retransmit the request. The time interval must be chosen
       carefully so as not to flood the network. Consider the case of a
       cable containing 100 machines that are just coming up after a
       power failure. Simply retransmitting the request every four
       seconds will inundate the net.

       As a possible strategy, you might consider backing off
       exponentially, similar to the way ethernet backs off on a
       collision. So for example if the first packet is at time 0:00,
       the second would be at :04, then :08, then :16, then :32, then
       :64. You should also randomize each time; this would be done
       similar to the ethernet specification by starting with a mask and
       'and'ing that with with a random number to get the first backoff.
       On each succeeding backoff, the mask is increased in length by one
       bit. This doubles the average delay on each backoff.

       After the 'average' backoff reaches about 60 seconds, it should be
       increased no further, but still randomized.

       Before each retransmission, the client should update the 'secs'
       field. [UDP checksum.]

Croft & Gilmore   [Page 7]

RFC 951   September 1985
Bootstrap Protocol

7.3. Server Receives BOOTREQUEST

[UDP checksum.] If the UDP destination port does not match the
'BOOTP server' port, discard the packet.

If the server name field (sname) is null (no particular server
specified), or sname is specified and matches our name or
nickname, then continue with packet processing.

If the sname field is specified, but does not match 'us', then
there are several options:

   1. You may choose to simply discard this packet.

   2. If a name lookup on sname shows it to be on this same cable,
   discard the packet.

   3. If sname is on a different net, you may choose to forward
   the packet to that address. If so, check the 'giaddr' (gateway
   address) field. If 'giaddr' is zero, fill it in with my
   address or the address of a gateway that can be used to get to
   that net. Then forward the packet.

If the client IP address (ciaddr) is zero, then the client does
not know its own IP address. Attempt to lookup the client
hardware address (chaddr, hlen, htype) in our database. If no
match is found, discard the packet. Otherwise we now have an IP
address for this client; fill it into the 'yiaddr' (your IP
address) field.

We now check the boot file name field (file). The field will be
null if the client is not interested in filenames, or wants the
default bootfile. If the field is non-null, it is used as a
lookup key in a database, along with the client's IP address. If
there is a default file or generic file (possibly indexed by the
client address) or a fully-specified path name that matches, then
replace the 'file' field with the fully-specified path name of the
selected boot file. If the field is non-null and no match was
found, then the client is asking for a file we dont have; discard
the packet, perhaps some other BOOTP server will have it.

The 'vend' vendor-specific data field should now be checked and if
a recognized type of data is provided, client-specific actions
should be taken, and a response placed in the 'vend' data field of
the reply packet. For example, a workstation client could provide

Croft & Gilmore   [Page 8]

RFC 951   September 1985
Bootstrap Protocol

an authentication key and receive from the server a capability for
remote file access, or a set of configuration options, which can
be passed to the operating system that will shortly be booted in.

Place my (server) IP address in the 'siaddr' field. Set the 'op'
field to BOOTREPLY. The UDP destination port is set to 'BOOTP

client'. If the client address 'ciaddr' is nonzero, send the
packet there; else if the gateway address 'giaddr' is nonzero, set
the UDP destination port to 'BOOTP server' and send the packet to
'giaddr'; else the client is on one of our cables but it doesnt
know its own IP address yet --use a method described in the 'Egg'
section above to send it to the client. If 'Egg' is used and we
have multiple interfaces on this host, use the 'yiaddr' (your IP
address) field to figure out which net (cable/interface) to send
the packet to. [UDP checksum.]

7.4. Server/Gateway Receives BOOTREPLY

[UDP checksum.] If 'yiaddr' (your [the client's] IP address)
refers to one of our cables, use one of the 'Egg' methods above to
forward it to the client. Be sure to send it to the 'BOOTP
client' UDP destination port.

7.5. Client Reception

Don't forget to process ARP requests for my own IP address (if I
know it). [UDP checksum.] The client should discard incoming
packets that: are not IP/UDPs addressed to the boot port; are not
BOOTREPLYs; do not match my IP address (if I know it) or my
hardware address; do not match my transaction id. Otherwise we
have received a successful reply. 'yiaddr' will contain my IP
address, if I didnt know it before. 'file' is the name of the
file name to TFTP 'read request'. The server address is in
'siaddr'. If 'giaddr' (gateway address) is nonzero, then the
packets should be forwarded there first, in order to get to the
server.

8. Booting Through Gateways

This part of the protocol is optional and requires some additional
code in cooperating gateways and servers, but it allows cross-gateway
booting. This is mainly useful when gateways are diskless machines.
Gateways containing disks (e.g. a UNIX machine acting as a gateway),
might as well run their own BOOTP/TFTP servers.

Gateways listening to broadcast BOOTREQUESTs may decide to forward or
rebroadcast these requests 'when appropriate'. For example, the

Croft & Gilmore   [Page 9]

RFC 951   September 1985
Bootstrap Protocol

gateway could have, as part of his configuration tables, a list of
other networks or hosts to receive a copy of any broadcast
BOOTREQUESTs. Even though a 'hops' field exists, it is a poor idea
to simply globally rebroadcast the requests, since broadcast loops
will almost certainly occur.

The forwarding could begin immediately, or wait until the 'secs'
(seconds client has been trying) field passes a certain threshold.

If a gateway does decide to forward the request, it should look at
the 'giaddr' (gateway IP address) field. If zero, it should plug its
own IP address (on the receiving cable) into this field. It may also

use the 'hops' field to optionally control how far the packet is
reforwarded. Hops should be incremented on each forwarding. For
example, if hops passes '3', the packet should probably be discarded.
[UDP checksum.]

Here we have recommended placing this special forwarding function in
the gateways. But that does not have to be the case. As long as
some 'BOOTP forwarding agent' exists on the net with the booting
client, the agent can do the forwarding when appropriate. Thus this
service may or may not be co-located with the gateway.

In the case of a forwarding agent not located in the gateway, the
agent could save himself some work by plugging the broadcast address
of the interface receiving the bootrequest into the 'giaddr' field.
Thus the reply would get forwarded using normal gateways, not
involving the forwarding agent. Of course the disadvantage here is
that you lose the ability to use the 'Egg' non-broadcast method of
sending the reply, causing extra overhead for every host on the
client cable.

9. Sample BOOTP Server Database

As a suggestion, we show a sample text file database that the BOOTP
server program might use. The database has two sections, delimited
by a line containing an percent in column 1. The first section
contains a 'default directory' and mappings from generic names to
directory/pathnames. The first generic name in this section is the
'default file' you get when the bootrequest contains a null 'file'
string.

The second section maps hardware addresstype/address into an
ipaddress. Optionally you can also overide the default generic name
by supplying a ipaddress specific genericname. A 'suffix' item is
also an option; if supplied, any generic names specified by the
client will be accessed by first appending 'suffix' to the 'pathname'

Croft & Gilmore   [Page 10]

RFC 951   September 1985
Bootstrap Protocol

appropriate to that generic name. If that file is not found, then
the plain 'pathname' will be tried. This 'suffix' option allows a
whole set of custom generics to be setup without a lot of effort.
Below is shown the general format; fields are delimited by one or
more spaces or tabs; trailing empty fields may be omitted; blank
lines and lines beginning with '#' are ignored.

```
    # comment line

    homedirectory
    genericname1     pathname1
    genericname2     pathname2
    ...

    % end of generic names, start of address mappings

    hostname1 hardwaretype hardwareaddr1 ipaddr1 genericname suffix
    hostname2 hardwaretype hardwareaddr2 ipaddr2 genericname suffix
```

14. Appendix C – RFC 951   29

```
    ...

Here is a specific example.  Note the 'hardwaretype' number is the
same as that shown in the ARP section of the 'Assigned Numbers' RFC.
The 'hardwaretype' and 'ipaddr' numbers are in decimal;
'hardwareaddr' is in hex.

    # last updated by smith

    /usr/boot
    vmunix          vmunix
    tip             ethertip
    watch           /usr/diag/etherwatch
    gate            gate.

    % end of generic names, start of address mappings

    hamilton        1 02.60.8c.06.34.98     36.19.0.5
    burr            1 02.60.8c.34.11.78     36.44.0.12
    101-gateway     1 02.60.8c.23.ab.35     36.44.0.32      gate 101
    mjh-gateway     1 02.60.8c.12.32.bc     36.42.0.64      gate mjh
    welch-tipa      1 02.60.8c.22.65.32     36.47.0.14      tip
    welch-tipb      1 02.60.8c.12.15.c8     36.46.0.12      tip

In the example above, if 'mjh-gateway' does a default boot, it will
get the file '/usr/boot/gate.mjh'.
```

Croft & Gilmore   [Page 11]

RFC 951   September 1985
Bootstrap Protocol

10. Acknowledgements

   Ross Finlayson (et. al.) produced two earlier RFC's discussing TFTP
   bootstraping [2] using RARP [1].

   We would also like to acknowledge the previous work and comments of
   Noel Chiappa, Bob Lyon, Jeff Mogul, Mark Lewis, and David Plummer.

REFERENCES

   1. Ross Finlayson, Timothy Mann, Jeffrey Mogul, Marvin Theimer. A
       Reverse Address Resolution Protocol. RFC 903, NIC, June, 1984.

   2. Ross Finlayson. Bootstrap Loading using TFTP. RFC 906, NIC,
       June, 1984.

   3. Mark Lottor. Simple File Transfer Protocol. RFC 913, NIC,
       September, 1984.

   4. Jeffrey Mogul. Broadcasting Internet Packets. RFC 919, NIC,
       October, 1984.

   5. David Plummer. An Ethernet Address Resolution Protocol. RFC
       826, NIC, September, 1982.

```
6.  Jon Postel.  File Transfer Protocol.  RFC 765, NIC, June, 1980.

7.  Jon Postel.  User Datagram Protocol.  RFC 768, NIC, August, 1980.

8.  Jon Postel.  Internet Protocol.  RFC 791, NIC, September, 1981.

9.  K. R. Sollins, Noel Chiappa.  The TFTP Protocol.  RFC 783, NIC,
    June, 1981.
```

```
Croft & Gilmore                                              [Page 12]
```

15. Appendix D – RFC 1533

This section is for academic interest only – for universities or research institutes.

```
Network Working Group                                    S. Alexander
Request for Comments: 1533                     Lachman Technology, Inc.
Obsoletes: 1497, 1395, 1084, 1048                             R. Droms
Category: Standards Track                          Bucknell University
                                                         October 1993


              DHCP Options and BOOTP Vendor Extensions

Status of this Memo

   This RFC specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements.  Please refer to the current edition of the "Internet
   Official Protocol Standards" for the standardization state and status
   of this protocol.  Distribution of this memo is unlimited.

Abstract

   The Dynamic Host Configuration Protocol (DHCP) [1] provides a
   framework for passing configuration information to hosts on a TCP/IP
   network.  Configuration parameters and other control information are
   carried in tagged data items that are stored in the "options" field
   of the DHCP message.  The data items themselves are also called
   "options."

   This document specifies the current set of DHCP options.  This
   document will be periodically updated as new options are defined.
    Each superseding document will include the entire current list of
   valid options.

   All of the vendor information extensions defined in RFC 1497 [2] may
   be used as DHCP options.  The definitions given in RFC 1497 are
   included in this document, which supersedes RFC 1497.  All of the
   DHCP options defined in this document, except for those specific to
```

DHCP as defined in section 9, may be used as BOOTP vendor information
extensions.

Table of Contents

Alexander & Droms   [Page 1]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

1. Introduction

   This document specifies options for use with both the Dynamic Host
   Configuration Protocol and the Bootstrap Protocol.

   The full description of DHCP packet formats may be found in the DHCP
   specification document [1], and the full description of BOOTP packet
   formats may be found in the BOOTP specification document [3]. This
   document defines the format of information in the last field of DHCP
   packets ('options') and of BOOTP packets ('vend'). The remainder of
   this section defines a generalized use of this area for giving
   information useful to a wide class of machines, operating systems and
   configurations. Sites with a single DHCP or BOOTP server that is
   shared among heterogeneous clients may choose to define other, site-
   specific formats for the use of the 'options' field.

   Section 2 of this memo describes the formats of DHCP options and
   BOOTP vendor extensions. Section 3 describes options defined in
   previous documents for use with BOOTP (all may also be used with
   DHCP). Sections 4-8 define new options intended for use with both
   DHCP and BOOTP. Section 9 defines options used only in DHCP.

   References further describing most of the options defined in sections
   2-6 can be found in section 12. The use of the options defined in
   section 9 is described in the DHCP specification [1].

   Information on registering new options is contained in section 10.

2. BOOTP Extension/DHCP Option Field Format

   DHCP options have the same format as the BOOTP "vendor extensions"
   defined in RFC 1497 [2]. Options may be fixed length or variable
   length. All options begin with a tag octet, which uniquely
   identifies the option. Fixed-length options without data consist of
   only a tag octet. Only options 0 and 255 are fixed length. All

15. Appendix D – RFC 1533   32

other options are variable-length with a length octet following the
tag octet. The value of the length octet does not include the two
octets specifying the tag and length. The length octet is followed
by "length" octets of data. In the case of some variable-length
options the length field is a constant but must still be specified.

Alexander & Droms   [Page 2]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

    Any options defined subsequent to this document should contain a
    length octet even if the length is fixed or zero.

    All multi-octet quantities are in network byte-order.

    When used with BOOTP, the first four octets of the vendor information
    field have been assigned to the "magic cookie" (as suggested in RFC
    951). This field identifies the mode in which the succeeding data is
    to be interpreted. The value of the magic cookie is the 4 octet
    dotted decimal 99.130.83.99 (or hexadecimal number 63.82.53.63) in
    network byte order.

    All of the "vendor extensions" defined in RFC 1497 are also DHCP
    options.

    Option codes 128 to 254 (decimal) are reserved for site-specific
    options.

    Except for the options in section 9, all options may be used with
    either DHCP or BOOTP.

    Many of these options have their default values specified in other
    documents. In particular, RFC 1122 [4] specifies default values for
    most IP and TCP configuration parameters.

3. RFC 1497 Vendor Extensions

    This section lists the vendor extensions as defined in RFC 1497.
    They are defined here for completeness.

3.1. Pad Option

    The pad option can be used to cause subsequent fields to align on
    word boundaries.

    The code for the pad option is 0, and its length is 1 octet.

     Code
     +-----+
     | 0 |
     +-----+

15. Appendix D – RFC 1533   33

```
Alexander & Droms                                          [Page 3]
RFC 1533        DHCP Options and BOOTP Vendor Extensions    October 1993
```

3.2. End Option

   The end option marks the end of valid information in the vendor
   field. Subsequent octets should be filled with pad options.

   The code for the end option is 255, and its length is 1 octet.

```
    Code
   +-----+
   | 255 |
   +-----+
```

3.3. Subnet Mask

   The subnet mask option specifies the client's subnet mask as per RFC
   950 [5].

   If both the subnet mask and the router option are specified in a DHCP
   reply, the subnet mask option MUST be first.

   The code for the subnet mask option is 1, and its length is 4 octets.

```
    Code    Len          Subnet Mask
   +-----+-----+-----+-----+-----+-----+
   |  1  |  4  | m1  | m2  | m3  | m4  |
   +-----+-----+-----+-----+-----+-----+
```

3.4. Time Offset

   The time offset field specifies the offset of the client's subnet in
   seconds from Coordinated Universal Time (UTC). The offset is
   expressed as a signed 32-bit integer.

   The code for the time offset option is 2, and its length is 4 octets.

```
    Code    Len          Time Offset
   +-----+-----+-----+-----+-----+-----+
   |  2  |  4  | n1  | n2  | n3  | n4  |
   +-----+-----+-----+-----+-----+-----+
```

```
Alexander & Droms                                          [Page 4]
RFC 1533        DHCP Options and BOOTP Vendor Extensions    October 1993
```

3.5. Router Option

The router option specifies a list of IP addresses for routers on the
client's subnet. Routers SHOULD be listed in order of preference.

The code for the router option is 3. The minimum length for the
router option is 4 octets, and the length MUST always be a multiple
of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 3   | n   | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.6. Time Server Option

The time server option specifies a list of RFC 868 [6] time servers
available to the client. Servers SHOULD be listed in order of
preference.

The code for the time server option is 4. The minimum length for
this option is 4 octets, and the length MUST always be a multiple of
4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 4   | n   | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.7. Name Server Option

The name server option specifies a list of IEN 116 [7] name servers
available to the client. Servers SHOULD be listed in order of
preference.

The code for the name server option is 5. The minimum length for
this option is 4 octets, and the length MUST always be a multiple of
4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 5   | n   | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

Alexander & Droms   [Page 5]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

3.8. Domain Name Server Option

The domain name server option specifies a list of Domain Name System
(STD 13, RFC 1035 [8]) name servers available to the client. Servers
SHOULD be listed in order of preference.

The code for the domain name server option is 6. The minimum length
for this option is 4 octets, and the length MUST always be a multiple

of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
|  6  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.9. Log Server Option

   The log server option specifies a list of MIT-LCS UDP log servers
   available to the client. Servers SHOULD be listed in order of
   preference.

   The code for the log server option is 7. The minimum length for this
   option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
|  7  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.10. Cookie Server Option

   The cookie server option specifies a list of RFC 865 [9] cookie
   servers available to the client. Servers SHOULD be listed in order
   of preference.

   The code for the log server option is 8. The minimum length for this
   option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
|  8  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

Alexander & Droms   [Page 6]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

3.11. LPR Server Option

   The LPR server option specifies a list of RFC 1179 [10] line printer
   servers available to the client. Servers SHOULD be listed in order
   of preference.

   The code for the LPR server option is 9. The minimum length for this
   option is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
|  9  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.12. Impress Server Option

15. Appendix D – RFC 1533   36

The Impress server option specifies a list of Imagen Impress servers
available to the client. Servers SHOULD be listed in order of
preference.

The code for the Impress server option is 10. The minimum length for
this option is 4 octets, and the length MUST always be a multiple of
4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 10  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.13. Resource Location Server Option

   This option specifies a list of RFC 887 [11] Resource Location
   servers available to the client. Servers SHOULD be listed in order
   of preference.

   The code for this option is 11. The minimum length for this option
   is 4 octets, and the length MUST always be a multiple of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 11  |  n  | a1  | a2  | a3  | a4  | a1  | a2  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

Alexander & Droms   [Page 7]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

3.14. Host Name Option

   This option specifies the name of the client. The name may or may
   not be qualified with the local domain name (see section 3.17 for the
   preferred way to retrieve the domain name). See RFC 1035 for
   character set restrictions.

   The code for this option is 12, and its minimum length is 1.

```
 Code   Len             Host Name
+-----+-----+-----+-----+-----+-----+-----+-----+--
| 12  |  n  | h1  | h2  | h3  | h4  | h5  | h6  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+--
```

3.15. Boot File Size Option

   This option specifies the length in 512-octet blocks of the default
   boot image for the client. The file length is specified as an
   unsigned 16-bit integer.

   The code for this option is 13, and its length is 2.

15. Appendix D – RFC 1533   37

```
 Code    Len    File Size
+-----+-----+-----+-----+
| 13  |  2  | l1  | l2  |
+-----+-----+-----+-----+
```

3.16. Merit Dump File

   This option specifies the path-name of a file to which the client's
   core image should be dumped in the event the client crashes. The
   path is formatted as a character string consisting of characters from
   the NVT ASCII character set.

   The code for this option is 14. Its minimum length is 1.

```
 Code    Len       Dump File Pathname
+-----+-----+-----+-----+-----+-----+---
| 14  |  n  | n1  | n2  | n3  | n4  | ...
+-----+-----+-----+-----+-----+-----+---
```

Alexander & Droms   [Page 8]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

   3.17. Domain Name

      This option specifies the domain name that client should use when
      resolving hostnames via the Domain Name System.

      The code for this option is 15. Its minimum length is 1.

```
 Code    Len          Domain Name
+-----+-----+-----+-----+-----+-----+--
| 15  |  n  | d1  | d2  | d3  | d4  | ...
+-----+-----+-----+-----+-----+-----+--
```

3.18. Swap Server

   This specifies the IP address of the client's swap server.

   The code for this option is 16 and its length is 4.

```
 Code    Len    Swap Server Address
+-----+-----+-----+-----+-----+-----+
| 16  |  n  | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+-----+-----+
```

3.19. Root Path

   This option specifies the path-name that contains the client's root
   disk. The path is formatted as a character string consisting of
   characters from the NVT ASCII character set.

15. Appendix D – RFC 1533   38

The code for this option is 17. Its minimum length is 1.

```
 Code    Len       Root Disk Pathname
+-----+-----+-----+-----+-----+-----+---
|  17 |  n  |  n1 |  n2 |  n3 |  n4 | ...
+-----+-----+-----+-----+-----+-----+---
```

Alexander & Droms   [Page 9]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

3.20. Extensions Path

   A string to specify a file, retrievable via TFTP, which contains
   information which can be interpreted in the same way as the 64-octet
   vendor-extension field within the BOOTP response, with the following
   exceptions:

          - the length of the file is unconstrained;
          - all references to Tag 18 (i.e., instances of the
            BOOTP Extensions Path field) within the file are
            ignored.

   The code for this option is 18. Its minimum length is 1.

```
 Code    Len       Extensions Pathname
+-----+-----+-----+-----+-----+-----+---
|  18 |  n  |  n1 |  n2 |  n3 |  n4 | ...
+-----+-----+-----+-----+-----+-----+---
```

4. IP Layer Parameters per Host

   This section details the options that affect the operation of the IP
   layer on a per-host basis.

4.1. IP Forwarding Enable/Disable Option

   This option specifies whether the client should configure its IP
   layer for packet forwarding. A value of 0 means disable IP
   forwarding, and a value of 1 means enable IP forwarding.

   The code for this option is 19, and its length is 1.

```
 Code    Len  Value
+-----+-----+-----+
|  19 |  1  | 0/1 |
+-----+-----+-----+
```

15. Appendix D − RFC 1533   39

4.2. Non-Local Source Routing Enable/Disable Option

   This option specifies whether the client should configure its IP
   layer to allow forwarding of datagrams with non-local source routes
   (see Section 3.3.5 of [4] for a discussion of this topic). A value
   of 0 means disallow forwarding of such datagrams, and a value of 1
   means allow forwarding.

   The code for this option is 20, and its length is 1.

    Code   Len Value
   +-----+-----+-----+
   | 20 | 1 | 0/1 |
   +-----+-----+-----+

4.3. Policy Filter Option

   This option specifies policy filters for non-local source routing.
   The filters consist of a list of IP addresses and masks which specify
   destination/mask pairs with which to filter incoming source routes.

   Any source routed datagram whose next-hop address does not match one
   of the filters should be discarded by the client.

   See [4] for further information.

   The code for this option is 21. The minimum length of this option is
   8, and the length MUST be a multiple of 8.

    Code   Len   Address 1   Mask 1
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   | 21 | n | a1 | a2 | a3 | a4 | m1 | m2 | m3 | m4 |
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
           Address 2   Mask 2
   +-----+-----+-----+-----+-----+-----+-----+-----+---
   | a1 | a2 | a3 | a4 | m1 | m2 | m3 | m4 | ...
   +-----+-----+-----+-----+-----+-----+-----+-----+---

15. Appendix D − RFC 1533   40

```
Alexander & Droms                                          [Page 11]
RFC 1533       DHCP Options and BOOTP Vendor Extensions   October 1993
```

4.4. Maximum Datagram Reassembly Size

   This option specifies the maximum size datagram that the client
   should be prepared to reassemble. The size is specified as a 16-bit
   unsigned integer. The minimum value legal value is 576.

   The code for this option is 22, and its length is 2.

```
    Code    Len       Size
   +-----+-----+-----+-----+
   |  22 |  2  |  s1 |  s2 |
   +-----+-----+-----+-----+
```

4.5. Default IP Time-to-live

   This option specifies the default time-to-live that the client should
   use on outgoing datagrams. The TTL is specified as an octet with a
   value between 1 and 255.

   The code for this option is 23, and its length is 1.

```
    Code    Len   TTL
   +-----+-----+-----+
   |  23 |  1  | ttl |
   +-----+-----+-----+
```

4.6. Path MTU Aging Timeout Option

   This option specifies the timeout (in seconds) to use when aging Path
   MTU values discovered by the mechanism defined in RFC 1191 [12]. The
   timeout is specified as a 32-bit unsigned integer.

   The code for this option is 24, and its length is 4.

```
    Code    Len            Timeout
   +-----+-----+-----+-----+-----+-----+
   |  24 |  4  |  t1 |  t2 |  t3 |  t4 |
   +-----+-----+-----+-----+-----+-----+
```

4.7. Path MTU Plateau Table Option

   This option specifies a table of MTU sizes to use when performing
   Path MTU Discovery as defined in RFC 1191. The table is formatted as
   a list of 16−bit unsigned integers, ordered from smallest to largest.
   The minimum MTU value cannot be smaller than 68.

   The code for this option is 25. Its minimum length is 2, and the
   length MUST be a multiple of 2.

    Code   Len   Size 1   Size 2
   +-----+-----+-----+-----+-----+-----+---
   | 25 | n | s1 | s2 | s1 | s2 | ...
   +-----+-----+-----+-----+-----+-----+---

5. IP Layer Parameters per Interface

   This section details the options that affect the operation of the IP
   layer on a per-interface basis. It is expected that a client can
   issue multiple requests, one per interface, in order to configure
   interfaces with their specific parameters.

5.1. Interface MTU Option

   This option specifies the MTU to use on this interface. The MTU is
   specified as a 16−bit unsigned integer. The minimum legal value for
   the MTU is 68.

   The code for this option is 26, and its length is 2.

    Code   Len   MTU
   +-----+-----+-----+-----+
   | 26 | 2 | m1 | m2 |
   +-----+-----+-----+-----+

5.2. All Subnets are Local Option

This option specifies whether or not the client may assume that all
subnets of the IP network to which the client is connected use the
same MTU as the subnet of that network to which the client is
directly connected. A value of 1 indicates that all subnets share
the same MTU. A value of 0 means that the client should assume that
some subnets of the directly connected network may have smaller MTUs.

The code for this option is 27, and its length is 1.

```
 Code   Len  Value
+-----+-----+-----+
|  27 |  1  | 0/1 |
+-----+-----+-----+
```

5.3. Broadcast Address Option

This option specifies the broadcast address in use on the client's
subnet. Legal values for broadcast addresses are specified in
section 3.2.1.3 of [4].

The code for this option is 28, and its length is 4.

```
 Code   Len      Broadcast Address
+-----+-----+-----+-----+-----+-----+
|  28 |  4  | b1  | b2  | b3  | b4  |
+-----+-----+-----+-----+-----+-----+
```

5.4. Perform Mask Discovery Option

This option specifies whether or not the client should perform subnet
mask discovery using ICMP. A value of 0 indicates that the client
should not perform mask discovery. A value of 1 means that the
client should perform mask discovery.

The code for this option is 29, and its length is 1.

```
 Code   Len  Value
+-----+-----+-----+
|  29 |  1  | 0/1 |
+-----+-----+-----+
```

Alexander & Droms   [Page 14]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

5.5. Mask Supplier Option

This option specifies whether or not the client should respond to
subnet mask requests using ICMP. A value of 0 indicates that the
client should not respond. A value of 1 means that the client should
respond.

The code for this option is 30, and its length is 1.

15. Appendix D – RFC 1533   43

```
   Code   Len  Value
  +-----+-----+-----+
  |  30 |  1  | 0/1 |
  +-----+-----+-----+
```

5.6. Perform Router Discovery Option

   This option specifies whether or not the client should solicit
   routers using the Router Discovery mechanism defined in RFC 1256
   [13]. A value of 0 indicates that the client should not perform
   router discovery. A value of 1 means that the client should perform
   router discovery.

   The code for this option is 31, and its length is 1.

```
   Code   Len  Value
  +-----+-----+-----+
  |  31 |  1  | 0/1 |
  +-----+-----+-----+
```

5.7. Router Solicitation Address Option

   This option specifies the address to which the client should transmit
   router solicitation requests.

   The code for this option is 32, and its length is 4.

```
   Code   Len            Address
  +-----+-----+-----+-----+-----+-----+
  |  32 |  4  | a1  | a2  | a3  | a4  |
  +-----+-----+-----+-----+-----+-----+
```

RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

5.8. Static Route Option

   This option specifies a list of static routes that the client should
   install in its routing cache. If multiple routes to the same
   destination are specified, they are listed in descending order of
   priority.

   The routes consist of a list of IP address pairs. The first address
   is the destination address, and the second address is the router for
   the destination.

   The default route (0.0.0.0) is an illegal destination for a static
   route. See section 3.5 for information about the router option.

   The code for this option is 33. The minimum length of this option is
   8, and the length MUST be a multiple of 8.

15. Appendix D – RFC 1533   44

```
  Code    Len          Destination 1          Router 1
 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 | 33  |  n  | d1  | d2  | d3  | d4  | r1  | r2  | r3  | r4  |
 +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
        Destination 2          Router 2
 +-----+-----+-----+-----+-----+-----+-----+-----+---
 | d1  | d2  | d3  | d4  | r1  | r2  | r3  | r4  | ...
 +-----+-----+-----+-----+-----+-----+-----+-----+---
```

6. Link Layer Parameters per Interface

   This section lists the options that affect the operation of the data
   link layer on a per-interface basis.

6.1. Trailer Encapsulation Option

   This option specifies whether or not the client should negotiate the
   use of trailers (RFC 893 [14]) when using the ARP protocol. A value
   of 0 indicates that the client should not attempt to use trailers. A
   value of 1 means that the client should attempt to use trailers.

   The code for this option is 34, and its length is 1.

```
  Code   Len  Value
 +-----+-----+-----+
 | 34  |  1  | 0/1 |
 +-----+-----+-----+
```

6.2. ARP Cache Timeout Option

   This option specifies the timeout in seconds for ARP cache entries.
   The time is specified as a 32-bit unsigned integer.

   The code for this option is 35, and its length is 4.

```
  Code   Len           Time
 +-----+-----+-----+-----+-----+-----+
 | 35  |  4  | t1  | t2  | t3  | t4  |
 +-----+-----+-----+-----+-----+-----+
```

6.3. Ethernet Encapsulation Option

   This option specifies whether or not the client should use Ethernet
   Version 2 (RFC 894 [15]) or IEEE 802.3 (RFC 1042 [16]) encapsulation
   if the interface is an Ethernet. A value of 0 indicates that the
   client should use RFC 894 encapsulation. A value of 1 means that the
   client should use RFC 1042 encapsulation.

   The code for this option is 36, and its length is 1.

```
  Code    Len  Value
```

15. Appendix D – RFC 1533   45

```
+-----+-----+-----+
| 36  |  1  | 0/1 |
+-----+-----+-----+
```

7. TCP Parameters

   This section lists the options that affect the operation of the TCP
   layer on a per-interface basis.

7.1. TCP Default TTL Option

   This option specifies the default TTL that the client should use when
   sending TCP segments. The value is represented as an 8-bit unsigned
   integer. The minimum value is 1.

   The code for this option is 37, and its length is 1.

```
 Code   Len   TTL
+-----+-----+-----+
| 37  |  1  |  n  |
+-----+-----+-----+
```

7.2. TCP Keepalive Interval Option

   This option specifies the interval (in seconds) that the client TCP
   should wait before sending a keepalive message on a TCP connection.
   The time is specified as a 32-bit unsigned integer. A value of zero
   indicates that the client should not generate keepalive messages on
   connections unless specifically requested by an application.

   The code for this option is 38, and its length is 4.

```
 Code   Len            Time
+-----+-----+-----+-----+-----+-----+
| 38  |  4  | t1  | t2  | t3  | t4  |
+-----+-----+-----+-----+-----+-----+
```

7.3. TCP Keepalive Garbage Option

   This option specifies the whether or not the client should send TCP
   keepalive messages with a octet of garbage for compatibility with
   older implementations. A value of 0 indicates that a garbage octet
   should not be sent. A value of 1 indicates that a garbage octet
   should be sent.

   The code for this option is 39, and its length is 1.

```
 Code   Len  Value
+-----+-----+-----+
| 39  |  1  | 0/1 |
+-----+-----+-----+
```

15. Appendix D – RFC 1533   46

8. Application and Service Parameters

   This section details some miscellaneous options used to configure
   miscellaneous applications and services.

8.1. Network Information Service Domain Option

   This option specifies the name of the client's NIS [17] domain. The
   domain is formatted as a character string consisting of characters
   from the NVT ASCII character set.

   The code for this option is 40. Its minimum length is 1.

```
    Code   Len      NIS Domain Name
   +-----+-----+-----+-----+-----+-----+---
   |  40 |  n  |  n1 |  n2 |  n3 |  n4 | ...
   +-----+-----+-----+-----+-----+-----+---
```

Alexander & Droms   [Page 18]

RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

8.2. Network Information Servers Option

   This option specifies a list of IP addresses indicating NIS servers
   available to the client. Servers SHOULD be listed in order of
   preference.

   The code for this option is 41. Its minimum length is 4, and the
   length MUST be a multiple of 4.

```
    Code   Len          Address 1               Address 2
   +-----+-----+-----+-----+-----+-----+-----+-----+--
   |  41 |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
   +-----+-----+-----+-----+-----+-----+-----+-----+--
```

8.3. Network Time Protocol Servers Option

   This option specifies a list of IP addresses indicating NTP [18]
   servers available to the client. Servers SHOULD be listed in order
   of preference.

   The code for this option is 42. Its minimum length is 4, and the
   length MUST be a multiple of 4.

```
    Code   Len          Address 1               Address 2
   +-----+-----+-----+-----+-----+-----+-----+-----+--
   |  42 |  n  |  a1 |  a2 |  a3 |  a4 |  a1 |  a2 |  ...
   +-----+-----+-----+-----+-----+-----+-----+-----+--
```

8.4. Vendor Specific Information

   This option is used by clients and servers to exchange vendor-
   specific information. The information is an opaque object of n
   octets, presumably interpreted by vendor-specific code on the clients
   and servers. The definition of this information is vendor specific.
   The vendor is indicated in the class-identifier option. Servers not
   equipped to interpret the vendor-specific information sent by a
   client MUST ignore it (although it may be reported). Clients which

15. Appendix D – RFC 1533   47

do not receive desired vendor-specific information SHOULD make an
attempt to operate without it, although they may do so (and announce
they are doing so) in a degraded mode.

If a vendor potentially encodes more than one item of information in
this option, then the vendor SHOULD encode the option using
"Encapsulated vendor-specific options" as described below:

The Encapsulated vendor-specific options field SHOULD be encoded as a
sequence of code/length/value fields of identical syntax to the DHCP
options field with the following exceptions:

Alexander & Droms   [Page 19]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

      1) There SHOULD NOT be a "magic cookie" field in the encapsulated
         vendor-specific extensions field.

      2) Codes other than 0 or 255 MAY be redefined by the vendor within
         the encapsulated vendor-specific extensions field, but SHOULD
         conform to the tag-length-value syntax defined in section 2.

      3) Code 255 (END), if present, signifies the end of the
         encapsulated vendor extensions, not the end of the vendor
         extensions field. If no code 255 is present, then the end of
         the enclosing vendor-specific information field is taken as the
         end of the encapsulated vendor-specific extensions field.

   The code for this option is 43 and its minimum length is 1.

   Code   Len   Vendor-specific information
   +-----+-----+-----+-----+---
   | 43 | n | i1 | i2 | ...
   +-----+-----+-----+-----+---

   When encapsulated vendor-specific extensions are used, the
   information bytes 1-n have the following format:

    Code   Len   Data item   Code   Len   Data item   Code
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
   | T1 | n | d1 | d2 | ... | T2 | n | D1 | D2 | ... | ... |
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

8.5. NetBIOS over TCP/IP Name Server Option

   The NetBIOS name server (NBNS) option specifies a list of RFC
   1001/1002 [19] [20] NBNS name servers listed in order of preference.

   The code for this option is 44. The minimum length of the option is
   4 octets, and the length must always be a multiple of 4.

    Code   Len   Address 1   Address 2
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
   | 44 | n | a1 | a2 | a3 | a4 | b1 | b2 | b3 | b4 | ...
   +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----

15. Appendix D – RFC 1533   48

8.6. NetBIOS over TCP/IP Datagram Distribution Server Option

   The NetBIOS datagram distribution server (NBDD) option specifies a
   list of RFC 1001/1002 NBDD servers listed in order of preference. The
   code for this option is 45. The minimum length of the option is 4
   octets, and the length must always be a multiple of 4.

```
 Code   Len         Address 1               Address 2
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
| 45  |  n  | a1  | a2  | a3  | a4  | b1  | b2  | b3  | b4  | ...
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----
```

8.7. NetBIOS over TCP/IP Node Type Option

   The NetBIOS node type option allows NetBIOS over TCP/IP clients which
   are configurable to be configured as described in RFC 1001/1002. The
   value is specified as a single octet which identifies the client type
   as follows:

```
   Value           Node Type
   -----           ---------
   0x1             B-node
   0x2             P-node
   0x4             M-node
   0x8             H-node
```

   In the above chart, the notation '0x' indicates a number in base-16
   (hexadecimal).

   The code for this option is 46. The length of this option is always
   1.

```
 Code   Len  Node Type
+-----+-----+-----------+
| 46  |  1  | see above |
+-----+-----+-----------+
```

Alexander & Droms   [Page 21]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

8.8. NetBIOS over TCP/IP Scope Option

   The NetBIOS scope option specifies the NetBIOS over TCP/IP scope
   parameter for the client as specified in RFC 1001/1002. See [19],
   [20], and [8] for character-set restrictions.

   The code for this option is 47. The minimum length of this option is
   1.

     Code   Len   NetBIOS Scope
    +-----+-----+-----+-----+-----+-----+----
    | 47 | n | s1 | s2 | s3 | s4 | ...
    +-----+-----+-----+-----+-----+-----+----

8.9. X Window System Font Server Option

   This option specifies a list of X Window System [21] Font servers
   available to the client. Servers SHOULD be listed in order of
   preference.

   The code for this option is 48. The minimum length of this option is
   4 octets, and the length MUST be a multiple of 4.

     Code   Len   Address 1   Address 2
    +-----+-----+-----+-----+-----+-----+-----+-----+---
    | 48 | n | a1 | a2 | a3 | a4 | a1 | a2 | ...
    +-----+-----+-----+-----+-----+-----+-----+-----+---

8.10. X Window System Display Manager Option

   This option specifies a list of IP addresses of systems that are
   running the X Window System Display Manager and are available to the
   client.

   Addresses SHOULD be listed in order of preference.

   The code for the this option is 49. The minimum length of this option
   is 4, and the length MUST be a multiple of 4.

     Code   Len   Address 1   Address 2

    +-----+-----+-----+-----+-----+-----+-----+-----+---
    | 49 | n | a1 | a2 | a3 | a4 | a1 | a2 | ...
    +-----+-----+-----+-----+-----+-----+-----+-----+---

Alexander & Droms   [Page 22]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

9. DHCP Extensions

   This section details the options that are specific to DHCP.

15. Appendix D – RFC 1533   50

9.1. Requested IP Address

This option is used in a client request (DHCPDISCOVER) to allow the
client to request that a particular IP address be assigned.

The code for this option is 50, and its length is 4.

```
 Code   Len           Address
+-----+-----+-----+-----+-----+-----+
|  50 |  4  | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+-----+-----+
```

9.2. IP Address Lease Time

This option is used in a client request (DHCPDISCOVER or DHCPREQUEST)
to allow the client to request a lease time for the IP address. In a
server reply (DHCPOFFER), a DHCP server uses this option to specify
the lease time it is willing to offer.

The time is in units of seconds, and is specified as a 32–bit
unsigned integer.

The code for this option is 51, and its length is 4.

```
 Code   Len          Lease Time
+-----+-----+-----+-----+-----+-----+
|  51 |  4  | t1  | t2  | t3  | t4  |
+-----+-----+-----+-----+-----+-----+
```

9.3. Option Overload

This option is used to indicate that the DHCP "sname" or "file"
fields are being overloaded by using them to carry DHCP options. A
DHCP server inserts this option if the returned parameters will
exceed the usual space allotted for options.

If this option is present, the client interprets the specified
additional fields after it concludes interpretation of the standard
option fields.

The code for this option is 52, and its length is 1. Legal values
for this option are:

Alexander & Droms   [Page 23]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

```
         Value   Meaning
         -----   --------
           1     the "file" field is used to hold options
           2     the "sname" field is used to hold options
           3     both fields are used to hold options

  Code   Len  Value
 +-----+-----+-----+
 |  52 |  1  |1/2/3|
 +-----+-----+-----+
```

9.4. DHCP Message Type

This option is used to convey the type of the DHCP message. The code
for this option is 53, and its length is 1. Legal values for this
option are:

```
        Value   Message Type
        -----   ------------
          1     DHCPDISCOVER
          2     DHCPOFFER
          3     DHCPREQUEST
          4     DHCPDECLINE
          5     DHCPACK
          6     DHCPNAK
          7     DHCPRELEASE
```

```
 Code   Len   Type
+-----+-----+-----+
| 53  |  1  | 1-7 |
+-----+-----+-----+
```

9.5. Server Identifier

This option is used in DHCPOFFER and DHCPREQUEST messages, and may
optionally be included in the DHCPACK and DHCPNAK messages. DHCP
servers include this option in the DHCPOFFER in order to allow the
client to distinguish between lease offers. DHCP clients indicate
which of several lease offers is being accepted by including this
option in a DHCPREQUEST message.

The identifier is the IP address of the selected server.

Alexander & Droms   [Page 24]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

The code for this option is 54, and its length is 4.

```
 Code   Len             Address
+-----+-----+-----+-----+-----+-----+
| 54  |  4  | a1  | a2  | a3  | a4  |
+-----+-----+-----+-----+-----+-----+
```

9.6. Parameter Request List

This option is used by a DHCP client to request values for specified
configuration parameters. The list of requested parameters is
specified as n octets, where each octet is a valid DHCP option code
as defined in this document.

The client MAY list the options in order of preference. The DHCP
server is not required to return the options in the requested order,
but MUST try to insert the requested options in the order requested

15. Appendix D − RFC 1533   52

by the client.

The code for this option is 55. Its minimum length is 1.

```
 Code   Len   Option Codes
+-----+-----+-----+-----+---
| 55  |  n  | c1  | c2  | ...
+-----+-----+-----+-----+---
```

9.7. Message

   This option is used by a DHCP server to provide an error message to a
   DHCP client in a DHCPNAK message in the event of a failure. A client
   may use this option in a DHCPDECLINE message to indicate the why the
   client declined the offered parameters. The message consists of n
   octets of NVT ASCII text, which the client may display on an
   available output device.

   The code for this option is 56 and its minimum length is 1.

```
 Code   Len      Text
+-----+-----+-----+-----+---
| 56  |  n  | c1  | c2  | ...
+-----+-----+-----+-----+---
```

Alexander & Droms   [Page 25]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

9.8. Maximum DHCP Message Size

   This option specifies the maximum length DHCP message that it is
   willing to accept. The length is specified as an unsigned 16-bit
   integer. A client may use the maximum DHCP message size option in
   DHCPDISCOVER or DHCPREQUEST messages, but should not use the option
   in DHCPDECLINE messages.

   The code for this option is 57, and its length is 2. The minimum
   legal value is 576 octets.

```
 Code   Len     Length
+-----+-----+-----+-----+
| 57  |  2  | l1  | l2  |
+-----+-----+-----+-----+
```

9.9. Renewal (T1) Time Value

   This option specifies the time interval from address assignment until
   the client transitions to the RENEWING state.

   The value is in units of seconds, and is specified as a 32-bit
   unsigned integer.

15. Appendix D – RFC 1533   53

The code for this option is 58, and its length is 4.

```
 Code   Len         T1 Interval
+-----+-----+-----+-----+-----+-----+
|  58 |  4  |  t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+-----+-----+
```

9.10. Rebinding (T2) Time Value

   This option specifies the time interval from address assignment until
   the client transitions to the REBINDING state.

   The value is in units of seconds, and is specified as a 32-bit
   unsigned integer.

   The code for this option is 59, and its length is 4.

```
 Code   Len         T2 Interval
+-----+-----+-----+-----+-----+-----+
|  59 |  4  |  t1 |  t2 |  t3 |  t4 |
+-----+-----+-----+-----+-----+-----+
```

Alexander & Droms   [Page 26]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

9.11. Class-identifier

   This option is used by DHCP clients to optionally identify the type
   and configuration of a DHCP client. The information is a string of n
   octets, interpreted by servers. Vendors and sites may choose to
   define specific class identifiers to convey particular configuration
   or other identification information about a client. For example, the
   identifier may encode the client's hardware configuration. Servers
   not equipped to interpret the class-specific information sent by a
   client MUST ignore it (although it may be reported).

   The code for this option is 60, and its minimum length is 1.

```
 Code   Len   Class-Identifier
+-----+-----+-----+-----+---
|  60 |  n  |  i1 |  i2 | ...
+-----+-----+-----+-----+---
```

9.12. Client-identifier

   This option is used by DHCP clients to specify their unique
   identifier. DHCP servers use this value to index their database of
   address bindings. This value is expected to be unique for all
   clients in an administrative domain.

   Identifiers consist of a type-value pair, similar to the

   It is expected that this field will typically contain a hardware type
   and hardware address, but this is not required. Current legal values
   for hardware types are defined in [22].

15. Appendix D – RFC 1533   54

The code for this option is 61, and its minimum length is 2.

```
Code   Len   Type  Client-Identifier
+-----+-----+-----+-----+-----+---
| 61 |  n  | t1  | i1  | i2  | ...
+-----+-----+-----+-----+-----+---
```

10. Extensions

   Additional generic data fields may be registered by contacting:

      Internet Assigned Numbers Authority (IANA)
      USC/Information Sciences Institute
      4676 Admiralty Way
      Marina del Rey, California 90292-6695

      or by email as: iana@isi.edu

   Implementation specific use of undefined generic types (those in the
   range 61-127) may conflict with other implementations, and
   registration is required.

11. Acknowledgements

   The authors would like to thank Philip Almquist for his feedback on
   this document. The comments of the DHCP Working Group are also
   gratefully acknowledged. In particular, Mike Carney and Jon Dreyer
   from SunSelect suggested the current format of the Vendor-specific
   Information option.

   RFC 1497 is based on earlier work by Philip Prindeville, with help
   from Drew Perkins, Bill Croft, and Steve Deering.

12. References

   [1] Droms, R., "Dynamic Host Configuration Protocol", RFC 1531,
       Bucknell University, October 1993.

   [2] Reynolds, J., "BOOTP Vendor Information Extensions", RFC 1497,
       USC/Information Sciences Institute, August 1993.

   [3] Croft, W., and J. Gilmore, "Bootstrap Protocol", RFC 951,
       Stanford University and Sun Microsystems, September 1985.

   [4] Braden, R., Editor, "Requirements for Internet Hosts -
       Communication Layers", STD 3, RFC 1122, USC/Information Sciences
       Institute, October 1989.

   [5] Mogul, J., and J. Postel, "Internet Standard Subnetting
       Procedure", STD 5, RFC 950, USC/Information Sciences Institute,
       August 1985.

   [6] Postel, J., and K. Harrenstien, "Time Protocol", STD 26, RFC
       868, USC/Information Sciences Institute, SRI, May 1983.

   [7] Postel, J., "Name Server", IEN 116, USC/Information Sciences

        Institute, August 1979.

    [8] Mockapetris, P., "Domain Names – Implementation and
        Specification", STD 13, RFC 1035, USC/Information Sciences
        Institute, November 1987.

    [9] Postel, J., "Quote of the Day Protocol", STD 23, RFC 865,
        USC/Information Sciences Institute, May 1983.

Alexander & Droms   [Page 28]
RFC 1533   DHCP Options and BOOTP Vendor Extensions   October 1993

    [10] McLaughlin, L., "Line Printer Daemon Protocol", RFC 1179, The
         Wollongong Group, August 1990.

    [11] Accetta, M., "Resource Location Protocol", RFC 887, CMU,
         December 1983.

    [12] Mogul, J. and S. Deering, "Path MTU Discovery", RFC 1191,
         DECWRL, Stanford University, November 1990.

    [13] Deering, S., "ICMP Router Discovery Messages", RFC 1256,
         Xerox PARC, September 1991.

    [14] Leffler, S. and M. Karels, "Trailer Encapsulations", RFC 893,
         U. C. Berkeley, April 1984.

    [15] Hornig, C., "Standard for the Transmission of IP Datagrams over
         Ethernet Networks", RFC 894, Symbolics, April 1984.

    [16] Postel, J. and J. Reynolds, "Standard for the Transmission of
         IP Datagrams Over IEEE 802 Networks", RFC 1042, USC/Information
         Sciences Institute, February 1988.

    [17] Sun Microsystems, "System and Network Administration", March
         1990.

    [18] Mills, D., "Internet Time Synchronization: The Network Time
         Protocol", RFC 1305, UDEL, March 1992.

    [19] NetBIOS Working Group, "Protocol Standard for a NetBIOS Service
         on a TCP/UDP transport: Concepts and Methods", STD 19, RFC 1001,
         March 1987.

    [20] NetBIOS Working Group, "Protocol Standard for a NetBIOS Service
         on a TCP/UDP transport: Detailed Specifications", STD 19, RFC
         1002, March 1987.

    [21] Scheifler, R., "FYI On the X Window System", FYI 6, RFC 1198,
         MIT Laboratory for Computer Science, January 1991.

    [22] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1340,
         USC/Information Sciences Institute, July 1992.

   13. Security Considerations

      Security issues are not discussed in this memo.

15. Appendix D – RFC 1533   56

```
Alexander & Droms                                            [Page 29]
RFC 1533       DHCP Options and BOOTP Vendor Extensions   October 1993


14. Authors' Addresses

   Steve Alexander
   Lachman Technology, Inc.
   1901 North Naper Boulevard
   Naperville, IL 60563–8895

   Phone: (708) 505–9555 x256
   EMail: stevea@lachman.com


   Ralph Droms
   Computer Science Department
   323 Dana Engineering
   Bucknell University
   Lewisburg, PA 17837

   Phone: (717) 524–1145
   EMail: droms@bucknell.edu


Alexander & Droms                                            [Page 30]
```

16. Appendix E – RFC 1350

This section is for academic interest only – for universities or research institutes.

```
Network Working Group                                      K. Sollins
Request For Comments: 1350                                        MIT
STD: 33                                                     July 1992
Obsoletes: RFC 783



                     THE TFTP PROTOCOL (REVISION 2)


Status of this Memo

   This RFC specifies an IAB standards track protocol for the Internet
   community, and requests discussion and suggestions for improvements.
   Please refer to the current edition of the "IAB Official Protocol
   Standards" for the standardization state and status of this protocol.
   Distribution of this memo is unlimited.
```

Summary

TFTP is a very simple protocol used to transfer files. It is from
this that its name comes, Trivial File Transfer Protocol or TFTP.
Each nonterminal packet is acknowledged separately. This document
describes the protocol and its types of packets. The document also
explains the reasons behind some of the design decisions.

Acknowlegements

The protocol was originally designed by Noel Chiappa, and was
redesigned by him, Bob Baldwin and Dave Clark, with comments from
Steve Szymanski. The current revision of the document includes
modifications stemming from discussions with and suggestions from
Larry Allen, Noel Chiappa, Dave Clark, Geoff Cooper, Mike Greenwald,
Liza Martin, David Reed, Craig Milo Rogers (of USC-ISI), Kathy
Yellick, and the author. The acknowledgement and retransmission
scheme was inspired by TCP, and the error mechanism was suggested by
PARC's EFTP abort message.

The May, 1992 revision to fix the "Sorcerer's Apprentice" protocol
bug [4] and other minor document problems was done by Noel Chiappa.

This research was supported by the Advanced Research Projects Agency
of the Department of Defense and was monitored by the Office of Naval
Research under contract number N00014-75-C-0661.

1. Purpose

TFTP is a simple protocol to transfer files, and therefore was named
the Trivial File Transfer Protocol or TFTP. It has been implemented
on top of the Internet User Datagram protocol (UDP or Datagram) [2]

Sollins   [Page 1]
RFC 1350   TFTP Revision 2   July 1992

so it may be used to move files between machines on different
networks implementing UDP. (This should not exclude the possibility
of implementing TFTP on top of other datagram protocols.) It is
designed to be small and easy to implement. Therefore, it lacks most
of the features of a regular FTP. The only thing it can do is read
and write files (or mail) from/to a remote server. It cannot list
directories, and currently has no provisions for user authentication.
In common with other Internet protocols, it passes 8 bit bytes of
data.

Three modes of transfer are currently supported: netascii (This is
ascii as defined in "USA Standard Code for Information Interchange"
[1] with the modifications specified in "Telnet Protocol
Specification" [3].) Note that it is 8 bit ascii. The term
"netascii" will be used throughout this document to mean this
particular version of ascii.); octet (This replaces the "binary" mode
of previous versions of this document.) raw 8 bit bytes; mail,
netascii characters sent to a user rather than a file. (The mail
mode is obsolete and should not be implemented or used.) Additional
modes can be defined by pairs of cooperating hosts.

Reference [4] (section 4.2) should be consulted for further valuable

directives and suggestions on TFTP.

2. Overview of the Protocol

   Any transfer begins with a request to read or write a file, which
   also serves to request a connection. If the server grants the
   request, the connection is opened and the file is sent in fixed
   length blocks of 512 bytes. Each data packet contains one block of
   data, and must be acknowledged by an acknowledgment packet before the
   next packet can be sent. A data packet of less than 512 bytes
   signals termination of a transfer. If a packet gets lost in the
   network, the intended recipient will timeout and may retransmit his
   last packet (which may be data or an acknowledgment), thus causing
   the sender of the lost packet to retransmit that lost packet. The
   sender has to keep just one packet on hand for retransmission, since
   the lock step acknowledgment guarantees that all older packets have
   been received. Notice that both machines involved in a transfer are
   considered senders and receivers. One sends data and receives
   acknowledgments, the other sends acknowledgments and receives data.

   Most errors cause termination of the connection. An error is
   signalled by sending an error packet. This packet is not
   acknowledged, and not retransmitted (i.e., a TFTP server or user may
   terminate after sending an error message), so the other end of the
   connection may not get it. Therefore timeouts are used to detect
   such a termination when the error packet has been lost. Errors are

   caused by three types of events: not being able to satisfy the
   request (e.g., file not found, access violation, or no such user),
   receiving a packet which cannot be explained by a delay or
   duplication in the network (e.g., an incorrectly formed packet), and
   losing access to a necessary resource (e.g., disk full or access
   denied during a transfer).

   TFTP recognizes only one error condition that does not cause
   termination, the source port of a received packet being incorrect.
   In this case, an error packet is sent to the originating host.

   This protocol is very restrictive, in order to simplify
   implementation. For example, the fixed length blocks make allocation
   straight forward, and the lock step acknowledgement provides flow
   control and eliminates the need to reorder incoming data packets.

3. Relation to other Protocols

   As mentioned TFTP is designed to be implemented on top of the
   Datagram protocol (UDP). Since Datagram is implemented on the
   Internet protocol, packets will have an Internet header, a Datagram
   header, and a TFTP header. Additionally, the packets may have a
   header (LNI, ARPA header, etc.) to allow them through the local
   transport medium. As shown in Figure 3−1, the order of the contents
   of a packet will be: local medium header, if used, Internet header,
   Datagram header, TFTP header, followed by the remainder of the TFTP
   packet. (This may or may not be data depending on the type of packet
   as specified in the TFTP header.) TFTP does not specify any of the
   values in the Internet header. On the other hand, the source and

16. Appendix E − RFC 1350   59

destination port fields of the Datagram header (its format is given
in the appendix) are used by TFTP and the length field reflects the
size of the TFTP packet. The transfer identifiers (TID's) used by
TFTP are passed to the Datagram layer to be used as ports; therefore
they must be between 0 and 65,535. The initialization of TID's is
discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates
the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the
formats of the various types of packets are discussed further in the
section on TFTP packets.

```
Sollins                                                      [Page 3]
RFC 1350                    TFTP Revision 2                 July 1992


          ----------------------------------------------------
          |  Local Medium  |  Internet  |  Datagram  |  TFTP  |
          ----------------------------------------------------
```
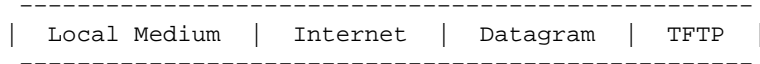
                      Figure 3−1: Order of Headers

4. Initial Connection Protocol

   A transfer is established by sending a request (WRQ to write onto a
   foreign file system, or RRQ to read from it), and receiving a
   positive reply, an acknowledgment packet for write, or the first data
   packet for read. In general an acknowledgment packet will contain
   the block number of the data packet being acknowledged. Each data
   packet has associated with it a block number; block numbers are
   consecutive and begin with one. Since the positive response to a
   write request is an acknowledgment packet, in this special case the
   block number will be zero. (Normally, since an acknowledgment packet
   is acknowledging a data packet, the acknowledgment packet will
   contain the block number of the data packet being acknowledged.) If
   the reply is an error packet, then the request has been denied.

   In order to create a connection, each end of the connection chooses a
   TID for itself, to be used for the duration of that connection. The
   TID's chosen for a connection should be randomly chosen, so that the
   probability that the same number is chosen twice in immediate
   succession is very low. Every packet has associated with it the two
   TID's of the ends of the connection, the source TID and the
   destination TID. These TID's are handed to the supporting UDP (or
   other datagram protocol) as the source and destination ports. A
   requesting host chooses its source TID as described above, and sends
   its initial request to the known TID 69 decimal (105 octal) on the
   serving host. The response to the request, under normal operation,
   uses a TID chosen by the server as its source TID and the TID chosen
   for the previous message by the requestor as its destination TID.
   The two chosen TID's are then used for the remainder of the transfer.

16. Appendix E − RFC 1350   60

As an example, the following shows the steps used to establish a
connection to write a file. Note that WRQ, ACK, and DATA are the
names of the write request, acknowledgment, and data types of packets
respectively. The appendix contains a similar example for reading a
file.

Sollins   [Page 4]
RFC 1350   TFTP Revision 2   July 1992

    1. Host A sends a "WRQ" to host B with source= A's TID,
       destination= 69.

    2. Host B sends a "ACK" (with block number= 0) to host A with
       source= B's TID, destination= A's TID.

   At this point the connection has been established and the first data
   packet can be sent by Host A with a sequence number of 1. In the
   next step, and in all succeeding steps, the hosts should make sure
   that the source TID matches the value that was agreed on in steps 1
   and 2. If a source TID does not match, the packet should be
   discarded as erroneously sent from somewhere else. An error packet
   should be sent to the source of the incorrect packet, while not
   disturbing the transfer. This can be done only if the TFTP in fact
   receives a packet with an incorrect TID. If the supporting protocols
   do not allow it, this particular error condition will not arise.

   The following example demonstrates a correct operation of the
   protocol in which the above situation can occur. Host A sends a
   request to host B. Somewhere in the network, the request packet is
   duplicated, and as a result two acknowledgments are returned to host
   A, with different TID's chosen on host B in response to the two
   requests. When the first response arrives, host A continues the
   connection. When the second response to the request arrives, it
   should be rejected, but there is no reason to terminate the first
   connection. Therefore, if different TID's are chosen for the two
   connections on host B and host A checks the source TID's of the
   messages it receives, the first connection can be maintained while
   the second is rejected by returning an error packet.

5. TFTP Packets

   TFTP supports five types of packets, all of which have been mentioned
   above:

          opcode   operation
            1   Read request (RRQ)
            2   Write request (WRQ)
            3   Data (DATA)
            4   Acknowledgment (ACK)
            5   Error (ERROR)

   The TFTP header of a packet contains the opcode associated with

that packet.

```
            2 bytes     string   1 byte     string  1 byte
            ------------------------------------------------
           | Opcode | Filename |   0  |    Mode    |   0  |
            ------------------------------------------------
```
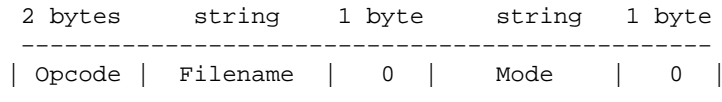
                       Figure 5-1: RRQ/WRQ packet

   RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format
   shown in Figure 5-1. The file name is a sequence of bytes in
   netascii terminated by a zero byte. The mode field contains the
   string "netascii", "octet", or "mail" (or any combination of upper
   and lower case, such as "NETASCII", NetAscii", etc.) in netascii
   indicating the three modes defined in the protocol. A host which
   receives netascii mode data must translate the data to its own
   format. Octet mode is used to transfer a file that is in the 8-bit
   format of the machine from which the file is being transferred. It
   is assumed that each type of machine has a single 8-bit format that
   is more common, and that that format is chosen. For example, on a
   DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with
   four bits of breakage. If a host receives a octet file and then
   returns it, the returned file must be identical to the original.
   Mail mode uses the name of a mail recipient in place of a file and
   must begin with a WRQ. Otherwise it is identical to netascii mode.
   The mail recipient string should be of the form "username" or
   "username@hostname". If the second form is used, it allows the
   option of mail forwarding by a relay computer.

   The discussion above assumes that both the sender and recipient are
   operating in the same mode, but there is no reason that this has to
   be the case. For example, one might build a storage server. There
   is no reason that such a machine needs to translate netascii into its
   own form of text. Rather, the sender might send files in netascii,
   but the storage server might simply store them without translation in
   8-bit format. Another such situation is a problem that currently
   exists on DEC-20 systems. Neither netascii nor octet accesses all
   the bits in a word. One might create a special mode for such a
   machine which read all the bits in a word, but in which the receiver
   stored the information in 8-bit format. When such a file is
   retrieved from the storage site, it must be restored to its original
   form to be useful, so the reverse mode must also be implemented. The
   user site will have to remember some information to achieve this. In
   both of these examples, the request packets would specify octet mode
   to the foreign host, but the local host would be in some other mode.
   No such machine or application specific modes have been specified in
   TFTP, but one would be compatible with this specification.

   It is also possible to define other modes for cooperating pairs of

16. Appendix E – RFC 1350   62

hosts, although this must be done with care. There is no requirement
that any other hosts implement these. There is no central authority
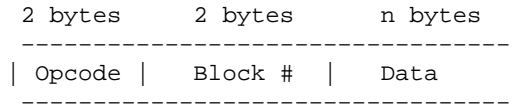that will define these modes or assign them names.

```
          2 bytes      2 bytes        n bytes
          ---------------------------------
          | Opcode |   Block #  |   Data     |
          ---------------------------------
```

          Figure 5−2: DATA packet

Data is actually transferred in DATA packets depicted in Figure 5−2.
DATA packets (opcode = 3) have a block number and data field. The
block numbers on data packets begin with one and increase by one for
each new block of data. This restriction allows the program to use a
single number to discriminate between new packets and duplicates.
The data field is from zero to 512 bytes long. If it is 512 bytes
long, the block is not the last block of data; if it is from zero to
511 bytes long, it signals the end of the transfer. (See the section
on Normal Termination for details.)

All packets other than duplicate ACK's and those used for
termination are acknowledged unless a timeout occurs [4]. Sending a
DATA packet is an acknowledgment for the first ACK packet of the
previous DATA packet. The WRQ and DATA packets are acknowledged by
ACK or ERROR packets, while RRQ

```
            2 bytes      2 bytes
            --------------------
            | Opcode |   Block # |
            --------------------
```
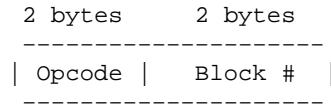
            Figure 5−3: ACK packet

and ACK packets are acknowledged by DATA or ERROR packets. Figure
5−3 depicts an ACK packet; the opcode is 4. The block number in
an ACK echoes the block number of the DATA packet being
acknowledged. A WRQ is acknowledged with an ACK packet having a
block number of zero.

```
          2 bytes     2 bytes       string    1 byte
          ----------------------------------------
```

```
| Opcode | ErrorCode |   ErrMsg   |   0   |
-----------------------------------------
```
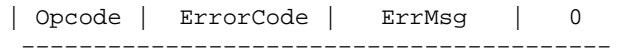
Figure 5-4: ERROR packet

An ERROR packet (opcode 5) takes the form depicted in Figure 5-4. An
ERROR packet can be the acknowledgment of any other type of packet.
The error code is an integer indicating the nature of the error. A
table of values and meanings is given in the appendix. (Note that
several error codes have been added to this version of this
document.) The error message is intended for human consumption, and
should be in netascii. Like all other strings, it is terminated with
a zero byte.

6. Normal Termination

The end of a transfer is marked by a DATA packet that contains
between 0 and 511 bytes of data (i.e., Datagram length < 516). This
packet is acknowledged by an ACK packet like all other DATA packets.
The host acknowledging the final DATA packet may terminate its side
of the connection on sending the final ACK. On the other hand,
dallying is encouraged. This means that the host sending the final
ACK will wait for a while before terminating in order to retransmit
the final ACK if it has been lost. The acknowledger will know that
the ACK has been lost if it receives the final DATA packet again.
The host sending the last DATA must retransmit it until the packet is
acknowledged or the sending host times out. If the response is an
ACK, the transmission was completed successfully. If the sender of
the data times out and is not prepared to retransmit any more, the
transfer may still have been completed successfully, after which the
acknowledger or network may have experienced a problem. It is also
possible in this case that the transfer was unsuccessful. In any
case, the connection has been closed.

7. Premature Termination

If a request can not be granted, or some error occurs during the
transfer, then an ERROR packet (opcode 5) is sent. This is only a
courtesy since it will not be retransmitted or acknowledged, so it
may never be received. Timeouts must also be used to detect errors.

Sollins   [Page 8]
RFC 1350   TFTP Revision 2   July 1992

I. Appendix

Order of Headers

                                            2 bytes

      | Local Medium | Internet | Datagram | TFTP Opcode |

```

TFTP Formats

```
    Type   Op #      Format without header

           2 bytes     string   1 byte     string   1 byte
           -------------------------------------------------
    RRQ/  | 01/02 | Filename  |  0  |    Mode    |  0  |
    WRQ    -------------------------------------------------
           2 bytes    2 bytes        n bytes
           ---------------------------------
    DATA  | 03    |  Block #  |   Data    |
           ---------------------------------
           2 bytes    2 bytes
           -------------------
    ACK   | 04    |  Block #  |
           -------------------
           2 bytes  2 bytes          string    1 byte
           ---------------------------------------
    ERROR | 05    | ErrorCode |  ErrMsg   |  0  |
           ---------------------------------------
```

Initial Connection Protocol for reading a file

1. Host A sends a "RRQ" to host B with source= A's TID,
   destination= 69.

2. Host B sends a "DATA" (with block number= 1) to host A with
   source= B's TID, destination= A's TID.

Error Codes

```
   Value     Meaning

   0         Not defined, see error message (if any).
   1         File not found.
   2         Access violation.
   3         Disk full or allocation exceeded.
   4         Illegal TFTP operation.
   5         Unknown transfer ID.
   6         File already exists.
   7         No such user.
```

Internet User Datagram Header [2]

   (This has been included only for convenience. TFTP need not be

implemented on top of the Internet User Datagram Protocol.)

   Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Values of Fields

   Source Port   Picked by originator of packet.

   Dest. Port   Picked by destination machine (69 for RRQ or WRQ).

   Length   Number of bytes in UDP packet, including UDP header.

   Checksum   Reference 2 describes rules for computing checksum.
                    (The implementor of this should be sure that the
                    correct algorithm is used here.)
                    Field contains zero if unused.

   Note: TFTP passes transfer identifiers (TID's) to the Internet User
   Datagram protocol to be used as the source and destination ports.

Sollins   [Page 10]
RFC 1350   TFTP Revision 2   July 1992

References

   [1] USA Standard Code for Information Interchange, USASI X3.4-1968.

   [2] Postel, J., "User Datagram Protocol," RFC 768, USC/Information
        Sciences Institute, 28 August 1980.

   [3] Postel, J., "Telnet Protocol Specification," RFC 764,
        USC/Information Sciences Institute, June, 1980.

   [4] Braden, R., Editor, "Requirements for Internet Hosts --
        Application and Support", RFC 1123, USC/Information Sciences
        Institute, October 1989.

Security Considerations

   Since TFTP includes no login or access control mechanisms, care must
   be taken in the rights granted to a TFTP server process so as not to
   violate the security of the server hosts file system. TFTP is often
   installed with controls such that only files that have public read
   access are available via TFTP and writing files via TFTP is
   disallowed.

```
Author's Address

   Karen R. Sollins
   Massachusetts Institute of Technology
   Laboratory for Computer Science
   545 Technology Square
   Cambridge, MA 02139−1986

   Phone: (617) 253−6006

   EMail: SOLLINS@LCS.MIT.EDU
```

17. Other Formats of this Document

This document is published in 10 different formats namely – DVI, Postscript, Latex, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages and SGML.

- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from – ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/other−formats/
- Plain text format is in: ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO Any help from you to translate to other languages is welcome.

The document is written using a tool called "SGML tool" which can be got from – http://www.xs4all.nl/~cg/sgmltools/ Compiling the source you will get the following commands like

- sgml2html databasehowto.sgml (to generate html file)
- sgml2rtf databasehowto.sgml (to generate RTF file)
- sgml2latex databasehowto.sgml (to generate latex file)

This document is located at –

- http://sunsite.unc.edu/LDP/HOWTO/Diskless−HOWTO.html

Also you can find this document at the following mirrors sites −

- http://www.caldera.com/LDP/HOWTO/Diskless−HOWTO.html
- http://www.WGS.com/LDP/HOWTO/Diskless−HOWTO.html
- http://www.cc.gatech.edu/linux/LDP/HOWTO/Diskless−HOWTO.html
- http://www.redhat.com/linux−info/ldp/HOWTO/Diskless−HOWTO.html
- Other mirror sites near you (network−address−wise) can be found at
  http://sunsite.unc.edu/LDP/hmirrors.html select a site and go to directory
  /LDP/HOWTO/Diskless−HOWTO.html

In order to view the document in dvi format, use the xdvi program. The xdvi program is located in tetex−xdvi*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Publishing | TeX menu buttons.

```
To read dvi document give the command −
        xdvi −geometry 80x90 howto.dvi
And resize the window with mouse. See man page on xdvi.
To navigate use Arrow keys, Page Up, Page Down keys, also
you can use 'f', 'd', 'u', 'c', 'l', 'r', 'p', 'n' letter
keys to move up, down, center, next page, previous page etc.
To turn off expert menu press 'x'.
```

You can read postscript file using the program 'gv' (ghostview) or 'ghostscript'. The ghostscript program is in ghostscript*.rpm package and gv program is in gv*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Graphics menu buttons. The gv program is much more user friendly than ghostscript. Also ghostscript and gv are available on other platforms like OS/2, Windows 95 and NT, you view this document even on those platforms.

```
To read postscript document give the command −
        gv howto.ps

To use ghostscript give −
        ghostscript howto.ps
```

Next PreviousContentsNextPreviousContents

2. Diskless Computer for Microsoft Windows 95/NT !!

Since Microsoft Windows 95/NT **DOES NOT** support diskless nodes, there is a intelligent work−around to overcome this short coming. Microsoft corporation will be **surprised** !!

2.1 VMWare package

Use the VMWare BIOS software with Linux which can host the Windows 95/98/NT. Linux will be the "host" OS and Windows 95/NT will be the "guest" OS. VMWare is NOT a emulator but has BIOS which allows you to install Windows 95/98/NT as the guest OS to linux. Install the VMWare on Linux server and than install Windows 95/NT on VMWare.

You can use the 'xhost' command and DISPLAY environment from **any** diskless node. See 'man xhost' on linux. At diskless node give −

```
        export DISPLAY=server_hostname:0.0
where server_hostname is the name of the server machine. And start X-terminal with
        xterm
```

Using VMWare, Diskless linux computers can run both Linux and MS Windows 95 programs. VMWare is at http://www.vmware.com.

2.2 VNC package from AT and T

You can also use the VNC (Virtual Network Computing) Technology from the telecom giant AT & T. VNC is GPLed and is a free software. Using VNC you can run Windows 95/NT programs on diskless linux computer but actually running on remote Windows95/NT server. VNC is at http://www.uk.research.att.com/vnc

3. Advantages of Diskless Computer

Diskless linux computer will become **immensely** popular and will be the product of this century and in the next century. The diskless linux computers will be very successful because of the availability of very high−speed network cards at very low prices. Today 100 Megabit per second (12.5 MB per sec transfer rate) network cards are common and in about 1 to 2 years 1000 MBit (125 MB per sec transfer rate) network cards will become very cheap and will be the standard.

In near future, Monitor manufacturers will place the CPU, NIC, RAM **right inside** the monitor to form a diskless computer!! This eliminates the diskless computer box and saves space. The monitor will have outlet for mouse, keyboard, network RJ45 and power supply.

The following are benefits of using diskless computers −

- Diskless Linux computers can run BOTH MS Windows 95/NT and linux programs.

- Total cost of ownership is very low in case of Diskless computers. Total cost of ownership is cost of initial purchasing + cost of maintainence. The cost of maintainence is usually **3 to 5 times** the cost of initial computer purchase and this cost is recurring year after year. In case of Diskless computers, the cost of maintainence is **completely eliminated**!!

- All the backups are centralized at one single main server.

- More security of data as it is located at server.

- No need of UPS battery, air−conditioning, dust proof environment for diskless clients, only server needs UPS battery, A/C and dust proof environment.

- Protection from Virus attack − Computer virus cannot attack diskless computers as they do not have any hard disk. Virus cannot do any damage to diskless computers. Only one single server box need to be protected against virus attack. This saves millions of dollars for the company by avoiding installtion of vaccines and cleaning the hard disks!!

- Server can have large powerful/high performance hard disks, can optimize the usage of disk space via sharing by many diskless computer users. Fault tolerance of hard disk failure is possible by using RAID on main server.

- Server can have 64 bit CPU SMP box having many CPUs or even linux super−computers. CPU power can be shared by many diskless computer users

- Sharing of central server RAM memory by many diskless computer users. For example, if many users are using web browser than at server RAM there will be only one copy of web browser in the RAM. In case Windows 95 PCs, many users need to have individual copy of web browser in local RAM and hence there is wastage of RAM space.

- Diskless linux computers can run programs on multiple servers using the "xhost" and DISPLAY environment.

- Very few system administrators required to maitain central server unlike Windows 95 PC clients which need many administrators.

- Zero administration at diskless client side. Diskless computers are absolutely maintainence free and troublefree.

- Long life of diskless clients − more than **100 years** without any hardware or software upgrades.

- Eliminates install/upgrade of hardware, software on diskless client side.

- Eliminates cost of cdrom, floppy, tape drive, modem, UPS battery, Printer parallel ports, serial ports etc..

- Can operate in places like factory floor where a hard disk might be too fragile.

4. Linux Terminal Server Project – LTSP

LTSP is a open source code project to build diskless linux computers.

At LTSP site you will find RPM packages for Redhat Linux and packages for Debian Linux which will save you lots of time. The subsequent chapters given in this document are for academic purposes only, which you can read them if you have more time.

Visit the LTSP and related sites at :−

- http://www.ltsp.org

- http://www.disklessworkstations.com

- http://www.slug.org.au/etherboot

- http://metalab.unc.edu/Linux/HOWTO/XFree86−Video−Timings−HOWTO.html

Related topics worth seeing −

- NCD X−terminal http://www.linuxdoc.org/HOWTO/mini/NCD−X−Terminal.html

5. EPROM Burners and Memory chips

In the following chapters you will need information about EPROM burners which are given below.

5.1 Non−Volatile Memory chips

Here is the brief descriptions of memory chips and their types.

- **PROM**: Pronounced prom, an acronym for programmable read−only memory. A PROM is a memory chip on which data can be written only once. Once a program has been written onto a PROM, it remains there forever. Unlike RAM, PROMs retain their contents when the computer is turned off. The difference between a PROM and a ROM (read−only memory) is that a PROM is manufactured as blank memory, whereas a ROM is programmed during the manufacturing process. To write data onto a PROM chip, you need a special device called a PROM programmer or PROM burner. The process of programming a PROM is sometimes called burning the PROM. An EPROM (erasable programmable read−only memory) is a special type of PROM that can be erased by exposing it to ultraviolet light. Once it is erased, it can be reprogrammed. An EEPROM is similar to a PROM, but requires only electricity to be erased.
- **EPROM**: Acronym for erasable programmable read−only memory, and pronounced e−prom, EPROM is a special type of memory that retains its contents until it is exposed to ultraviolet light. The ultraviolet light clears its contents, making it possible to reprogram the memory. To write to and erase an EPROM, you need a special device called a PROM programmer or PROM burner. An

EPROM differs from a PROM in that a PROM can be written to only once and cannot be erased. EPROMs are used widely in personal computers because they enable the manufacturer to change the contents of the PROM before the computer is actually shipped. This means that bugs can be removed and new versions installed shortly before delivery. A note on EPROM technology: The bits of an EPROM are programmed by injecting electrons with an elevated voltage into the floating gate of a field−effect transistor where a 0 bit is desired. The electrons trapped there cause that transistor to conduct, reading as 0. To erase the EPROM, the trapped electrons are given enough energy to escape the floating gate by bombarding the chip with ultraviolet radiation through the quartz window. To prevent slow erasure over a period of years from sunlight and fluorescent lights, this quartz window is covered with an opaque label in normal use.

- **EEPROM**: Acronym for electrically erasable programmable read−only memory. Pronounced double−e−prom or e−e−prom, an EEPROM is a special type of PROM that can be erased by exposing it to an electrical charge. Like other types of PROM, EEPROM retains its contents even when the power is turned off. Also like other types of ROM, EEPROM is not as fast as RAM. EEPROM is similar to flash memory (sometimes called flash EEPROM). The principal difference is that EEPROM requires data to be written or erased one byte at a time whereas flash memory allows data to be written or erased in blocks. This makes flash memory faster.
- **FRAM**: Short for Ferroelectric Random Access Memory, a type of non−volatile memory developed by Ramtron International Corporation. FRAM combines the access speed of DRAM and SRAM with the non−volatility of ROM. Because of its high speed, it is replacing EEPROM in many devices. The term FRAM itself is a trademark of Ramtron.
- **NVRAM**: Abbreviation of Non−Volatile Random Access Memory, a type of memory that retains its contents when power is turned off. One type of NVRAM is SRAM that is made non−volatile by connecting it to a constant power source such as a battery. Another type of NVRAM uses EEPROM chips to save its contents when power is turned off. In this case, NVRAM is composed of a combination of SRAM and EEPROM chips.
- **Bubble Memory**: A type of non−volatile memory composed of a thin layer of material that can be easily magnetized in only one direction. When a magnetic field is applied to circular area of this substance that is not magnetized in the same direction, the area is reduced to a smaller circle, or bubble. It was once widely believed that bubble memory would become one of the leading memory technologies, but these promises have not been fulfilled. Other non−volatile memory types, such as EEPROM, are both faster and less expensive than bubble memory.
- **Flash Memory**: A special type of EEPROM that can be erased and reprogrammed in blocks instead of one byte at a time. Many modern PCs have their BIOS stored on a flash memory chip so that it can easily be updated if necessary. Such a BIOS is sometimes called a flash BIOS. Flash memory is also popular in modems because it enables the modem manufacturer to support new protocols as they become standardized.

5.2 List of EEPROM Burner manufacturers

For a list of **EPROM burner manufacturers** visit the Yahoo site and go to economy−>company−>Hardware−>Peripherals−>Device programmers.

- Yahoo URL for EPROMs is at http://dir.yahoo.com/Business_and_Economy/Companies/Computers/Hardware/Peripherals/Device_Programm

- Advanced Research Technology B.V − development, production and sales of electronic programmer equipment; development of hardware and software.
- Advin Systems Inc. − PC−based device programmers that support the latest in package types and

device technologies.

- Andromeda Research Labs – manufactures a portable eprom and device programming system.
- B and C Microsystems, Inc – offers test and duplication/programming equipment for PCMCIA (PC) Cards, ISA/PCI Cards, SIMMs, Memory Devices (including FLASH), PLDs.
- BP Microsystems – Device Programmers.
- Bytek – designs, develops, manufactures and markets micro−processor−based, modular electronic systems used to program and test semiconductor devices. Product line includes the ChipBurner.
- Concentrated Programming Ltd – offers a full range of device programming solutions.
- Dataman Programmmers Ltd. – manufacture of hand−help EPROM programmer/emulator. Also sell PC−based programmers, and Gang−Pro programmers.
- General Device Instruments – IC Device programmers. Universal and Gang programmers for Pld, Flash, microcontrollers, Proms, EEproms, Memory, Epld, Mach and many other ic devices.
- HI−LO System Research Co., Ltd. – manufacturer of universal and gang device programmers.
- ICE Technology – EPROM and universal device programmers which support memories, microcontrollers, and programmable logic devices.
- Iceprom – in−circuit erasable programmable read−only memory.
- Incept Ltd.
- International Microsystems Inc – High speed reliable gang programmer. (PROM, FLASH, Microcontroller, PCMCIA memory card).
- JED Microprocessors Pty. Ltd. – plugs into a PC printer port D25 connector, and programs any 28−pin or 32−pin EPROM and FLASH device.
- Logical Devices, Inc – device programming for PLDs, FPGAs, PROMs, microcontrollers. Producers of CUPL compiler for programmable logic and the ALLPRO and Chipmaster device programmer.
- MCL Systems – new method not only for programming but also for developing your new hardware with Integrated Controller Unit. And you don't need to be an expert.
- MQP Electronics – manufacturer of universal device programmers, gang programmers, production software, and package converters. High thoughput and reliability.
- Needham's Electronics – manufacturer of device programmers.
- NP Programming Services – provides programming for memory and logic parts.
- Program Automation, Inc. – independent service company specializing in high volume PROM programming, including flash I/Cs.
- Stag Programmers Inc – manufacturer of prom and logic programmers, production handling equipment and UV erasers.
- Sunrise Electronics – universal device programmers, gang and in−circuit programmers with life time support.
- System General Co. – Device Programmer, EPROM Writer and IC Tester
- Tribal Microsystems – universal and gang device programmers, 8051 and EPROM emulators, test and burn−in sockets and production sockets.
- Universal Device Programmers

6. Introduction to Network Booting and Etherboot

This chapter is written by Ken Yap ken.yap@acm.org and explains how to bootstrap your computer from a program stored in non−volatile memory without accessing your hard disk. It is an ideal technique for maintaining and configuring a farm of linux boxes.

6.1 What is Network booting?

Network booting is an old idea. The central idea is that the computer has some bootstrap code in non−volatile memory, e.g. a ROM chip, that will allow it to contact a server and obtain system files over a network link.

6.2 How does it work

In order to boot over the network, the computer must get

1. an identity
2. an operating system image and
3. usually, a working filesystem.

Consider a diskless computer (DC) that has a network boot ROM. It may be one of several identical DCs. How can we distinguish this computer from others? There is one piece of information that is unique to that computer (actually its network adapter) and that is its Ethernet address. Every Ethernet adapter in the world has a unique 48 bit Ethernet address because every Ethernet hardware manufacturer has been assigned blocks of addresses. By convention these addresses are written as hex digits with colons separating each group of two digits, for example – **00:60:08:C7:A3:D8** .

The protocols used for obtaining an IP address, given an Ethernet address, are called **Boot Protocol (BOOTP)** and **Dynamic Host Configuration Protocol (DHCP)**. DHCP is an evolution of BOOTP. In our discussion, unless otherwise stated, anything that applies to BOOTP also applies to DHCP. (Actually it's a small lie that BOOTP and DHCP only translate Ethernet addresses. In their foresight, the designers made provision for BOOTP and DHCP to work with any kind of hardware address. But Ethernet is what most people will be using.)

An example of a BOOTP exchange goes like this:

DC: Hello, my hardware address is **00:60:08:C7:A3:D8**, please give me my IP address.

BOOTP server: (Looks up address in database.) Your name is aldebaran, your IP address is 192.168.1.100, your server is 192.168.1.1, the file you are supposed to boot from is /tftpboot/vmlinux.nb (and a few other pieces of information).

You may wonder how the DC found the address of the BOOTP server in the first place. The answer is that it didn't. The BOOTP request was broadcast on the local network and any BOOTP server that can answer the request will.

After obtaining an IP address, the DC must download an operating system image and execute it. Another

Internet protocol is used here, called **Trivial File Transfer Protocol (TFTP)**. TFTP is like a cut−down version of FTP−−−there is no authentication, and it runs over User Datagram Protocol (UDP) instead of Transmission Control Protocol (TCP). UDP was chosen instead of TCP for simplicity. The implementation of UDP on the DC can be small so the code is easy to fit on a ROM. Because UDP is a block oriented, as opposed to a stream oriented, protocol, the transfer goes block by block, like this:

```
DC: Give me block 1 of /tftpboot/vmlinux.nb.
TFTP server: Here it is.
DC: Give me block 2.
```

and so on, until the whole file is transferred. Handshaking is a simply acknowledge each block scheme, and packet loss is handled by retransmit on timeout. When all blocks have been received, the network boot ROM hands control to the operating system image at the entry point.

Finally, in order to run an operating system, a root filesystem must be provided. The protocol used by Linux and other Unixes is normally **Network File System (NFS)**, although other choices are possible. In this case the code does not have to reside in the ROM but can be part of the operating system we just downloaded. However the operating system must be capable of running with a root filesystem that is a NFS, instead of a real disk. Linux has the required configuration variables to build a version that can do so.

6.3 Netbooting in Practice

Net Loader is a small program that runs as a BIOS extension, usually on an EPROM on the NIC. It handles the BOOTP query and TFTP loading and then transfers control to the loaded image. It uses TCP/IP protocols but the loaded image doesn't have to be Linux. The loaded image can be anything, even DOS. They can also be loaded from a floppy for testing and for temporary setups.

Besides commercial boot ROMs, there are **TWO** sources for free packages for network booting. Free implementations of TCP/IP net loaders are −

1. **ETHERBOOT**http://www.slug.org.au/etherboot/ and
2. **NETBOOT**http://www.han.de/~gero/netboot.html

Etherboot uses built−in drivers while Netboot uses Packet drivers. First you have to ascertain that your network card is supported by Etherboot or Netboot. Eventually you have to find a person who is willing to put the code on an EPROM (Erasable Programmable Read Only Memory) for you but in the beginning you can do **network booting from a floppy**.

To create a boot floppy, a special boot block is provided in the distribution. This small 512 byte program loads the disk blocks following it on the floppy into memory and starts execution. Thus to make a boot floppy, one has only to concatenate the boot block with the Etherboot binary containing the driver for one's network card like this:

```
        # cat floppyload.bin 3c509.lzrom > /dev/fd0
```

Get the nfsboot package (the package is available from your favourite linux mirror site in the /pub/Linux/system/Linux−boot directory). It contains a booteprom image for the network cards (like wd8013) which can be directly burned in. See also the LTSP site at http://www.ltsp.org

Before you put in the network boot floppy, you have to set up three services on Linux −

> 1. BOOTP (or DHCP)
> 2. TFTP and
> 3. NFS.

You don't have to set up all three at once, you can do them step by step, making sure each step works before going on to the next.

Bootp

Install Bootp. See bootp*.rpm on Redhat linux cdrom. See also LTSP site for RPM packages at http://www.ltsp.org. See also unix manual pages 'man 5 bootptab', 'man 8 bootpd', 'man 8 bootpef', 'man 8 bootptest'. You then have to ensure that this server is waiting for bootp requests. The daemon can be run either directly by issuing command

```
        bootpd −s
```

Or by using inetd edit the file /etc/inetd.conf and put a line like this:

```
        bootps dgram    udp     wait    root    /usr/sbin/in.bootpd     bootpd
```

Insert or uncomment the following two lines in /etc/services:

```
bootps          67/tcp          # BOOTP server
tftp            69/udp          # TFTP server
```

If you had to modify /etc/inetd.conf, then you need to restart inetd by sending the process a HUP signal.

```
        kill −HUP <process id of inetd>.
```

Next, you need to give bootp a database to map Ethernet addresses to IP addresses. This database is in /etc/bootptab. You must modify it by inserting the IP addresses of your gateway, dns server, and the ethernet address(es) of your diskless machine(s). It contains lines of the following form:

```
        aldebaran.foo.com:ha=006008C7A3D8:ip=192.168.1.100:bf=/tftpboot/vmlinuz.nb
```

Other information can be specified but we will start simple.

Another example of /etc/bootptab is :

```
global.prof:\
        :sm=255.255.255.0:\
        :ds=192.168.1.5:\
        :gw=192.168.1.19:\
        :ht=ethernet:\
        :bf=linux:
machine1:hd=/export/root/machine1:tc=global.prof:ha=0000c0863d7a:ip=192.168.1.140:
machine2:hd=/export/root/machine2:tc=global.prof:ha=0800110244e1:ip=192.168.1.141:
machine3:hd=/export/root/machine3:tc=global.prof:ha=0800110244de:ip=192.168.1.142:
```

global.prof is a general template for host entries, where

- sm field contains the subnet mask
- ds field contains the address of the Domain Name Server
- gw field contains the default gateway address
- ht field contains the lan media hardware type
- bf field contains the name of the boot file

After this, every machine must have a line:

- the first field contains the host name,
- hd field contains the directory of the bootfile,
- the global template can be included with the tc field,
- ha field contains the hardvare address of the ethernet card,
- ip field contains the assigned ip address.

Now boot the DC with the floppy and it should detect your Ethernet card and broadcast a BOOTP request. If all goes well, the server should respond to the DC with the information required. Since /tftpboot/vmlinux.nb doesn't exist yet, it will fail when it tries to load the file. Now you need to compile a special kernel, one that has the option for mounting the root filesystem from NFS turned on. You also need to enable the option to get the IP address of the kernel from the original BOOTP reply. You also need to compile the Linux driver for your network adapter into the kernel instead of loading it as a module. It is possible to download an initial ramdisk so that module loading works but this is something you can do later.

You cannot install the zImage resulting from the kernel compilation directly. It has to be turned into a tagged image. A tagged image is a normal kernel image with a special header that tells the network bootloader where the bytes go in memory and at what address to start the program. You use a program called mknbi−linux to create this tagged image. This utility can be found in the Etherboot distribution. After you have generated the image, put it in the /tftpboot directory under the name specified in /etc/bootptab. Make sure to make this file world readable because the tftp server does not have special privileges.

Tftp

For TFTP, see tftp*.rpm on Redhat Linux cdrom. TFTP (Trivial File Transfer Protocol) is a file transfer protocol, such as ftp, but it's much simpler to help coding it in EPROMs. TFTP can be used in two ways:

- **Simple tftp:** means that the client can acces to your whole file system. It's simpler but it's a big security hole (anyone can get your password file via tftp).
- **Secure tftp:** the tftp server uses a chroot.2 system call to change it's own root directory. Anything outside the new root directory will be completely inaccessible. Because of the chroot dir becomes the new root dir, the hd filed in the bootptab must reflect the new situation. For example: when using insecure tftp, the hd field contains the full path to the boot directory: /export/root/machine1. When using secure tftp whith /export as root dir, then /export becomes / and the hd field must be /root/machine1.

Tftpd is normally started up from inetd with a line like this in /etc/inetd.conf.

```
tftp dgram udp wait root /usr/sbin/tcpd in.tftpd −s /tftpboot
#tftp   dgram   udp     wait    root    /usr/sbin/in.tftpd      tftpd /export
```

Again, restart inetd with a HUP signal and you can retry the boot and this time it should download the kernel image and start it. You will find that the boot will continue until the point where it tries to mount a root filesystem. At this point you must configure and export NFS partitions to proceed.

NFS root filesystem

For various reasons, it's not a good idea to use the root filesystem of the server as the root filesystem of the DCs. One is simply that there are various configuration files there and the DC will get the wrong information that way. Another is security. It's dangerous to allow write access (and write access is needed for the root filesystem, for various reasons) to your server's root. However the good news is that a root filesystem for the DC is not very large, only about 30 MB and a lot of this can be shared between multiple DCs.

Ideally, to construct a root filesystem, you have to know what files your operating system distribution is expecting to see there. Critical to booting are device files, files in /sbin and /etc. You can bypass a lot of the hard work by making a copy of an existing root filesystem and modifying some files for the DC. In the Etherboot distribution, there is a tutorial and links to a couple of shell scripts that will create such a DC root filesystem from an existing server root filesystem. There are also troubleshooting tips in the Etherboot documentation as this is often the trickiest part of the setup.

The customised Linux kernel for the DC expects to see the root filesystem at /tftpboot/(IP address of the DC), for example: /tftpboot/192.168.1.100 in the case above. This can be changed when configuring the kernel, if desired.

Now create or edit /etc/exports (see 'man 5 exports' and 'man 8 exportfs') on the server and put in a line of the following form:

```
/tftpboot/192.168.1.100 aldebaran.foo.com(rw,no_root_squash)
```

The rw access is needed for various system services. The no_root_squash attribute prevents the NFS system from mapping root's ID to another one. If this is not specified, then various daemons and loggers will be unhappy.

Start or restart the NFS services (rpc.portmap and rpc.mountd) and retry the diskless boot. If you are

successful, the kernel should be able to mount a root filesystem and boot all the way to a login prompt. Most likely, you will find several things misconfigured. Most Linux distributions are oriented towards disked operation and require a little modification to suit diskless booting. The most common failing is reliance on files under /usr during the boot process, which is normally imported from a server late in the boot process. Two possible solutions are −

1. Provide the few required files under a small /usr directory on the root filesystem, which will then be overlaid when /usr is imported, and

2. Modify the paths to look for the files in the root filesystem. The files to edit are under /tftpboot/192.168.1.100 (remember, this is the root directory of the DC).

You may wish to mount other directories from the server, such as /usr (which can be exported read−only).

Burn EPROM

When you are satisfied that you can boot over the network without any problems, you may wish to put the code on an EPROM.

6.4 Uses of Network booting

X−terminals are one natural use of network booting. The lack of a disk in the terminal makes it quieter and contributes to a pleasant working environment. The machine should ideally have 16MB of memory or more and the best video card you can find for it. This is an ideal use for a high−end 486 or low−end Pentium that has been obsoleted by hardware advances. Other people have used network booting for clusters of machines where the usage is light on the DC and does not warrant a disk, e.g. a cluster of classroom machines.

6.5 For more information

Your first stop should be the Etherboot home page: http://www.slug.org.au/etherboot/

There you will find links to other resources, including a mailing list you can subscribe to, where problems and solutions are discussed.

Related documents

- NFS−root Mini Howto at /usr/doc/HOWTO/mini or on Linux cdrom.
- Linux Networking−HOWTO by Terry Dawson, at /usr/doc/HOWTO or on linux cdrom 94004531@postoffice.csu.edu.au
- NET−3−Howto at /usr/doc/HOWTO or on Linux cdrom.
- /usr/src/linux/README about configuring and compiling new kernels

7. Redhat Linux configuration

The DC requests to mount /tftpboot/< *IP address of DC* > (in Linux Kernel 2.1 and above it is − /tftpboot/< *name of DC in bootptab* >) as its root directory '/' by NFS from server. You must export this from the server (rw, no_root_squash) because the DC wants to write on it (log files, etc).

The root directory / must contain /sbin, /bin, /lib, /etc, /var, /tmp, /root, /dev and /proc.

/sbin, /bin, /lib can be a copy of an existing Redhat Linux system. They can be shared between all DCs. But hard links only. By the way, don't link to server originals.

/etc, /var and /dev should be non−sharable copies. Customise /etc/sysconfig/network, /etc/sysconfig/network−scripts/ifcfg−eth0, /etc/fstab, /etc/conf.modules, and others. Turn off all network services you don't need. Remove all stuff you don't need from /var, e.g. RPM db, lpd files.

/root and /proc should just exist. /tmp should exist and be mode 1777.

You probably want to create /usr and /home mount points. /usr can be mounted ro (read−only).

About 10 MB per DC plus about 15 MB of shared files should be sufficient. By the way, if your DCs are quite similar, the kernel image can also be shared.

Here is an illustrative script to create the first root filesystem.

```
#!/bin/sh
if [ $# != 1 ]
then
        echo Usage: $0 client-IP-addr
        exit 1
fi

cd /

umask 022

mkdir -p /tftpboot/$1

# just make these ones
for d in home mnt proc tmp usr
do
        mkdir /tftpboot/$1/$d
        done

        chmod 1777 /tftpboot/$1/tmp

        touch /tftpboot/$1/fastboot
        chattr +i /tftpboot/$1/fastboot

        # copy these ones
        cp -a bin lib sbin dev etc root var /tftpboot/$1

cat <<EOF
Now, in /tftpboot/$1/etc, edit
```

```
              sysconfig/network
              sysconfig/network-scripts/ifcfg-eth0
              fstab
              conf.modules

and configure

              rc.d/rc3.d
EOF
```

Here is an illustrative script to duplicate the root filesystem

```
#!/bin/sh
if [ $# != 2 ]
then
        echo Usage: $0 olddir newdir
        exit 1
fi

cd /tftpboot

if [ ! -d $1 ]
then
        echo $1 is not a directory
        exit 1
fi

umask 022

mkdir -p $2

# just make these ones
for d in home mnt proc tmp usr
do
        mkdir $2/$d
done

chmod 1777 $2/tmp

touch $2/fastboot
chattr +i $2/fastboot

# link these ones
for d in bin lib sbin
do
        (cd $1; find $d -print | cpio -pl ../$2)
done

# copy these ones
for d in dev etc root var
do
        cp -a $1/$d $2
done

cat <<EOF
Now, in /tftpboot/$2/etc, edit

        sysconfig/network
        sysconfig/network-scripts/ifcfg-eth0
        fstab (maybe)
```

```
        conf.modules (maybe)
```

and configure

```
        rc.d/rc3.d
```
EOF

7.1 X−terminal

On the server, make sure the DC is matched by a clause in /etc/X11/xdm/Xaccess and comment out the :0 in /etc/X11/xdm/Xservers. Then make sure that xdm is run from the init scripts.

On the client, run X −query server

You will get the xdm login box and then all your X clients will run on the server.

For other applications use − you could use diskless technique for netboot routers, print servers (but should not be spooling print server), standalone apps, etc.

8. LanWorks BootWare PROMs

This information may save you time. In order to make LanWorks BootWare(tm) PROMs to correctly start up a Linux kernel image, the "bootsector" part of the image must be modified so as to enable the boot prom to jump right into the image start address. The net−bootable image format created by netboot/etherboot's `mknbi−linux' tool differs and will not run if used with BootWare PROMs.

A modified bootsector together with a Makefile to create a BootWare−bootable image after kernel compilation can be found at −

- Bwimage package ftp://ftp.ipp.mpg.de/pub/ipp/wls/linux/bwimage−0.1.tgz
- See also http://www.patoche.org/LTT/net/00000096.html
- LanWorks BootWare Boot ROMs http://www.3com.com/lanworks

Refer to the README file for installation details. Currently, only "zImage"−type kernels are supported. Unfortunately, kernel parameters are ignored.

This section courtesy of Jochen Kmietsch email to − jochen.kmietsch@tu−clausthal.de for any questions.

9. Etherboot

Etherboot is a package for creating ROM images that can download code over the network to be executed on an x86 computer. Typically the computer is diskless and the code is Linux, but these are not the only possibilities.

This document are at the Etherboot Home Page. This document explains how to install, configure and use the Etherboot package.