YAGI

(yet another ghost installer)

Image deployment in a local area network

DaNiel Ettle
FH-Regensburg
Informatikstudent

11.September 2002

Contents

1 Introduction

In most daily things we can't imagine to live without computers. In some cases it is impossible to keep companies running without PCs. So there is a great market for PC manufacturers to satisfy the needs of firms and private persons. Also nowadays all Computers have pre-installed Operating Systems on their hard drive. So all manufacturers are searching for a Toolkit to install their new PCs in a fast, reliable, cheap and independent way.
On the other hand are System Administrators who have to install and run machines in their office.
So it is a common problem faced by IT managers to ensure that client systems in their enterprises can boot from appropriate software images using appropriate configuration parameters. These selected boot images and configuration parameters must be acquired from selected servers in the enterprise as dictated by the needs of the particular environment, the capabilities or mission of the user, the resources available within the client, etc. Furthermore, these clients should boot consistently and in an inter operable manner regardless of the sources or vendors of the software and the hardware of both client and server machines.

2 Getting involved

To get an Operating system on the hard drive you can install it from CD or other mediums to keep it running. This is the standard way. In case you have many computers with identical hardware platform you can choose two other ways to install computers in a faster and cheaper way.
In the following it will be explained what the difference between a Backup and a Cloning or Disk imaging is.

2.1 Backup

Backing up your important data is as old as computers exist. The Backup Media have changed from one Medium to another and back again like magnetic tape or Disk operated devices. The advantage this type is that you need only to save the data you have produced or input in the computer like documents or source code. This will save space on your backup media and the amount of time, to save only modified data, is equally decreasing. Another way to save time and backup mediums is to save only files you have modified since the last backup you have taken. This strategy type is called 'incremental' Backup. So all data you generated are safe on disk, or whatever. The Operating system or the programs are on other Media. The disadvantage is that the Operating System and all programs have to be installed separately in a time wasting process, if the system has crashed. During the installation of the Operating System or the programs the user only has to follow some simple instructions to copy the files to the hard disk. Also there must be done partitioning, formating, some Path variables have to be modified and serials for some programs must be provided. During this install period the Person who installs the machine is idle unless he is waiting for the next interaction. In the day-by-day use of PC's, backups are generally used to save documents or/and source code. In this area backup has the best performance/cost/time benefit and is preferable compared to Cloning.

2.2 Cloning

Cloning Disks is another way to 'back up' Computers. Cloning a Hard drive is nothing else than to make a complete backup of your Hard disk. Cloning does a complete bit-by-bit transfer from your hard drive to another one, or an image file. The Operating system, the Partitioning table, the Bootsector, all Programs and your personal stuff will be transfered, and/or packed and stored.

Some Backup Programs allow to save the boot-record to Tape or something like that and restore it to another Disk. This is the Idea of Cloning a Disc. The advantage of this method is that you need not to install the Operating System, Programs or all the other stuff you generally are used to work with, on your computer.
In most cases you can boot the recovery tool from floppy or from CD. After starting the restore mode all data will be copied to your computer and there is no User Interaction necessary. However, in many cases where exists an advantage there also exists a disadvantage. In this case it is the amount of space you need to store the data of your hard disk. This size depends on how good the compression algorithm in the backup tool is. Generally it will be compressed to half of the size it will take on the Hard-Drive, but this is heavily dependent on the data. Compared to Backups, especially incremental backups, Cloning needs a lot more disk-space. Cloning computers is useful if you are an Administrator of a PC Pool at a University or a great Company. Most of the computers have identical hardware and are connected via LAN. So if a student or a colleague crashes the computer you can setup it in a short time. In almost the same manner it's possible to install a computer room for a workshop with minium effort and time.

2.3 Overview

So what to do? You can decide which form of installing computers or restoring fits better to your needs. Both forms have advantages or disadvantages. Maybe the table will help you to choose the best one for you.

Overview Backup/Cloning		
action	backup	cloning
size of image/data	small	big
time to backup and restore	depends on the amount of data and the backup medium. heavily fluctuating. [1min to several hours]	depends on the disksize and the amount of data stored on it. [30min - 120min]
interaction	choosing files to restore, possibly changing backup medium	only at start
hardware independent (IDE, SCSI, IEEE)	no (in special cases difficult)	no
file system	depended	independent
restoring data over network	only some parts	yes
remote control	maybe possible	yes

3 Netboot basics

Booting from Ethernet in an IP based LAN was announced in RFC 951 from Bill Croft (Stanford University), and John Gilmore (Sun Microsystems) in September 1985.
The 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 first idea of bootstrap loading over tftp is described in RFC 906 by Ross Finlayson (Stanford University) in, June 1984. So far, BOOTP is the deprecated way to boot clients over Ethernet.

3.1 Boot Prom

The Hardware requirements for booting over Ethernet are that your Network Card supports that feature. Most network cards come with a blank (E)EPROM socket even though it is seldom used. When it is used, it is typically filled with a proprietary EPROM[figure 1] from the network card manufacturer. You can put an Etherboot EPROM there instead.
If you are familiar with electronics construction, an alternative is to use an EEPROM card. There is a schematic and PCB artwork for such a card at the web site where you got the Etherboot distribution. This EEPROM card plugs onto the ISA bus and can be reprogrammed with software.

Figure 1: standard EPROM for an NIC

Some high-end network cards, for example the 3Com 905B, have a socket for an EEPROM which can be programmed in situ with the right utilities. See any release notes accompanying Etherboot for more information.

Old Cards, I mean really old cards that you can only find in museums, don't have a Socket for EPROM or EEPROMS. If your NIC has one, you are lucky and can flash a bootprom for your Card. All information you need and getting the right hardware and software you will find at the etherboot[etherboot.sourceforge.net] page.
If you have bought a new Ethernet Card maybe the card supports PXE[2], which is explained in the next section, so you don't have to burn an EPROM. Thats the easiest way, so choose a Network Card with PXE support if you buy a new one.

3.2 PXE

The PXEPreboot eXecution Environment) is part of an Intel initiative called Wired for Management.
PXE was born to support PC's mainly as fat clients, enabling the installation of the OS from the network. Also, PXE is used here as a way to boot the diskless thin client.
PXE embodies three technologies that will establish a common and consistent set of preboot services within the boot firmware of Intel Architecture systems:

- A uniform protocol for the client to request the allocation of a network address and subsequently request the download of a Network Bootstrap Program (NBP) from a network boot server.
- A set of APIs available in the machine's pre-boot firmware environment that constitutes a consistent set of services that can be employed by the NBP or the BIOS.
- A standard method of initiating the pre-boot firmware to execute the PXE protocol on the client machine.

The API[figure 2] services provided by PXE for use by the BIOS or NBP are:

- **Preboot Services API.** Contains several global control and information functions.
- **Trivial File Transport Protocol (TFTP) API.** Enables opening and closing of TFTP connections, and reading packets from and writing packets to a TFTP connection.
- **User Datagram Protocol (UDP) API.** Enables opening and closing UDP connections, and reading packets from and writing packets to a UDP connection.
- **Universal Network Driver Interface (UNDI) API.** Enables basic control of and I/O through the client's network interface device. This allows the use of universal protocol drivers such that the same universal driver can be used on any network interface that implements this API.

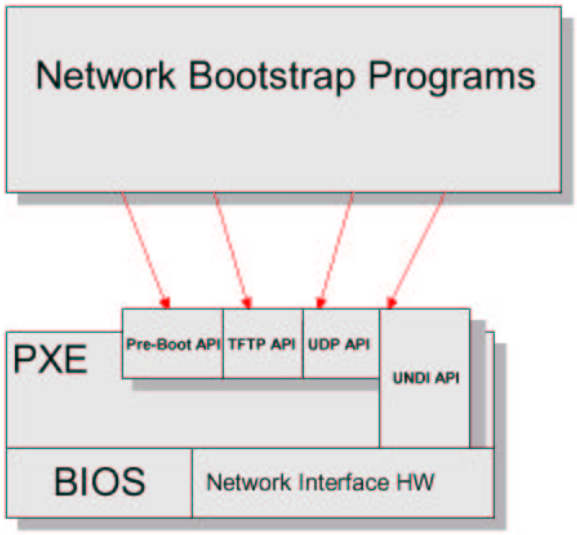

Figure 2: PXE API

Here is a short overview[figure 3] of the protocols and services which are running before and after the Remote Boot.

4 Pre-Boot-Loader

First, a bootloader is not an OS. It is rather more a very small part of an OS. Let's see how the cold boot works. After the BIOS loads sector zero (CHS=0:0:1) of the boot drive to address 0000:7C00h it checks the loaded sector for the magic bootstrap signature bytes: 55h at offset 510 and 0AAh at offset 511. (Many BIOSes will load and execute sector zero, regardless of the value of these bytes.) The CPU register DL is set to the boot drive number: 0 for floppy drive A, or 80h for hard drive C. BIOS jumps to sector 0 code it just loaded. The Bootloader is now going to work. This is a small program that can show a dialog of different config's which you can choose from. In some cases the Bootloader supports some more features like partitioning, formating and some small network capabilities. After choosing an option the Loader executes the commands to load the kernel.

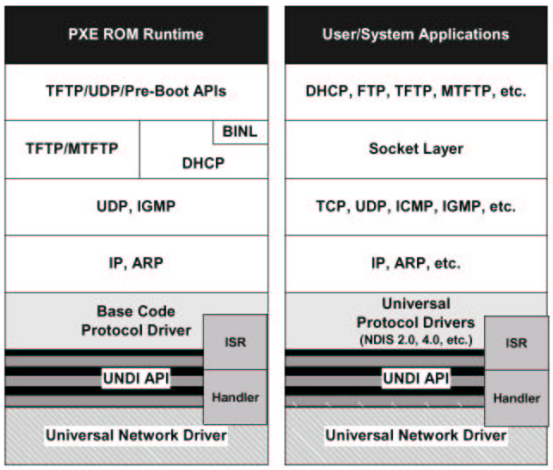

Figure 3: PXE Stack-Before and After Remote Boot

4.1 bpbatch

BpBatch is a versatile remote-boot processor, that can be downloaded for free from the Web. It can perform a large variety of actions on a computer at boot-time, before any operating system operation has started. Actions performed by BpBatch ranges from partitioning hard disks to authenticating users, including a graphical interface. The main feature of BpBatch is the partition cloning facility, which let's you create an image of a computer's hard disk partition and then distribute and install this image on to a cluster of PC.

4.1.1 config

Look at the dhcpd.conf, in the appendix[listing 15], to enable the BpBatch.
The minimum entry into the bpbatch-config file should look like this:

Listing 1: bpbatch config-file

```
Set CacheNever="ON"
linuxboot "kernel-2.4.19" "root=/dev/ram init=/linuxrc" "initrd-2.4.19.gz"
```

The CacheNever option means that no data can be written to an swap-device on the client side. All data have to be processed immediately.

4.1.2 result

The command list doesn't let any wish unfulfilled. Nevertheless it supports only FAT, FAT32 and ext2. This means that only Systems can be installed that one of this Filesystems support. If they don't run on this Filesystems they can't be installed. Our requirements are to be file-system independent, so this solution can not be chosen. Sadly because of all the nice features and the look of the install-menu. Maybe in the future more file systems will be supported and we can make a compromise. So far we have to search another pre-boot loader which fits our needs.

4.2 GRUB

GNU GRUB is a Multiboot loader. It was derived from Erich Boleyn's GRand Unified Bootloader. It is an attempt to produce a bootloader that has both the capability to be friendly to beginning or otherwise non- technically interested users and the flexibility to

help experts in diverse environments. It is compatible with Multiboot kernels (such as GNU Mach and the chaos kernel storm), Free/Net/OpenBSD, and Linux. It supports all other kernels via chain loading. It has a menu interface and a powerful BASH-like command-line interface. Recent developments in GRUB include the ability to boot from the network (by incorporating Etherboot's network drivers and TFTP support), support for ReiserFS (in addition to the support for BSD FFS, Linux ext2fs, and DOS FAT) as well as a scripting language in early development stages.

4.2.1 installing

To use grub via network booting you have to set the configure line options. At first you have to activate your NIC like –enable-eepro100 for enabling Etherexpress Pro/100 driver. In any configuration the line option –enable-diskless have to be set for creating the pxegrub Bootstrap Loader binary. This binary we copy to our tftpd-root and choose this in the dhcpd.conf as filename. At least we have to create a so called 'menu.lst' for grub. This can look like this one:

Listing 2: grub config file

```
# give me 10 seconds to interrupt the boot and change things
timeout 10
# boot from the first image by default
default 0
# if something goes wrong fallback to first entry - entry <n>
fallback 0
# make me look good
color red/blue blue/cyan


### YAGI
# set the title that is displayed in the menu
title YAGI
# the root file system is a network disk (nd)
root (nd)
# path to kernel and parameters.
kernel /bzImage-2.4.19 root=/dev/nfs nfsroot=192.168.1.2:/YAGI/
    linuxroot devfs=mount vga=extended
boot

### local system on hard drive
# set the title that is displayed in the menu
title local disk (hd0)
root        (hd0,0)
makeactive
chainloader +1
```

4.2.2 result

After all, grub is a nice tool, but it has a great disservice. For booting over network it has to support the network card and to compile this statically into the kernel. That's in most cases harmless but for new cards it would take a long time if GRUB supported the new version. It's more probably that the kernel supports the new card faster than the bootloader. Maybe in the future we can use GRUB as Loader. The menu is really nice.

4.3 PXELINUX

PXELINUX is a SYSLINUX derivative, for booting Linux off a network server, using a network ROM conforming to the Intel PXE (Pre-Execution Environment) specification. PXELINUX works with every version of PXE/NIC and combinations of which, I have tested. PXELINUX is like pxegrub, it supports a small colour-menu but no features to format or partition hard disks. Partitioning and Formatting would be a nice feature but in our case it's not needed. PXELINUX can be found where you find SYSLINUX.

4.3.1 installing

There is nothing special to do. Just download the latest SYSLINUX tar.gz and extract it. There you find the following files `pxelinux.0, pxeloader`. Copy these files to your tftp directory. Create a directory named `pxelinux.cfg` and a file named `C0A80` for a complete Class C-Network.
This is a queer thing. I try to explain how the name 'C0A80' will accomplish.
First, pxelinux will search for the config file using its own IP address in upper case hexadecimal, e.g. 192.0.2.91 ⇒ C000025B (you can use the included program "gethostip" to compute the hexadecimal IP address for any host.) If that file is not found, it will remove one hex digit and try again. Ultimately, it will try looking for a file named "default" (in lower case). Here is a short example: For 192.0.2.91, it will try C000025B, C000025, C00002, C0000, C000, C00, C0, C, and default, in that order. Well, after knowing where the name originates we can look into at the config.

Listing 3: C0A80-file for pxelinux

```
prompt          1
timeout         30
default         yagi
display         boot.msg

F1 boot.msg
F2 info.msg
F3 about.msg

label initrd
    kernel ../kernel/bzImage-2.4.19
    append initrd=../kernel/initdisk.gz init=/auto root=/dev/ram0
    ipappend 1

label yagi
    kernel ../kernel/bzImage-2.4.19
    append root=/dev/nfs nfsroot=192.168.1.2:/YAGI/linuxroot ro
        init=/boot NFSDIR=192.168.1.2:/YAGI/linuxroot
    ipappend 1
```

Let's explain the parameters.
prompt: show the prompt cursor for input menu name
timeout: load default label after <n> seconds
default: default label to load
display: show following msg at startup
F<n>: show msg <string> by pressing one of the F-keys
label: catch following kernel, add append line and execute it. In line commands should be obvious.

The files `boot.msg, info.msg` and `about.msg` are plain text ASCII files with a few command options, like clear screen for example. So lets look at the `boot.msg` file.

Listing 4: boot.msg

```
^L
^O0b

                                    _
          __   ______ _____ _(_)
         / / / / __ `/ __ `/ / /
        / /_/ / /_/ / /_/ / /_/ /
        \__, /\__,_/\__, /_/
       /____/        /____/
      yet another ghost installer

^O07
F1=[main]    F2=[info]    F3=[about]
```

The letters *^L* are the command to clear the screen. To input that into vi(m), press in the insert-mode the keys: <CTRL>+<V> followed by pressing <CTRL>+<L> will insert the letter *^L*. This will clear the screen at first. The letters *^O0b* and *^O07* changes the colours of the output.

4.3.2 result

PXELINUX boots in the first stage. That means that no Network card support is needed in PXELINUX, the support of the interface is handled by the 'second stage', which is in this case the linux kernel.
PXELINUX is the best choice we can make. The installation is easy and the supported hardware is excellent. So let's use it!

5 Linux Boot Process

After turning on Power the Hardware jumps into the BIOS, passing some options among other things the Boot sequence. When chosen 'booting over network' or something like this, the BIOS jumps into the Boot PROM, in new cards the Boot PROM is replaced by PXE, the PXE tries to get a valid IP-Address during a broadcast-request over BOOTP or DHCP. Subsequently there will be a short description of what happens after Powering on the Computer. After that there will be the boot concept of Linux explained.

5.1 Boot Terminology

The meaning of Bootstrap comes primarily from a Boot. The original Bootstrap is shown in a picture [figure 4].

- **Loader:** Program that moves bits from disk (usually) to memory and then transfers CPU control to the newly 'loaded' bits (executable)
- **Bootloader / Bootstrap:** Program that loads the 'first program' (the kernel)
- **Boot PROM / PROM Monitor / BIOS:** Persistent code that is 'already loaded' on power- up
- **Boot Manager:** Program that lets you choose the 'first program' to load

Figure 4: Bootstrap

5.2 What is a Loader?

A program that moves bits (usually) from disk to memory and then transfers control to the newly loaded bits (executable).

5.3 Bootstrap Loader (Boot Loader)

'The program that loads the first program' would be the best description to fit what a Bootstrap does. Usually the mini program is 'staged' in two files. The primary and the secondary Loader. To load these files, more exactly the primary stage, there is some firmware support, e.g. BIOS, needed. This 'hardware' support copies the first stage into the memory and executes it. The first stage loads then the secondary stage. This is a bigger program. You all know that programs. They are called, grub, lilo, ... and so on. The first and the second stage together are known under 'bootloader'. They allow to set some parameters for the kernel or choose your Operating system which you want to boot. Maybe this graphic[figure 5] will explain better.

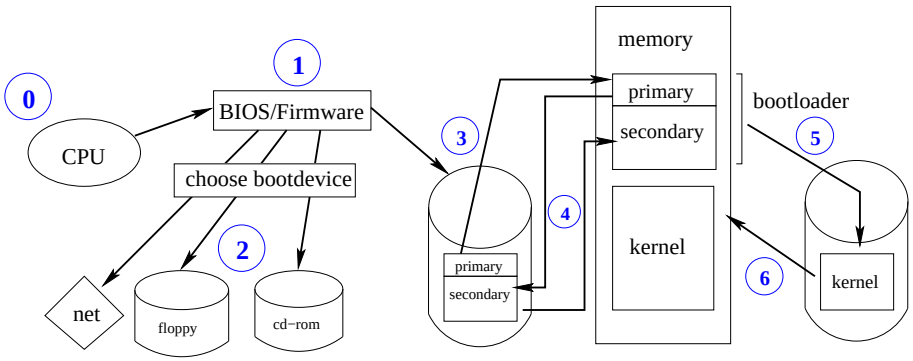

Figure 5: BootLoader

0 **Power on status:** After pressing the button on the power supply the CPU starts working.

1 **BIOS initial:** The CPU loads the BIOS Code and executes them. User interrupt at this moment is possible to modify the Boot Device or Boot Order.

2 **Boot Device:** Reading BIOS options and choosing Boot Device.

3 **first stage:** The BIOS now loads the Bootstrap on the Boot Device into the memory and executes them.

4 **secondary stage:** The executions of the first stage now loads the second stage from the Boot device and executes likewise.

5 **kernel:** The Bootloader now loads the kernel into memory. This kernel location can be different from the Boot Device.

6 **finish:** Kernel is copied into memory and executed.

5.4 Kernel startup

Now the Linux kernel takes over the control of the system. It initialises itself, eventually auto probes devices, initialises detected devices, and, if all this is done, it starts init, the process that parents all other user-level processes of the system. Here is a list in what order the kernel initials.

* identify bootstrap processor (BSP)
* setup_arch()
* init crucial subsystems
* parse_options()
* setup kernel profiling
* enable interrupts (sti())
* calibrate_delay() – BogoMIPS
* init subsystems needing delay
* check_bugs()
* smp_init()
* spawn init as a 'kernel thread'
* become idle process!

5.5 Invocation of init

init() begins life as a 'kernel thread' and ends by starting the user-level init process `/sbin/init`

* acquire 'the big kernel lock' on a multiprocessor (MP)
* perform high-level initialisation do_basic_setup()
* free_init memory
* release lock
* try to exec (in user space) the init process
* panic if unsuccessful

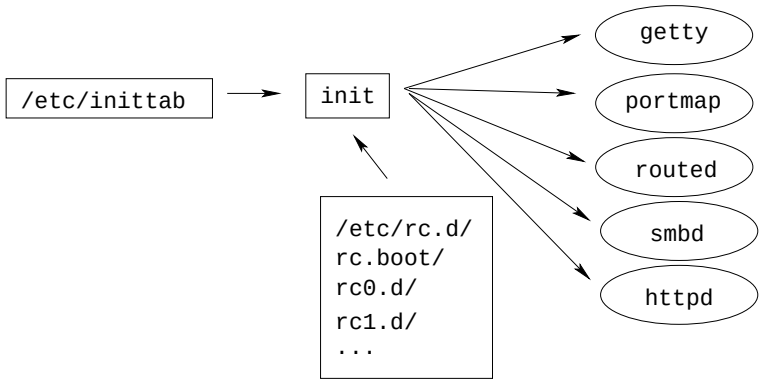

Figure 6: /sbin/init, the order of booting

Now you are at the point where you can choose where to go. The init process reads its configuration file (/etc/inittab) and processes scripts or commands instructed in this file.

The default runlevel and first script to be executed, if not booting in emergency (-b) mode, can be set here.

During the first step, hardware initialisation, you won't find many customizable options. You'll find instructions on how to modify BIOS settings in your main board manual. If you have adapters that have their own BIOS, check these manuals in case you must or can change something.

5.6 Start scripts

The first script to be executed from init on system startup is `/etc/rc.d/boot`. Here everything is started. Somewhere located in this script there is a section where scripts, that are only called once at startup, will be executed. This should look like this:

Listing 5: executing startup scripts in `/etc/rc.d/boot.d`

```
if test -d /etc/init.d/boot.d/ ; then
        for i in /etc/init.d/boot.d/S*; do
        test -f $i || continue
                $i start
        done
fi
```

After processing these scripts the init-process [figure 6] jumps into the denoted runlevel of the initab. The default runlevel is 3 so there will be all programmes executed which suffice the following expression. `/etc/rc.d/rc3.d/S*`

5.7 Symbolic links

For some files we now create symbolic links to the RAM-Disk or to the virtual /proc file system. All links, we talk about where created in the `/YAGI/linuxroot` directory of course. In the root directory we link the `/tmp` directory to `/var/tmp` and create this directory in the `/var`. Now change to the `/etc`. Link the `adjtime` to `../RAM/etc/adjtime` and the `mtab` to `/proc/mounts`. Ready.

5.8 Prepare the tar.gz ramdrive-files

Building RAM disks is very easy. Nevertheless it takes a long time and you have to handpick the files you need.
Copy the `/var` to a location of your choice. Remove all files and directory's you don't need. This depends on your distribution and what you have installed. In normal case you can remove anything with httpd, mysql, ... or something like that, if not needed for installation. Compare this directory with the runlevel, you start at boot time. So you can find out what you need or not.
Some errors during the boot up could arise if you deleted a file which is needed by a program. When this happens examine carefully the RAM drive disks and build a new one if you forgot something.
Let's do the RAM-drive directory. Layout an empty dir with these four directories: `dev, etc, tmp, yagi`.
Copy the file `HOSTNAME, adjtime, defaultdomain` and `ld.so.cache` into the `/etc` directory. Maybe you want to modify the `HOSTNAME` file.
Last but not least you have to do this for the `/dev` directory. Delete the devices which have no assignment on the install machines.
From each directory make tar.gz files like `ramdrive.tar.gz, dev.tar.gz and var.tar.gz`. Copy these three files into the `/YAGI/linuxroot/YAGI` folder your arranged before these.

6 YAGI

Some commercial and free-ware Tools already exist in that sector. All commercial programs are ”black box” solutions. That means you can not configure individual things to your needs and you are limited to the features of the respective programs. You can only use the option that the coders have found resourceful or more worse the business economist specified the features. Nevertheless there are some good programs on the market. The Tool Ghost from Symantec[1] is one of the best Programs out of the commercial sector. So I decided to call my program yagi, what to announce as ”yet another ghost installer”. So the idea of YAGI was born.

6.1 Idea

A tool based on UNIX[5] should be developed. It’s a fact that UNIX is one of the most powerful Operating Systems that exist. The long evolution and the resulting stability was the point for choosing UNIX. In case of the BSD-Family[4] (NetBSD, FreeBSD and OpenBSD) and of course Linux the source code of the kernel plus the userland programs are Open Source. So it is no problem to modify programs to fit your needs, or to build a free Disk-Cloning Tool based on a free Operating System which everyone can use and modify.
Before I start to explain how to implement and use the Tool a small abstract of the requirements.

6.2 Pegged requirements

The basics pegged requirements:

- **boot Unix-system complete over network (no need of local Disk-space).** You need not to install anything on your machine to deploy images. Also there is no need to save the image on the local disk first, what means that you can not use the whole diskspace for your applications.
- **create images independent of file-system or hardware.** You don’t have to care about your file system. Whatever you have (e.g. NTFS, FAT32, ext2, XFS, reiserfs) it should be able to clone. Also if you have IDE, SCSI, S-ATA, IEEE, . . . , all hardware should be supported.
- **fast and reliable transfer.** A common and fast protocol which supports flow control and affirmation of packets it receives. So the transfer of the image would be correct.
- **using standard Unix tools.** There should be no great need of compiling and installing special programs. This enhances the count of platforms yagi can be installed to and run on.
- **easy to use.** Keep the user input in any case as low as it can be.

The high demands:

- **status of transferring.** Progressbar of installation, estimated time till finishing, . . .
- **further automatisation.** Automatisation in the way to reduce the interaction before, during, and after the installation process.
- **auto detection of hardware for individual image deployment.** Parsing the BIOS or the MAC-Address and choose the image by means of those parameters.

6.3 Architecture

There exist two sides. The server and the client. The Daemons DHCP and FTP have not to run on one server. Likewise the ftp server have not to stand in the broadcast domain of the DHCP Server. Maybe you boot from a floppy or a CD so you only have to copy the two scripts `eatdisk` and `cookdisk` to your client and start them. Nevertheless there are some minimum requirements. So let's see which they are.

6.3.1 eatdisk

On the serverside a DHCPD Server has to run and provide IP-Addresses in the local broadcast domain. This can be any Server that is able to provide bootp request. Also an standard ftp server is needed to store the image. The Server has to be capable to support more than one client.
The client has to be able to boot from Network. Nothing else is needed. Anyway it has to be connected to the switch which is linked with the server that provides the DHCP.

6.3.2 cookdisk

If you are not booting from CD or Floppy you have to run a DHCP-Server. The image also can fetched from CD instead of the network. In our case it's served by ftp. Likewise to eatdisk the client has to be able to boot from Network. The script fetches data from a source file (here it's overhanded via ftp) and writes the bytes to the hard disk in the local client.

7 Installing YAGI-Server

First of all, what is a YAGI Server?
A YAGI Server is a standard UNIX, or any other Operating system, who can provide dhcp, tftp, nfs and ftp services. To install the YAGI-Server choose your favourite Unix system with following daemons and tools. In this case Linux is used.
On the Client side there are computers which have a Network interface which is bootable over Network.
The two sides (Server and Clients) are connected over a switch or hub. In this Broadcast domain all Clients can boot from Ethernet.

7.1 DHCP

DHCP, the Dynamic Host Configuration Protocol, describes the means by which a system can connect to a network and obtain the necessary information for communication upon that network. We use you the ISC (Internet Software Consortium) DHCP implementation, so all implementation-specific information given here is for use with the ISC distribution.

When dhclient, the DHCP client, is executed on the client machine, it begins broadcasting requests for configuration information. By default, these requests are on UDP port 68. The server replies on UDP 67, giving the client an IP address and other relevant network information such as netmask, router, and DNS servers. All of this information comes in the form of a DHCP "lease" and is only valid for a certain time (configured by the DHCP server maintainer). In this manner, stale IP addresses for clients no longer connected to the network can be automatically reclaimed.

So far, PXE also tries to get a valid IP Address from the DHCPD Server.

By all means you should check your startup script of the dhcp daemon. Make sure that the device, the network card which connects to the internal broadcast domain, is chosen right. If you haven't configured the right device the Client won't boot. Likewise it can evoke instability in the office-network, a workmates computer can suddenly catch an

IP-Address from your IP-Address range you offer from your server. So it would be encapsulated in your network and can no longer work in the office network.

7.1.1 A word about bootp

Well, as heard before the BOOTP Protocol should not longer be used, but why?

The first reason is that in most cases you have to launch dhcp-client after you booted up the minimum system. So it's in nearly all cases necessary to have dhcpd running on your server. It makes no sense to run BOOTP also if dhcpd can do this, too.
Another advantage is that DHCP allows more options, for example you can send so called Vendor options to your client. So you can specify special parameters for each machine or for all machines. This will reduce the administrative work.
It also uses the same ports as BOOTP, so DHCP can be seen as enhanced BOOTP service and should be used instead of the superceded BOOTP version.

7.2 tftpboot

Trivial file transfer protocol. Trivial in the meaning there that are only a few commands supported and there is no user-authentification. So it's clear that tftp never is used as standard ftp-daemon. During booting from a network and serving files like pre-boot-loaders and linux source it isn't necessary, likewise not possible to authenticate persons or machines.
tftpd is started over inetd, in the line you can choose a directory which is used as ftp-root. You should use this option because if you don't, everyone can copy any file from your server!

Listing 6: entry into `inetd.conf` for tftpd

```
tftp    dgram   udp     wait    root    /usr/sbin/in.tftpd      in.tftpd -s /YAGI/tftpboot
```

7.3 NFS-Server

We need an NFS-Server for mounting the root-file system and the userland environment after the kernel has successfully started up. NFS is always called unstable and insecure. That might be right in the case of insecure, because the authentication is only based on IP-Addresses. In our close environment during booting and deploying images this is not a security hole. So we don't have to worry about any security breaches in our network.
Well, let's see the config of the NFS-Server in `/etc/exports` .

Listing 7: example `/etc/exports` on YAGI-Server

```
/YAGI/linuxroot  192.168.1.0/24(ro)
/usr             192.168.1.0/24(ro)
```

Do your configuration carefully, mistakes in that case or simple mistypes can result in security holes. Although be careful with write access (rw instead of ro), for debugging and if the staff knows what to do this might be okay, else wise set this parameter to read-only, so no one can overwrite your configuration or delete files of your server system!

7.3.1 NFS-Root

As you see above we have a directory that is called
tt /YAGI/linuxroot. This directory is simple a copy of the 'original' root file-system on the server.
In the following you should copy these directories to the location that you have chosen as the linuxroot in your NFS-config.

* /bin
* /dev
* /etc
* /lib
* /proc (only create)
* /root
* /sbin
* /tmp (only create)
* /usr (only create)
* /var (only create)

To copy the device-directory you can use the following command.

Listing 8: copy command for `/dev` directory

```
cp -rp /dev/ /YAGI/linux/root/dev
```

The directories /var, /tmp, /proc and /usr have to be empty because they will be created or mounted at startup. Last but not least the Kernel has to support NFS-Root, which is explained in the next section.

7.4 Built a proper Kernel

It's very simple to compile a new kernel. goto kernel.org, grab the latest kernel and do a regular 'make menuconfig'.

Following the menu and choose anything you would choose as normal. Some things you have to point out. All network cards which are installed in the clients have to be compiled as 'static' into the kernel. If you choose 'module' it will not work because the necessary files are saved on the server directory. The kernel tries to load the module files from `/lib/modules/<kernel-version>`, in our case the /lib directory doesn't exist at the time the kernel starts. The system has to mount the nfs-root at startup for getting access to the /lib file system. To mount the nfs-share you need a working Ethernet-device.

Also activate the hardware detection, the tmpfs file system, the NFS-root support and the devpts. You can proof the configuration if you look at the .config file and see something like this.

Listing 9: coutout from kernel-config

```
...
# File systems
CONFIG_TMPFS=y
CONFIG_RAMFS=y
CONFIG_DEVPTS_FS=y

...

# Network File Systems
CONFIG_NFS_FS=y
CONFIG_NFS_V3=y
CONFIG_ROOT_NFS=y
...
```

Thats all. Now you can go on with compiling the kernel.

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

The new kernel is located under `arch/i386/boot/bzImage`. Copy the kernel to the `/tftpboot/kernel` directory. Also do not forget to make sure that the directory `/usr/lib/<kernel-version>` is accessible over NFS to yagi. Normally this should not be a problem. If an error occurs during the boot-process when ldconfig is running, make the NFS /usr share writable for one startup. Remember, if you change something in the `/etc/exports` you have to restart the NFS daemon. Use `kill -HUP 'cat /var/run/mountd.pid'` or something like `/etc/rc.d/nfsserver restart` for restarting the service.

8 Modified scripts

Well, we have to configure some shell-scripts that linux will execute at startup. Most of the scripts can be disabled. Generally we start runlevel three. So we have to check the `/etc/rc.d/rc.3/*` files, delete or rename them.

8.1 Config file yagi

Under `/YAGI/linuxroot/etc/sysconfig/yagi` we create our own CONFIG File for easier adapting to other environments.
Here is an example. Configure the parameters to your needs.

Listing 10: example `/etc/sysconfig/yagi`

```
# This is the configuration File for YAGI
# edit Variables only here

# How should your YAGI Server be named
YAGI_SVR_NAME="YAGI"

# IP-Address of YAGI-Server
YAGI_SERVER=192.168.1.2

# Directory of user-land binaries which should be ro mounted
YAGI_USR=/usr

# Directory of user-land binaries which should be ro mounted
YAGI_HOME=/YAGI/linuxroot/home/yagi
```

This file will be needed by some scripts which are explained now. Similarly you can add Variables for your individual needs in this file.

8.2 /etc/rc.d/boot

This is the file linux executes after the kernel has unpacked itself.
Here we have to create the Ram drives, untar it, do some mounting and re-mountings. The best thing is to show script and explain what it does before long talk about what to do.

Listing 11: `/etc/rc.d/boot`

```
....
#
# Building an RAM DISK for individual workstations
#
echo -n "Preparing RAM-Disk for local config"
```

```
6  mount -t tmpfs -o size=23M,nr_inodes=10k,mode=700 tmpfs /RAM
   rc_status -v -r
8
   rc_reset
10 #
   # Building an RAM DISK for individual workstations
12 #
   echo -n "Preparing RAM-Disk for /var"
14 mount -t tmpfs -o size=42M,nr_inodes=10k,mode=755 tmpfs /var
   rc_status -v -r
16

18 # Make /proc available
   rc_reset
20 echo -n "Mounting /proc device"
   mount -n -t proc proc /proc
22 rc_status -v -r

24 echo -n "Mounting /dev/pts"
   optpts="-o mode=0620,gid=5"
26 mount -n -t devpts $optpts devpts /dev/pts
   rc_status -v -r
28
   rc_reset
30 #
   # Untarring filesystem into place
32 #
   echo "Untarring Files into RAM-Drive on /RAM for local-config..."
34 test -w /proc/progress && \
         echo "52 Creating basic filesystem" >/proc/progress
36 mkdir /RAM/dev
   mknod /RAM/dev/null c 1 3
38 tar --exclude=*.tgz -xpzf /YAGI/ramdrive.tar.gz -C          \
         /RAM >/dev/null 2>&1
40 rc_status -v1; rc_reset

42
   #
44 # Untarring /var
   #
46 echo "Untarring /var into RAM-Drive ..."
   tar --exclude=*.tgz -xpzf /YAGI/var.tar.gz -C /var >/dev/null 2>&1
48 rc_status -v1; rc_reset
   ....
```

More or less all functions are documented in the script itself. Nevertheless I explain the lines and what they are standing for.

Line	commands
5	prompt to console
6	create and mount an tmpfs filesystem with disksize 5M for `/etc`
7	prompt an [done] if command before has been executed successfully
14	create and mount an tmpfs filesystem with disksize 23M for `/var`
21	mount the /proc fileystem
24-26	mounting console device
33-39	ungzip and untarring ramdrive.tar.gz from `/YAGI/linuxroot/YAGI`
47	ungzip and untarring var.tar.gz from `/YAGI/linuxroot/YAGI`

All Ramdrives are mounted read/write accessable. So every client can write its own config-files and logfiles into memory.

8.2.1 Why using tmpfs

If I had to explain tmpfs in one breath, I'd say that tmpfs is like a ramdisk, but different. Like a ramdisk, tmpfs can use your RAM, but it can also use your swap devices for storage.

And while a traditional ramdisk is a block device and requires a mkfs command of some kind before you can actually use it, tmpfs is a filesystem, not a block device; you just mount it, and it's there. All in all, this makes tmpfs the niftiest RAM-based filesystem I've had the opportunity to meet.
You're probably wondering about how big that tmpfs filesystem was that we mounted before. The answer to that question is a bit unexpected, especially when compared to disk-based filesystems. `/var` or `/RAM` will initially have a very small capacity, but as files are copied and created, the tmpfs filesystem driver will allocate more VM and will dynamically increase the filesystem capacity as needed. And, as files are removed from `/var`, the tmpfs filesystem driver will dynamically shrink the size of the filesystem and free VM resources, and by doing so return VM into circulation so that it can be used by other parts of the system as needed. Since VM is a precious resource, you don't want anything hogging more VM than it actually needs, and the great thing about tmpfs is that this all happens automatically.
The other major benefit of tmpfs is its blazing speed. Because a typical tmpfs filesystem will reside completely in RAM, reads and writes can be almost instantaneous. Even if some swap is used, performance is still excellent and those parts of the tmpfs filesystem will be moved to RAM as more free VM resources become available. Having the VM subsystem automatically move parts of the tmpfs filesystem to swap can actually be good for performance, since by doing so, the VM subsystem can free up RAM for processes that need it. This, along with its dynamic resizing abilities, allow for much better overall OS performance and flexibility than the alternative of using a traditional RAM disk.

8.3 /etc/rc.d/boot.d/

The chronological order of starting the scripts is nescearry because if portmap isn't running you can not mount netshares for example. So the correct order of booting is significant. To ensure that the bootsequence is executed in the right order the scripts are sorted in alphabetical order. For easier changing the alphabetic sequence and thus the ascending order, regularly a 'S' followed by a two digit number is inserted before the name of the script. All scripts are symbolic links to the above directory. Here a directory-listing from `/etc/rc.d/boot`

Listing 12: `/etc/rc.d/boot`

```
S01boot.idedma -> ../boot.idedma
S02boot.proc -> ../boot.proc
S07boot.swap -> ../boot.swap
S08boot.clock -> ../boot.clock
S09boot.ldconfig -> ../boot.ldconfig
S10boot.localnet -> ../boot.localnet
S20boot.setup -> ../boot.setup
S22boot.klog -> ../boot.klog
S23boot.ipconfig -> ../boot.ipconfig
```

8.3.1 /etc/rc.d/boot.ldconfig/

In this file there is not much to modify. What you should have already done is that you copied the `ld.so.cache` file to the RAM-Disk. In other words, the location of `ld.so.cache` file is under `/RAM/etc/` directory after the initial startup through `/etc/rc.d/boot`. So we change all `/etc/ld.so.cache` to `/RAM/etc/ld.so.cache` in this file.
Check carefully if this file is really there and called by `ldconfig`. If not, some strange things could happen.

8.4 /etc/rc3.d/netshares

For mounting the netshares I have written an own small script. Here are some abridgement lines from the file. I copy&pasted only the essential parts here. All other is similar to other scripts, look if you find an `/etc/skeleton` in your distribution. So it's easy to modify or make your own runlevel-script.

Listing 13: `/etc/rc.d/boot`

```
...
#Check for existence of needed config file and read it
test -r /etc/sysconfig/yagi && . /etc/sysconfig/yagi
...
echo -n "Starting mounting NFS $YAGI_SVR_NAME Shares"
mount -t nfs -o ro,tcp,vers=2 $YAGI_SERVER:$YAGI_USR /usr
mount -t nfs -o rw,tcp,vers=2 $YAGI_SERVER:$YAGI_HOME /RAM/yagi
# Remember status and be verbose
rc_status -v
...
```

9 YAGI-tools

For creating and writing images I have written some small programs which are based on G4U[7]. These are only some small shell-scripts and one perl program for displaying the status and the progress of the installing and writing to a disk.
It is recommended to create an ftp-user for storing or reading images. The user only needs ftp access, so there is no full shell account required. Just as well it is possible to put the image on another server instead of the YAGI server.

9.1 eatdisk

This script creates the image.
It reads the whole hard disk over dd and streams the output to an ftp-client which transfers it to the server.
The script is very small. At first it checks if the command line arguments are suitable. The minimum requirement is the server name and the name of the image you have to declare. The third parameter sets the local disk device. In default case it is set to `/dev/hda`.
Then it starts a 5 seconds delay to interrupt the process before reading from disk.
According to these settings the image creation is starting. In detail the following commands are executed. The file system tool `dd` starts with reading the hard disk. The output is piped through an `gzip -9`. Thereby the image file size is reduced to the effective size of data located on the hard disk. Otherwise the image would as big as the raw data-size of the hard disk.
The compressed data is sent via ftp to the server. ncftpget is parametrised for auto login into the server. The upload process starts automatically. The image size grows as fast as the read process on the client-side has advanced. In many cases the whole hard disk of the client contains no data. There exist free space on the fixed disk. This free space can be better compressed than the data-areas. Implication of that is that the image would grow in a variable way. After the progress a small goodbye message is displayed.
The complete script can be found in the appendix.

9.2 cookdisk

This script writes the image to the client.
It begins with an ftp connect to the specified server in the command line. The received data will be streamed through gunzip and then written with dd onto the local fixed disk in the

client.
It s also a very short script and also only standard UNIX tools are used. In detail the script checks the following parameters and executes accordingly.

`cookdisk` awaits minimum two command line options. The first parameter is the server name and the second mandatory parameter is the image name. The third parameter is optional and declares the disk device e.g. hda, hdb, ... Default setting here is also `/dev/hda`.
After processing this parameters it loops through a 5 seconds start sequence. This gives you the chance to interrupt the execution without writing any data. Maybe you will be fast enough and recognise an error in your input or, the worst case, if you mixed up the keyboards on your desktop.
Now the image writing starts. The standard lukemftp client connects to the specified server and performs an auto login. The also specified file is fetched from the server and handed over to gunzip. We have to unzip this because we have packed the image with gzip as described before. The extracted out-stream is piped into the fileutil dd.
Similarly to `eatdisk` the first time there will be a lot transfered from the server to the client. This is trivial and should not be described here.
After the progress a small goodbye message is displayed.
The complete script can be found in the appendix.

9.3 YAGIso

YAGI System Overview program. This is a small perl-program which will read out some system-parameters and show them on the display. It's updated every 3 seconds. This refresh-time is enough, because most of the values don't change and the others, that change don't do that in such an fast way. The most needful function of this program is that you can see the status of the writing and reading of an image. To make this possible it was mandatory to modify the dd source code an recompile it. How this is done is explained in the next section.

The actual process is calculated as follows.
From the proc-file system the maximum hard drive capacity can be read out via the ide-channel. This value is located under `/proc/ide/hda/capacity` for example.

The program uses the ncurses libs to draw the boxes and the progress bar, so make sure you have installed the perl-module, elsewhere the program doesn't start.

In future versions maybe it's possible to calculate the exact time how long it takes to write the image to the disk.
The perl-program can be found in the appendix.

9.4 modified dd

As said in the section above, we have to modify the dd command. There are only small enhancements for getting an output to the screen and writing the actual block count to a file. The complete source code of the modified dd.c you will find in an additional file. Here I only explain the lines which have been attached or modified. This will help more than a `diff` if you have an another version of the fileutils than here. So let's take a look what have to be done.

Listing 14: modified dd.c (partitial)

```
...
// File for Statistic output. done by DaN for yagi
FILE* statsf;
...

...
else
```

```
8   copy_simple (bufstart, n_bytes_read);

10   // modified by Stefan | DaN for yagi
     if ((++count_print_reduction%100)==0)
12   {
       fprintf(stderr, "\r%d of %d blocks written...", (long) (r_partial+
           r_full), (long) max_records);
14     fflush(stderr);
       fseek(statsf, 0, SEEK_SET);
16     fprintf(statsf, "%d blocks\n", (long) (r_partial+r_full) );
       fflush(statsf);
18   }
}
20
   // modified by Stefan
22 fprintf(stderr, "\r                                                    \r");
   fflush(stderr);
24 ...

26 ...
main (int argc, char **argv)
28 {
   int i;
30 int exit_status;

32 // modified by DaN for yagi
   statsf=fopen("/var/dd.stats","w");
34 ...
```

I give a short explanation of what the sections do in the program.
In line 3 an File Handle is created.
Line 7 to line 19 is somewhere in the middle of the source code. In fact this is the main part to recode. As you see both outputs are active, the output to the console and the output to the file-handle is done. With some C-knowledge it should be no problem for you to read this.
Now let's take a look at the main function. Here in line 33 the file handle is opened to a file for writing. Stefan Robl[6] helped me by recoding this.

10 Performance Tests

To figure out how fast yagi is, and how it depends on CPU speed or HD size I ran some test. All test have been made on Toshiba Laptops. On regular Workstations the disk speed should be better than in these tests. As a matter of fact is that the Clients where all Notebooks, you would like to ask what's so special with this. Well, Notebooks hard drive are all in 2.5" form factor. The speed averages at 5400 rpm and also the write performance is good deal worse than on regular desktop hard-drives.
The time tables, to the tests described here, you can find in the appendix on the sides from 27 to 28.
At first the hardware of the server where in all tests constant. It was used an Pentium4 2.0GHz with a Intel Etherexpress 100 Network Card. 512 Megabyte Memory and a usual Maxtor 20GB Ultra ATA/100. The performance could be improved if you use two network cards and bonding them and/or use a disk-raid system.
The first test[table 1] was to find out how the modified dd binary would affect the speed of disk writing. As exposed the time of writing to disks wasn't changing very much. This is very good for us so we can run the YAGIso without having any great loss of time.
The same was done with another machine to see how great the differences between hardware platforms are. We supposed some great changes. As a matter of fact the time difference was very great. You can see this in table 2.
In table 3 you can find data with one machine an two different disk and image sizes. In the

first cycle I chose a small disk image and an small image. The second pass was an great big disk and a image size double big as the one before. I tried to figure out how long would be the minimum and the maximum time of time to install one computer. Alike it can be seen the influence of hardware if you accord with other tables.
The hardware in this test[table 4] is exact the same. Only the disk size changes from test to test. The intension in this case was to discover the write time of different image sizes on varying disk sizes. It's possible to write an image from an 20GB Disk Device to an 30GB or greater Device. The system is runnable without no problems. The advantage is that the smaller disksize produces a smaller image and is therefore faster in writing to the disk.
The last two test where made to prepare YAGI for Real Life mission. One test where driven with 20 Computers and the other with 30 Computers. To be able to compare the difference between one machine and the amount of all, a stand alone install where also driven. Alike the block size has altered from default to 1 Megabyte. As result can be seen that the change of the block size have an tremendous change in the install time. In the case of the standalone install the change is even higher than in the total install. This is due to sth. the heavy load of network traffic. The first time all machines will leech a great amount of data to their hard disk. So the network couldn't cope with the needs of the clients.

11 Real World Lab

After adequate test series yagi is ready for the real world.
Up to 40 machines have to be installed. Nearly all computers have the same disk size. In the table below you can see how long it took to install the machines. As stated below you can see the difference between installing one machine with a image or all together.
The result is that not as supposed the time of installing would increase tremendously. Far from it the time is nearly the same.
Most of the test where driven with only twenty laptops because of the test sequence which was running on them more machines where not needed.
The requirements in this environment was to install quickly 20 machines with an software image on which run several hardware and software test under windows to prove the quality of the machines. After the clearance of this machines the next 20 machines have to be installed with an another image. To reboot automatically the new machines after the installation has finished we just entered following command line
`cookdisk <server> <image> && shutdown -nr now`.
So the computers performed a automatically reboot after installing the image and run into the windows software to begin with the tests.
Another time the creation of the image wasn't finished before closing time. The master notebook which created the image had to run a hour or longer to create the image. The solution is as simple as the above. We started up all notebooks and log in. Using the following command line
`sleep 1000 && cookdisk <server> <image>`
enabled us to go home and have an already installed image on the next morning.
As you can see the choice to set on an standard UNIX like operating system we have all the power of commands this operating system provides. The flexibility is greater than any black box designed commercial program and of course it is cheaper. In the Appendix on page 44 you can find some photos of the installing procedure.

12 future prospects

Okay, let's see what can be improved or what has already been tested.

Idea 1:
An idea was based on the fact that all clients which install the same image get exact by the

same data over the network. So it should be possible to run a multicast ftp daemon instead of the regular ftp daemon. On the client site nearly all ftp can handle a multicast stream. The problem in that case is that the clients have to start all at exactly the same moment. This problem I have solved with the following method. We set the timeout connection of the ftp client to 5min or more. In the `cookdisk` we add a line before the ftp client start that generates an empty file with the client-ip-address in a directory on the server. So far the client is ready, now lets see the server side. It's very simple, we write a script which checks how many files, or if all IP-Addresses we have uploaded with the modified ĉookdisk script exists. Well, if all files are there we start the multicast ftp daemon. After the program has started it begins to multicast the image file. All clients that are waiting for a connection will start immediately when the server is running. This very simple trick works very well.
This nice feature has one great disadvantage. Multicast stream includes no ACK or NACK. That means if a client doesn't get an IP packet it will never know that it has missed one.
I have discovered that problem when some clients were not ready with writing to the hard disk but the image was already been ready sent over the network. It's clear, that the client's had missed some packets because d̂d wanted still to write.
This phenomenon occurs in my case when I tried to install more than 10clients with this method.
It depends surely on the network cards, the switch or hub and other magic things how many clients can be installed via multicast.
After thinking about some time about this problem I had an idea how this problem can be solved. Due to the lack of time I couldn't test this idea. What follows are merely the basic ideas of the concept.
IPv6 is the solution. Version 6 of IP can set transfer speed to an IP-Address. Get clued? Right, we set a speed limit for each IP-Address that the server and all clients can cope with. So it can theoretically be assured that all clients get all packets from the server. Obviously that can not be guaranteed.

Idea 2:
The process of installing can be more automated.
First there can be set some Vendor Options in the `dhcpd.conf` for clients so it will be possible to parse this option in the `cookdisk` script and fetch an image automatically without typing anything into the client machine. Of course some modifications have to be done. The client must auto execute the `cookdisk` at startup, but this is not a great problem.

Idea 3:
The local disk can be mounted as shared via NFS or Samba. So modifications directly after the installations are possible and computers can be individualised with config files for example.

Idea 4:
UNIX also supports many remote features. So it's possible to access every machine over ssh, rsh, telnet, ftp or whatever. So you can start the clients and let them boot. After they have booted completely you can login from your desktop PC and start the installation. Also its possible to see the progress of the installation over the network.
That's only a fraction of that what is possible. Imagine a web front end where you see all machines that have booted and a list of images that are possible to install. A lot of lines code have to be implemented to reach this target, nevertheless it is possible.

That are only some Ideas I have had. Therefore it will exist a lot more.

13 Toshiba Europe GmbH

The complete development and testing was done and sponsored by Toshiba Europe GmbH.

14 greetings and thanks

First of all I would thank Mum & Dad. They supported me during my studying with a great heart and trusted in my abilities. Especially Mum was always solicitous that I have enough to eat. Alike I would thank my brother who built me an fly screen that prevented me from being pricked all over by mosquitos during my long computer nights.
Thanks to Hubert Feyrer and Prof. Juergen Sauer, my Diploma-Thesis supervisors. They gave me plenty of rope and I had a great time where I learned a lot more than I thought.
Team23, the students of my semester at the FH-Regensburg.
Greetings and thanks to all Open Source Developers. Keep on coding and thanks for all the stuff.
Last and not least, all my friends who invited me to parties, spending holidays with me, drinking coke, beer and doing stuff at all.
So long, and thanks for the best time of my life...

Herbert ”Dad” Ettle, Dagmar ”Mum” Ettle, Christian ”ChrisE” Ettle, Karl Ettle, Helene Ettle, Markus ”Whisky” Wistop, Alexander ”Ad” Adlhoch, Karl ”Koarl” Ertl, Eva, Karsten ”Preis” Witzke, Marion Stadler, Fabian ”Fab” Abke, Bernd ”Bapf” Stapf, Daniel ”Cocker” Hoepfl, Martin ”Polar” Kloeckner, Christian ”KrK” Kraus, Tino ”Rogue” Hirschmann, Klaus ”Mac” Brandl, Ingo ”Bif” Frank, Thomas ”Blixxer” Graf, Martina ”tdeer” Hirsch, Marcus ”Teschi” Prinz, Herbert ”Aries_Blu” Meilhammer, Matthias ”matmaxx” Weigel, Stefan ”castla” Schlosser, Andreas ”anderl” Gleissl, ”AlienMind” Nickl, Thomas ”sherriff” Mayer, Wolfgang ”sniper” Wagner, Andreas ”delta9” Neumeier, ”nordish”, Bettina ”Plymmo” Echinger, Beate ”BT” Echinger, Sabine ”Bine” Abke, Katrin ”zak” Zahnweh, Alice Sefr, Ines ”ini” Buckl, Sabine ”lion” Mueller, Franzi, unknown CIP-Pool women, Hubert ”hubertf” Feyrer, Juergen Sauer, Martin Pohl, Edwin Schicker, Martin Opel, Claudia Durchholz, Frau Hirschmann, Batman, Harold ”logix” Gutch, Alex ”decay” Goller, Stefan ”AmigaRulez” Robl, Andrea ”Andrew” Ondracek, Eugen ”eugenO” Obermeier, Thorsten ”thorstenS” Steller, Stefan ”stefanG” Grosse, leahcim, wuschel, Obelix2000, oneway, double-p, Sonja84 aka Streusel, Mephisto, Teehase, eckes, schrett, Wolfgang Genger, Roland Gassner, Manfred Kirchberger, Vadim Martahaler, God, old Airport Munich Ultraschall, RAF-Club, suite15, Union Move, Street Parade, the first few Loverparades

15 Appendix

15.1 Performance Test - Time Tables

To improve the write time, discover the influence of the different versions of dd I run several test. Here are the results.

Table 1: Write Speed test with differents versions of dd

System: Pentium4 1.5 GHz/256 MB/30GB — **Image Size:** 4.5GB	
standard dd	1:53:27
modified dd with screen output only	1:53:41
modified dd with screen output and file output	1:54:15

Table 2: Read/Write test with regular and modified dd with screen and file output

System: Pentium3 1.0 GHz/256 MB/20GB — **Image Size:** 1.1GB		
version of dd	*reading*	*writing*
regular	0:30:26	1:27:34
modified	0:32:15	1:30:08

Table 3: Read/Write test. regular and modfied dd. same hardware/changing disksize

System: Pentium4 1.0 GHz/256 MB		
Image size: 1029061668 bytes — **Disk size:** 20GB HD		
version of dd	*reading*	*writing*
regular	0:25:05	0:50:16
modified	0:25:56	0:50:36
Image size: 2432401719 bytes — **Disk size:** 30GB HD		
regular	0:48:05	1:23:26
modified	0:48:34	1:24:37

Table 4: Read/Write Test with system `dd` and several disk sizes

System: Pentium4 1.6 GHz/256 MB		
Image size: 1112634549 bytes — **Disk size:** 20GB HD		
image from disk	*reading*	*writing*
20 GB	0:26:18	0:50:47
Image size: 1199118301 bytes — **Disk size:** 30GB HD		
image from disk	*reading*	*writing*
20 GB		0:53:56
30 GB	0:38:18	1:22:53
Image size: 1097823111 bytes — **Disk size:** 40GB HD		
image from disk	*reading*	*writing*
20 GB		0:50:13
30 GB		1:14:17
40 GB	0:44:17	1:41:12

Table 5: Write Test with different block sizes and amount of computers

System: Pentium Celeron 1.8 GHz/256 MB		
Image size: 2799942587 bytes — **Disk size:** 30GB HD		
nr. of machines	*default*	*block size 1M*
1	1:13:57	0:32:13
20	∼2:11:23	∼1:37:42

Table 6: Write Test with different block sizes and amount of computers 2

System: Pentium4 1.8 GHz/256 MB		
Image size: bytes — **Disk size:** 30GB HD		
nr. of machines	*default*	*block size 1M*
1	0:59:57	0:32:05
20	∼1:41:23	∼1:28:23

15.2 Example `dhcpd.conf`

Listing 15: example `dhcpd.conf`

```
#
# DHCP configuration file. ISC DHCP server v3.0
#
#
# Global options

### dont update DNS
ddns-update-style none;
ddns-updates off;

### allow netboot
allow bootp;
allow booting;
boot-unknown-clients on;

### free IP Address if Client forgot to sent DHCP_RELEASE
one-lease-per-client true;
use-host-decl-names on;

### short for times for testing
default-lease-time 600; max-lease-time 720;

### default times for real-life deployment
#default-lease-time 72000; max-lease-time 144000;

min-lease-time 23;

### define grub-options for *lst
option option-150 code 150 = text;

### define pxelinux option space
option space pxelinux;
option pxelinux.magic      code 208 = string;
option pxelinux.configfile code 209 = text;
option pxelinux.pathprefix code 210 = text;
option pxelinux.reboottime code 211 = unsigned integer 32;

### define bpbatch options for interactive mode
option option-135 code 135 = text;

subnet 192.168.1.0 netmask 255.255.255.0 {
        ### generall dhcp settings
        range 192.168.1.23 192.168.1.223;
        option broadcast-address 192.168.1.255;
        option subnet-mask 255.255.255.0;
        option routers 192.168.1.2;
        option domain-name "toshiba-tro.de team23.org gwdg.de";
        option domain-name-servers 192.168.21.5, 192.168.21.6;
        option time-offset 775238962;
        option ip-forwarding off;

        # all path options relative to tftp-server


        ### general options for booting over network
        next-server 192.168.1.2;
        option root-path "/YAGI/linuxroot/";    # relative tftp-server path
        #filename "/grub/pxegrub";              # grub
        filename "/pxelinux/pxelinux.0";        # syslinux(pxelinux)
        #filename "/bpb/bpbatch";               # bpbatch


        ### options for grub
```

```
        option option-150 "/grub/menu.lst";
66
68      ### options for pxelinux
        option pxelinux.magic f1:00:74:7e;      # practical magic
70      option pxelinux.configfile "yagi";      # evident
        option pxelinux.pathprefix "/YAGI/tftpboot/pxelinux/";
72      option pxelinux.reboottime 5;           # reboot after 5 min
74
        ### BpBatch command-line argument : -i == interactive
76      # You can also specify a script name (do not include the
        # trailing .bpb extension)
78      option option-135 "bpb/yagi-ascii"; # -i  bpb/yagi-ascii";
80
        ### some more options
82      server-name lenny;
        #option ntp-servers ntps1.gwdg.de, ntps2.gwdg.de;
84      #option time-servers rdate.uni-regensburg.de;
        #option font-servers lenny.simpsons.org;
86      #option netbios-name-servers 134.76.63.252;
        }
88
host coke {
90      hardware ethernet 00:90:27:A4:C8:E2;
        fixed-address 192.168.1.23;
92      server-name "coke.deam.org";
}
```

15.3 cookdisk script

Listing 16: cookdisk script

```
#!/bin/sh
#
# YAGI – cookdisk – dan@deam.org
#
# based on g4u, hubert@feyrer.de
#
# fetch disk image and write it to local disk

server=$1
image=$2
device=$3

if [ "$device" = "" ]; then
        device=hda
fi

if [ "$image" = "" –o "$server" = "" ]; then
        echo ""
        echo "usage: $0 <servername> <imagename> [DEVICE]"
        echo "default device is hda - change only if needed"
        echo ""
        echo "example:   $0 lenny tecra9000"
        echo ""
        exit 1
fi

echo "start working in ..."

for i in 5 4 3 2 1 ; do
     echo $i ; sleep 1
done

echo "thats the point of no return - working..."
ftp –o "|gunzip|dd of=/dev/${device}" ftp://yagi:yagi@${server}/${image}

echo ""
echo "well done - cu next time..."
```

15.4 eatdisk script

Listing 17: eatdisk script

```
#!/bin/sh
#
# YAGI – eatdisk – dan@deam.org
#
# based on g4u, hubert@feyrer.de
#
# create image and write it over ftp to a server

server=$1
imgname=$2
device=$3

if [ "$device" = "" ]; then
        device=hda
fi

if [ "$imgname" = "" –o "$server" = "" ]; then
        echo ""
        echo "usage: $0 <servername> <imagename> [DEVICE]"
        echo "default device is hda - change only if needed"
        echo ""
        echo "example:   $0 lenny tecra8000"
        echo ""
        exit 1
fi

echo "start working in ..."

for i in 5 4 3 2 1 ; do
     echo $i ; sleep 1
done

echo "working..."
( dd if=/dev/$device | gzip –9 ) | ncftpput –d /var/log/ftp.log –c –u yagi
     –p yagi $server $imgname

echo ""
echo "thats all folks..."
```

15.5 YAGIso script

Listing 18: YAGI System Information

```
#!/usr/bin/perl -w
#
# YAGI - System Information Tool
#
# DaN - dan@deam.org
#


# Programm settings and libaries
use strict;             # trying harder
use Curses;
use Curses::Widgets;  # Included to import select_colour & scankey
use Curses::Widgets::TextField;
use Curses::Widgets::TextMemo;
use Curses::Widgets::ProgressBar;


# declare Variables
my ($mwh, $progr, $blocksAll, $blocksDone, $cpu, $load, $boottime, $percent
    , $doneblocks);
my (@capacity, @blocks) = ("");


# this is line 23 :-)

# Set up the environment
$mwh = new Curses;
#noecho();
halfdelay(5);
$mwh->keypad(1);
curs_set(0);
# this is line 23 ;-)


# give me a nice look
$mwh->attrset(COLOR_PAIR(select_colour('green')));
$mwh->box(ACS_VLINE, ACS_HLINE);
$mwh->attrset(0);

$mwh->standout();
$mwh->addstr(0, 1, "Welcome to the YAGI System overview!");
$mwh->standend();

# read the capacity data of hda
open(daten, "</proc/ide/ide0/hda/capacity") || die "capcacity of hda could
    no be read\n";
while(<daten>) {
        push(@capacity,$_);
}
close(daten);

# show me boottime
$boottime=Curses::Widgets::TextField->new({
  CAPTION               => "Boot Uptime",
  CAPTIONCOL            => "red",
  LENGTH                => 42,
  MAXLENGTH             => 42,
  MASK                  => undef,
  VALUE                 => "  ".procinfo(7,0,32),
  INPUTFUNC             => \&scankey,
  FOREGROUND            => "white",
  BACKGROUND            => 'black',
  BORDER                => 1,
  BORDERCOL             => 'green',
```

```
    FOCUSSWITCH             => "\t\n",
    CURSORPOS               => 0,
    TEXTSTART               => 0,
    PASSWORD                => 0,
    X                       => 5,
    Y                       => 2,
    READONLY                => 1,
});


# show me current load
$load=Curses::Widgets::TextField->new({
    CAPTION                 => "load average",
    CAPTIONCOL              => "red",
    LENGTH                  => 50,
    MAXLENGTH               => 50,
    MASK                    => undef,
    VALUE                   => "The Simpsons",
    INPUTFUNC               => \&scankey,
    FOREGROUND              => "white",
    BACKGROUND              => 'black',
    BORDER                  => 1,
    BORDERCOL               => 'green',
    FOCUSSWITCH             => "\t\n",
    CURSORPOS               => 0,
    TEXTSTART               => 0,
    PASSWORD                => 0,
    X                       => 5,
    Y                       => 6,
    READONLY                => 1,
});

# how many blocks are done
$blocksDone=Curses::Widgets::TextField->new({
    CAPTION                 => "current Block",
    CAPTIONCOL              => "yellow",
    LENGTH                  => 15,
    MAXLENGTH               => 23,
    MASK                    => undef,
    VALUE                   => '  12439',
    INPUTFUNC               => \&scankey,
    FOREGROUND              => "white",
    BACKGROUND              => 'black',
    BORDER                  => 1,
    BORDERCOL               => 'green',
    FOCUSSWITCH             => "\t\n",
    CURSORPOS               => 0,
    TEXTSTART               => 0,
    PASSWORD                => 0,
    X                       => 10,
    Y                       => 10,
    READONLY                => 1,
});


# how many blocks are done
$percent=Curses::Widgets::TextField->new({
    CAPTION                 => "percent",
    CAPTIONCOL              => "blue",
    LENGTH                  => 7,
    MAXLENGTH               => 10,
    MASK                    => undef,
    VALUE                   => "   0%",
    INPUTFUNC               => \&scankey,
    FOREGROUND              => "white",
    BACKGROUND              => 'black',
    BORDER                  => 1,
```

```
    BORDERCOL                => 'green',
    FOCUSSWITCH              => "\t\n",
    CURSORPOS                => 0,
    TEXTSTART                => 0,
    PASSWORD                 => 0,
    X                        => 32,
    Y                        => 10,
    READONLY                 => 1,
});


# capacity field [hda]
$blocksAll=Curses::Widgets::TextField->new({
    CAPTION                  => "total Blocks",
    CAPTIONCOL               => "red",
    LENGTH                   => 15,
    MAXLENGTH                => 23,
    MASK                     => undef,
    VALUE                    => "  ".$capacity[1],
    INPUTFUNC                => \&scankey,
    FOREGROUND               => "white",
    BACKGROUND               => 'black',
    BORDER                   => 1,
    BORDERCOL                => 'green',
    FOCUSSWITCH              => "\t\n",
    CURSORPOS                => 0,
    TEXTSTART                => 0,
    PASSWORD                 => 0,
    X                        => 46,
    Y                        => 10,
    READONLY                 => 1,
});


# is the bitch working?
$cpu=Curses::Widgets::TextMemo->new({
        CAPTION                  => "CPU Info",
        CAPTIONCOL               => "red",
        LENGTH                   => 38,
        MAXLENGTH                => 230,
        LINES                    => 5,
        MASK                     => undef,
        VALUE                    => "bla\nblub",
        INPUTFUNC                => \&scankey,
        FOREGROUND               => undef,
        BACKGROUND               => 'black',
        BORDER                   => 1,
        BORDERCOL                => 'green',
        FOCUSSWITCH              => "\t",
        CURSORPOS                => 0,
        TEXTSTART                => 0,
        PASSWORD                 => 0,
        X                        => 5,
        Y                        => 17,
        READONLY                 => 1,
});


# how far to go
$progr=Curses::Widgets::ProgressBar->new({
    CAPTION                  => "done",
    CAPTIONCOL               => "yellow",
    LENGTH                   => 60,
    VALUE                    => 0,
    FOREGROUND               => "yellow",
    BACKGROUND               => 'black',
    BORDER                   => 1,
```

```
        BORDERCOL               => "green",
        HORIZONTAL              => 1,
        X                       => 5,
        Y                       => 13,
        MIN                     => 0,
        MAX                     => 100,
});

# needs to be only drawn once
$blocksAll->draw($mwh, 1);
$boottime->draw($mwh, 1);

#for(my $i=0;$i<$capacity[1];$i+=100)
while(1)
{
        $doneblocks=ddstats();
        $load->setField( VALUE => "  ".procinfo(7,36,72) );
        $blocksDone->setField( VALUE => "  ".$doneblocks );
        $percent->setField( VALUE => ($doneblocks/$capacity[1])*100 );
        $progr->setField( VALUE => ($doneblocks/$capacity[1])*100 );
        $cpu->setField( VALUE => " ".procinfo(9,0,32)."\n ".procinfo
            (10,0,32)."\n ".procinfo(11,0,32)."\n ".procinfo(12,0,32)."\n "
            .procinfo(13,0,32) );

        $load->draw($mwh, 1);
        $blocksDone->draw($mwh, 1);
        $percent->draw($mwh, 1);
        $progr->draw($mwh, 1);
        $cpu->draw($mwh, 1);

        sleep 1;
}



# read the actual status of writing to disk
sub ddstats{
        open (daten, "</var/dd.stats") || die "dd stats file could no be
            read\n";
        @blocks = <daten>;
        close (daten);

        return substr($blocks[0],0,-8);
}

# read out procinfo and return line <n> from <x> to <y>
sub procinfo{
        my $line = shift;
        my $start = shift;
        my $stop = shift;

        # do a system call and write output to tmp-file
        system '/usr/bin/procinfo > /tmp/procinfo.txt';

        my @plines = (""); # remember strict?
        open (daten, "</tmp/procinfo.txt") || die "procinfo could not be
            read\n";
        while(<daten>) { push(@plines,$_); }
        close(daten);

        return substr($plines[$line],$start,$stop);
}


# show cursor again
curs_set(1);
```

```
260
    # The END block just ensures that Curses always cleans up behind itself
    endwin();

    # thats all folks
265 exit 0;
```

15.6 BIOS Bootdevice

Photo of BIOS dialog to choose the Boot device

Figure 7: Menue to choose Boot device

Menue icon to choose if network boot should be enabled

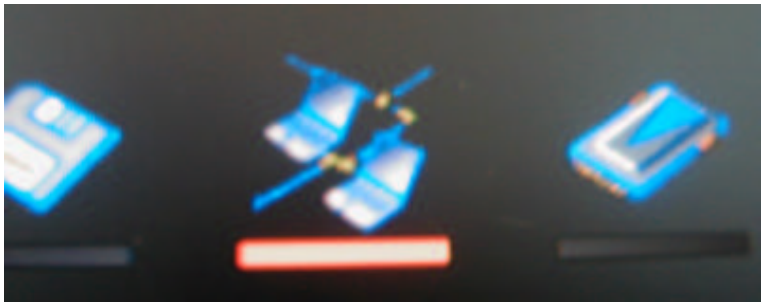

Figure 8: Small Picture from Network Boot Logo

15.7 PXE Bootscreen

Photo of PXE BIOS Boot screen. Old PXE BIOS Version, because of not Displaying Networkcard.

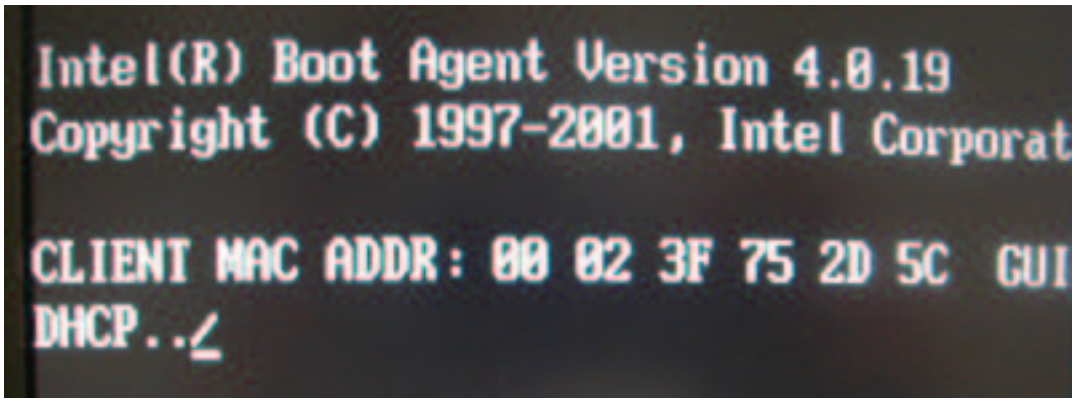

Figure 9: PXE BIOS screen (old version)

Photo of PXE BIOS Boot screen with new 2.0 Version of PXE. Network card ist displayed.

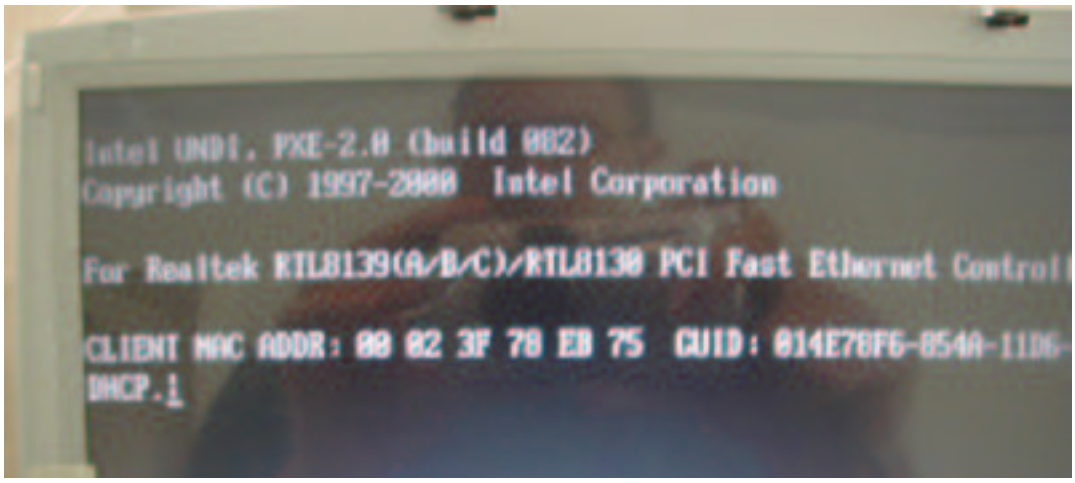

Figure 10: PXE BIOS screen (new version)

15.8 PXE Bootscreen

After fetching an valid IP Address from the DHCP Server the PXELINUX is been loaded over tftp. The follwing output is displayed. In real-life ypu won't see this because of the fast execution of this step.

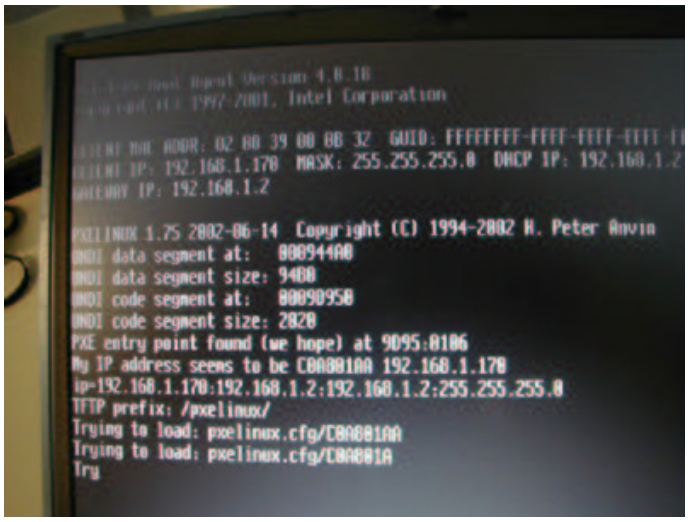

Figure 11: PXE Linux boot sequenz

THE PXELINUX Boot Splash screen. This output can be individual modified.

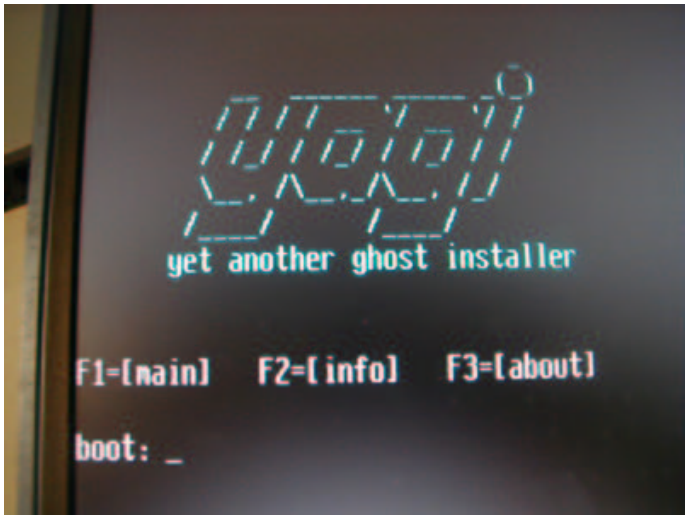

Figure 12: The PXELINUX Boot Splash

15.9 eatdisk screen shots

Photos of eatdisk after executing. Command line options in this case was `eatdisk lenny satellite2410`. Lenny was in this name the server name and satellite2410 the image name of the laptop

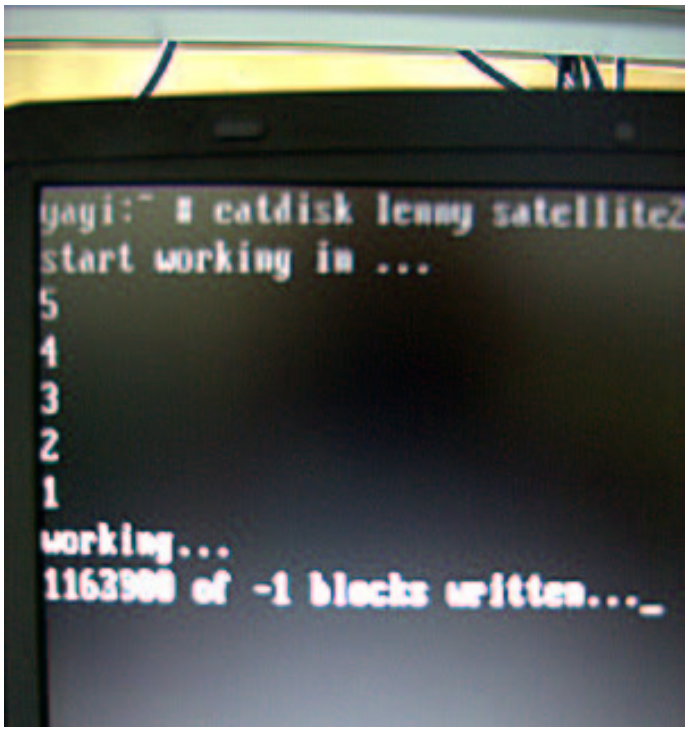

Figure 13: eatdisk output after starting, during reading image

Close up shot of the output from the modified dd tool. You can see the blocks which are already written. The LCD is to slow so the output isn't as sharp as on an regular Monitor.

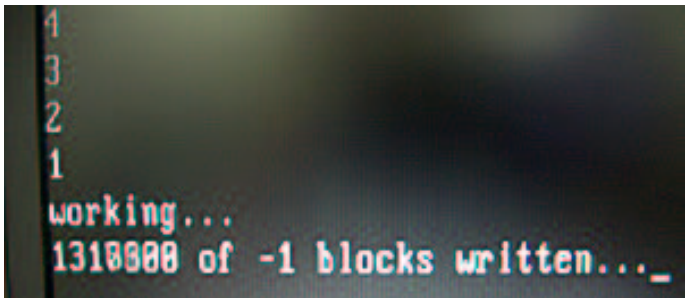

Figure 14: Close up shot of the eatdisk output during processing data

15.10 cookdisk screen shots

Screen shot ot cookdisk after command line input `cookdisk lenny s2410`. Here is lenny the server name and s2410 the image name. If you have not an DNS server or your entry in the `resolv.conf` is wrong try the IP address instead of the name. Notice that the image name has been renamed from satellite2410 to s2410. Less letters to type. :-)

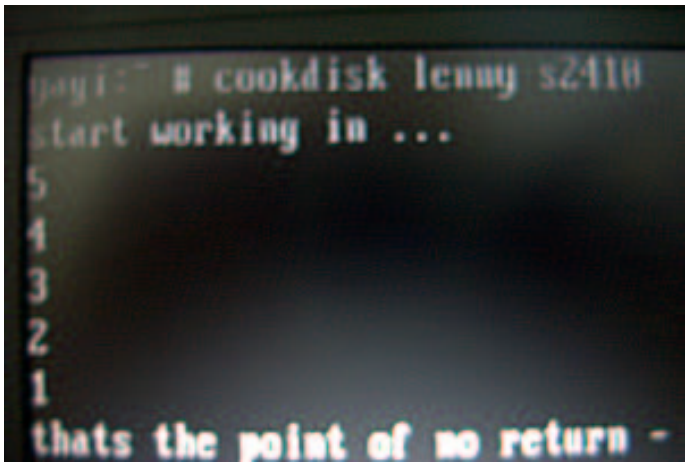

Figure 15: cookdisk output

After writing the image you see some time and speed data. So you can compare this with my test environment.

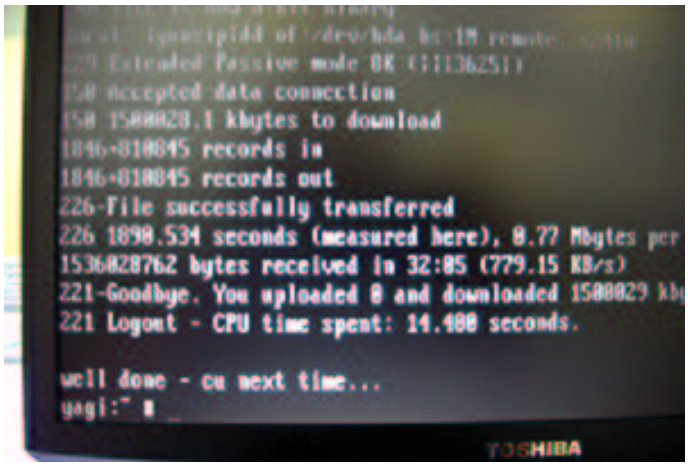

Figure 16: cookdisk after writeing the image. summary

15.11 YAGIso screen shots

Here is the screen output from YAGIso (YAGI System Overview). As you can see there are some usual data like uptime, load, and so on. Likewise you can see a progress bar and two field called 'current block' and 'total block'. The meaning is obvious and is not explained here.

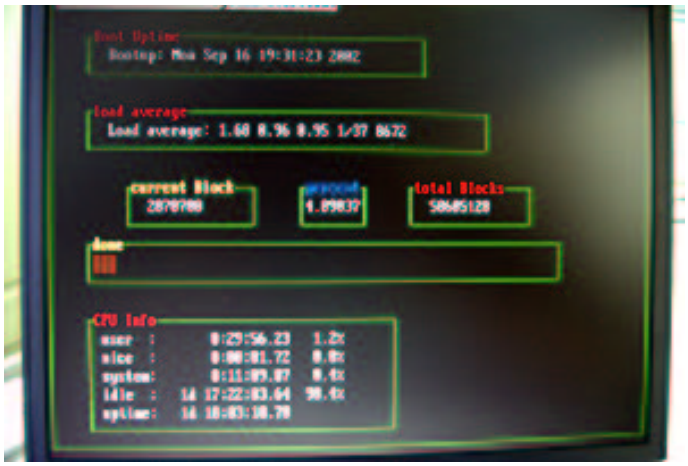

Figure 17: full screen shot of YAGIso.

The progressbar calculation and the percent display is trivial and likewise not explained.

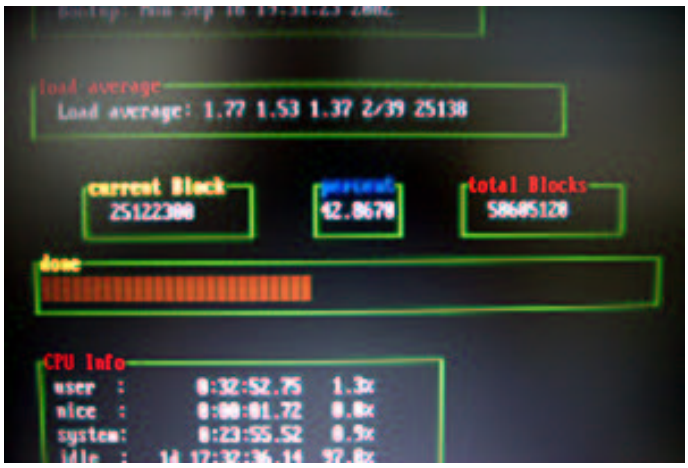

Figure 18: detailed YAGIso screens hot of the progressbar.

15.12 Real World action

Here you can see the Real World deployment environment. On this photo you see 20 machines who are installed simultaneously. In this special way there where all notebooks.

Figure 19: Real World Lab Photo 1

Another view of the application area.

Figure 20: Real World Lab Photo 2

16 Resources

- http://www.faqs.org, RFC
- http://www.freebsd.org, how-to’s
- http://www.feyrer.de, g4u
- http://www.linuxdocs.de, how-to’s
- http://www.stud.uni-goettingen.de/ dsuchod/ldc/node5.html, Thin Clients
- http://www.google.com
- http://www.linuxnetmag.com/de/issue5/m5diskless1.html
- http://www.zelow.no/floppyfw/install.html
- http://www.escape.de/users/outback/linux/

List of Figures

List of Tables

Listings

References

[1] Symantec Ghost™ *http://www.symantec.com/sabu/ghost/,* ©1995-2002 Symantec Corporation. All rights reserved.

[2] Intel PXE Spezifikation, *http://www.intel.com/labs/manage/wfm/index.htm* , Wired for Management (WfM)

[3] LaTeX, *Helmut Kopka, Band1: Einfuehrung, 3., ueberarbeitete Auflage* , ©2000 by Addison-Wesley Verlag

[4] BSD™, *BSD is a registered trademark of Berkeley Software Design, Inc.*

[5] UNIX™, *UNIX is a registered trademark licensed to X/OPEN.*

[6] Stefan Robl (stefan@qdev.de), Student of Computer Science, Friendship since 1998

[7] Ghost for Unix, developed by Hubert Feyrer [FH-Regensburg Sysadmin], www.feyrer.de

17 Explanation

1. I hereby declare that I have written this dissertation on my own, have not yet submitted it for any other examination purposes, have used only the cited sources and aids and have clearly indicated all direct quotes and borrowed ideas.

2. I am aware that dissertation as examination material will become the property of the independent state of Bavaria.

3. I hereby give my consent that the Fachhochschule Regensburg allows students of the Fachhochschule to read this dissertation and that it may publish the dissertation by using my name as the author, provided that the publication is not ruled out.

Regensburg, September 11, 2002 _____

DaNiel Ettle