# ADSL software router with firewalling and virtual private networking on embedded devices with Linux on the example of a SEGA Dreamcast gaming console.

Christian Berger

October 19, 2004

Revision History

| Revision | Date | Author |
|---|---|---|
| Revision 1.2 | 06/11/2003 | Christian Berger (c.berger@tu-braunschweig) |
| Revision 1.1 | 04/03/2003 | Christian Berger (c.berger@tu-braunschweig) |
| Revision 1.0 | 03/27/2003 | Christian Berger (c.berger@tu-braunschweig) |

**Abstract**

This article describes the necessary steps and backgrounds for building a software router with the operating system Linux on a SEGA gaming console. Therefore, it points out the way of building and designing a *toolchain* for compiling programs and the Linux kernel in conclusion with the build of a Linux operating system from scratch running entirely in the memory.

Today, the total costs of ownership *(TCO)* of a personal computer are that low, so why am I trying to build a software router on a gaming console? More aspects wake my curiosity to walk on the very stony way for getting Linux on the gaming console.

First, the hardware architecture is a non-x86 computer system, so the attraction for getting in contact with a foreign platform is one reason. Next, I want to know, how to setup and use *cross compilers*. If you have a running cross compiler, you're on a very shiny way - but, cross compiler raise (other) problems you won't probably have if you are just trying to compile your favorite program which was normally designed for an x86 platform. Last but not least, I haven't found any articles describing these steps I've shortly explained above, so, I want to be the first :-)


*SEGA Dreamcast* gaming console.

**Dedication**

This document is my way to show my gratitude to the great Open Source Community out in the world for their great work.

# Contents

# List of Figures

# List of Tables

# 1 Introduction and theory

This chapter is about introducing to the world of *cross compiling* and the associated problems. It also describes the general differences between compiling programs and building a Linux system from scratch for an x86 architecture and a foreign platform on the example of the Hitachi SH7750 chipset embedded in the *SEGA Dreamcast*. Last but not least it lists the hardware components that are necessary to use this article.

## 1.1 What *is* cross compiling?

A short description of cross compiling would be the following term:

*Cross compiling is the procedure for building a program for a platform different from the one on which the cross compiler runs. "Platform" does not only mean the hardware architecture but also software platforms, e.g. the process for building the* GNU/Hurd *operating system from sources on a running Linux for the same hardware architecture is also a cross compiling.*

But "cross compiling" covers more than the short statement above. For using and building a cross compiler, you need to know more than only *download the source ; ./configure ; make ; make install*. Frequently, problems appear which you wouldn't expect if you just compile a package for your computer. A generic solution for arising problems would be correcting it by installing or compiling the needed libraries - but, mostly, this *doesn't* work with a program which has to be cross compiled.

Personally, I would describe this process at least *recursive*. You've got a source code of the famous program *foo* and you notice that it depends on library *libfoobar*. So, you get the source of this library, but then, you remember, the library itself depends on another library *libbar*...

As you notice, you firstly have to compile the package *libbar*, then you can correct the problems of library *libfoobar* and then, you can finally compile the original source of *foo*.

Don't get frightened - many programs you want to compile are well described or you may get help from the Internet, news groups or mailing lists from all around the world.

The programs used in this article have all been tested and compiled. Mostly, they include a patch which represents a workaround of problem *xyz* for the platform *SEGA Dreamcast*.

## 1.2 Cross compiling programs

If you want to cross compile a progam, the steps don't obviously differ from the ones needed to compile it for your personal computer. But there are some things to keep in mind for avoiding getting tired or crazy.

First, the platform *which* runs the cross compiler is called *host*. The target platform, *for which* the program is compiled, is called *target* (really? :-).

The program, you've compiled doesn't run - on your host, unless you've got a simulator or you've loaded it to the target. All the programs in this article are loaded in the target because it doesn't exist a simulator for Linux for the *SEGA Dreamcast* unfortunately.

The following problems while cross compiling appear frequently during the writing process of this article (ordered by difficulty):

1. The included *Makefile* uses an improper compiler.

   A generic solution would be customizing the *Makefile* by courtesy of an environment variable:

   ```
   ...
   christian@helicon:~$ export CC=/usr/local/foo/cc
   christian@helicon:~$ make
   ...
   ```

   Another possibility would be the modification of the *Makefile* and the change of the entry for the *CC*= variable.

   The examples above don't refer only to the C - compiler but also to *ar*, an archiver, to *ranlib*, an indexer for archives or to other programs used to compile the source.

2. The included *Makefile* doesn't find the proper header - files or libraries.

   This problem has normally two reasons: The *Makefile* searches in the improper directories or the library - and header - files don't exist. The first problem can be solved as mentioned above: You modify your environment or the *Makefile* of the source - package. The latter is solved by installing the missing library.

3. The source - package doesn't include a *Makefile*.

   This problem is sometimes easy to solve, but sometimes you have to cheat and bluff *make* that you have a correctly constructed *Makefile*.

   These packages often include a script called *configure*. This script tries to build a *Makefile* using lots of templates (*config.in*, *Makefile.in*, ...) for getting an optimal *Makefile* which is suitable for your system. Occasionally, you can use the *configure* - script for building a *cross compiling* - *Makefile*, but often you will run into problems due to missing test programs because these test programs produced by the *configure* - script can't run on the host platform.

Sometimes, there's a trick to run `configure` for the host platform and edit the resultant *Makefile* for the target platform.

4. The compiler complains about *xyz*.

   This never looks good. The problem can be solved easily or costs nights to be solved. Often, it's a missing directory or program that can be added easily. But sometimes, the compiler complains about code that can't be compiled for the target you wish. This is up to assembly code that won't run on or can't be interpreted by the target platform. The latter is hard to solve because you have to look for a workaround, a switch or an environment variable that prohibits the use of native assembly code. The *README* included in the source - package would be a good start.

5. The source compiles well but doesn't run on the target platform.

   This problem is very hard to solve. Do you check every output of the compiling process? Is the program executable on the target platform? Is it a problem of rights (*chmod*)? Are some device nodes missed? Are some configuration files missed? You may solve the problems by using *gdb*, the *GNU debugger*.

As you see, cross compiling is nothing you do in five minutes, but it's something very interesting and challenging. The programs described in this article come along with a patch that helps you to compile the source for the sample target *SEGA Dreamcast* and comprehend the modifications I've got to perform.

## 1.3 Building Linux from scratch

*Why are we building a Linux from scratch? I've heard that . . .*

Surely, there exist many distributions, for workstations and for embedded devices, such as *EmLinux* or *ucLinux*. But they have a main lack: They won't run on the *SEGA Dreamcast* because these distributions support only boards with a large demand either from industry or community. *SEGA Dreamcast* has "only" a small demand of individualists (the IRC channel contains about ten people all the time I was there).

Another point of view is that these distributions are ready-to-run, i.e. you download or install it, burn a CD or transfer an image to a board of your choice and everything's down. A self-made Linux distribution does (or doesn't) only that what *you* want. Not more, not less. Furthermore, you can specify which compiling options of a software package you need. And finally you understand what your distribution is doing.

In spite of that, these distributions are also a good start for getting to know what we need.

*Why aren't we using NetBSD? I've read that . . .*

Certainly, you can use NetBSD for building a software router. I was *very* surprised how faultless their userland compiles for the *SEGA Dreamcast*. But, a few remarks: The NetBSD kernel is about 3,3 MB *without* the userland. You've got only 16 MB

in your *SEGA Dreamcast*. Some HowTos exist for connecting a *SEGA Dreamcast* running NetBSD with an NFS server, but I don't want to run another computer for using my *SEGA Dreamcast* as a software router.

So, these are the reasons I describe the process of building a *Linux distribution*.

While reading, you will build an embedded Linux *from scratch*, i.e. you are responsible for the steps directly after loading and executing the kernel.

But take in consideration that you're building a Linux for an embedded device. You want to maximize the functionality but have to minimize the need of bytes. Keep this *trade-off* in mind while selecting programs you want to run on an embedded device.

As a valuable start for building a complete Linux system directly from scratch, you probably would read the book *"Linux from scratch"* [Beekmans98]. This book describes the steps for building a Linux system from sources by yourself (this process is a cross compile, do you remember?).

Reverting to our main goal, that book is a great guide for a workstation with a hard disk drive and a lot of space for holding various programs. So, we have to build a *really small* Linux "distribution" for our software router.

The following table points out some *really* essential programs for running a kernel and a shell on your *SEGA Dreamcast*:

| **Package** | **Size** | **License** | **Why?** |
|---|---|---|---|
| `linux-2.4.18.tar.bz2` | 23 MB | GPLv2 | The kernel. |
| `busybox-0.60.5.tar.gz` | 767 KB | GPLv2 | A login shell (among others). |

Table 1.1: List of essential programs for running a kernel and a shell on your *SEGA Dreamcast*

Two sources? Yes, you're right. You need only these two packages for getting an embbed Linux distribution. But don't be glad too early, it's much work for getting these both packages compiled and loaded by the *SEGA Dreamcast*.

## 1.4 What do I need for starting with this article?

You need at least the following stuff to get along with the next two chapters:

- Of course, a *SEGA Dreamcast* gaming console.
- A personal computer with a running Linux distribution. I'm using the Debian GNU/Linux distribution *Woody*, but any other distribution with the *GNU Compiler Collection* would do the job. It's possible to do the job with any other Unix you want to (for example NetBSD, FreeBSD) or even Microsoft Windows

with Cygwin extensions, but I'll describe in this article the steps using (Debian) GNU/Linux, so keep this in your mind.

- A CD burner for transferring the software to your *SEGA Dreamcast*.

The following things make your life easier:

- A *Coder's Cable*. This is a serial cable for connecting a *SEGA Dreamcast* gaming console through a serial port with your host.
- A *VGA box*. This is an adapter for using your computer screen with a *SEGA Dreamcast*.

For a complete ADSL software router with firewalling and virtual private networking, you have to be in need of the following components:

- An ethernet networking adapter for the *SEGA Dreamcast*. Either the *SEGA Broad Band Adapter* (based on the RTL8139 chipset) or the *SEGA LAN Adapter/HIT-0300* (based on the MB86967) would do the job.
- Of course, the whole xDSL stuff: DSL modem, switch or hub, cables ...

The following table gives you a short overview of the costs for getting your feet messed up with this fun:

| **Component** | **Costs** |
|---|---|
| *SEGA Dreamcast* | 80 $ |
| Coder's Cable | 15 $ |
| VGA box | 25 $ |
| SEGA Broad Band Adapter | 110 $ |
| SEGA LAN Adapter/HIT-0300 | 145 $ |

Table 1.2: Costs for getting the whole stuff.

As you see, you will start your work for an ADSL software router with firewalling and virtual private networking for about 190$. Surely, it's much more as you would pay for a hardware router, but are you able to re-program your hardware router let alone to run your own Linux distribution?

# 2 Setting up an initial toolchain

This chapter describes the necessary steps for installing the Linux kernel and busybox mentioned above on your *SEGA Dreamcast*. It's based on Bill Gatliff's great article [Gatliff01]. This chapter doesn't want to replace Bill's article but it wants to update, complete and exchange some steps of his article.

## 2.1 Setting up your workstation

For proceeding, it's necessary to have the proper software installed. At least, you *need* a C - compiler with the building environment (all libraries and headers) and the Concurrent Versions System for getting the Linux kernel patches from their CVS repositories. Please check this and refer to your distribution handbook for help.

## 2.2 Setting up a cross compiling environment

You'll need up to twelve packages plus some build directories. So, I suggest creating a special directory to collect all needed packages and seperate them from the rest of your system:

```
christian@helicon:~$ mkdir -p Dreamcast/toolchain
```

This is just a proposal. You may install your toolchain in `/usr/local`, but according to my experience it's better to seperate the additional software for your workstation located in `/usr/local` from the toolchain.

For the future, I omit `christian@helicon:~` from my prompt to save space. The next step is to set up some environment variables:

```
$ export TARGET=sh4-linux
$ export PREFIX=/home/christian/Dreamcast/toolchain
$ export PATH=$PREFIX/bin:$PATH
```

Note, while you were logging out, your environment settings got lost! But you can create a file named `~/.dcenv` for example and let your `~/.bash_profile` set your environment if you login:

```
export TARGET=sh4-linux
export PREFIX=/home/christian/Dreamcast/toolchain
export PATH=$PREFIX/bin:$PATH
```

In your `~/.bash_profile`, you simply add:

```
  .
  .
  .
source ~/.dcenv
  .
  .
  .
```

Now, you're ready to download the initial stuff.

## 2.3 Downloading the source packages

Table 2.1 lists the package, size and URL, where you can download the source packages. A good choice is the download into a folder of our new cross compiling environment:

```
$ mkdir SRC
```

The next step is downloading and patching the Linux kernel. This is organized in three substeps:

1. Download the Linux kernel, version 2.4.18.
2. Fetch the sh - patches through CVS

   These patches are SuperH specific and the basic layer for the *SEGA Dreamcast* patches.
3. Fetch the *SEGA Dreamcast* patches through CVS

Figure 2.1 illustrates how these patches interact.

The first step mentioned above is fetching the Linux kernel sources. The kernel is licensed under the terms of the GNU Public License, Version 2 and about 23.0 MB. A nice program for the job is `wget`:

| Package | Size | License | URL |
|---|---|---|---|
| `binutils-2.11.2` | 9,701 KB | (L)GPLv2 | ftp://ftp.gnu.org/gnu/binutils/binutils-2.11.2.tar.gz |
| `gcc-3.0.1` | 17,631 KB | (L)GPLv2 | ftp://ftp.gnu.org/gnu/gcc/gcc-3.0.1/gcc-3.0.1.tar.gz |
| `glibc-2.2.4` | 16,023 KB | (L)GPLv2 | ftp://ftp.gnu.org/gnu/glibc/glibc-2.2.4.tar.gz |
| `busybox-0.60.5` | 767 KB | GPLv2 | http://www.busybox.net/downloads/busybox-0.60.5.tar.gz |
| `sh-boot-20010831-1455` | 233 KB | LGPLv2 | http://www.linuxdevices.com/files/article020/sh-boot-20010831-1455.tar.gz |
| `binutils (Patch)` | 92 KB | – | http://www.linuxdevices.com/files/article020/binutils-2.11.2-sh-linux.diff |
| `gcc (Patch)` | 101 KB | – | http://www.linuxdevices.com/files/article020/gcc-3.0.1-sh-linux.diff |
| `glibc (Patch)` | 617 KB | – | http://www.linuxdevices.com/files/article020/glibc-2.2.4-sh-linux.diff |
| `sh-boot (Patch)` | 35 KB | – | http://www.linuxdevices.com/files/article020/sh-boot-20010831-1455.diff |

Table 2.1: Source packages, sizes and URLs.

```
$ cd Dreamcast
$ mkdir BUILD
$ mkdir KERNEL
$ cd KERNEL
$ wget http://www.kernel.org/pub/linux/kernel\
  /v2.4/linux-2.4.18.tar.bz2
```

The backslash \ indicates that the following line belongs to the end of the actual line. The next step is the download of the linux-sh patches from the CVS repository (located at *www.sourceforge.net*). CVS will ask for a password, just press enter, no password is required:

```
$ cvs -d:pserver:anonymous@cvs.sf.net:\
  /cvsroot/linuxsh login
$ cvs -z3 -d:pserver:anonymous@cvs.sf.net:\
```

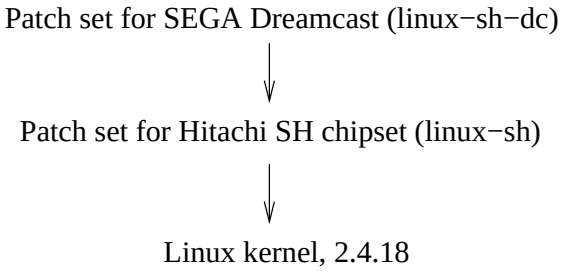


Figure 2.1: The patches against the Linux kernel.

```
  /cvsroot/linuxsh co -r linux-2_4_18 linux
$ cvs -d:pserver:anonymous@cvs.sf.net:\
  /cvsroot/linuxsh logout
$ mv linux linux-sh
```

The last step is the download of the linux-sh-dc patches from the CVS repository (located at *www.sourceforge.net*):

```
$ cvs -d:pserver:anonymous@cvs.sf.net:\
  /cvsroot/linuxdc login
$ cvs -z3 -d:pserver:anonymous@cvs.sf.net:\
  /cvsroot/linuxdc co -r linux-sh-dc-2_4_18 linux-sh-dc
$ cvs -d:pserver:anonymous@cvs.sf.net:\
  /cvsroot/linuxdc logout
```

Now, the tricky moment begins. You have to remove the CVS control directories to avoid annoying warnings and errors during kernel compilation and merge all three sources together:

```
$ cd linux-sh
$ for i in $(find . -type d -name "CVS" -print); do rm -fr $i; done
$ cd ..
$ cd linux-sh-dc
$ for i in $(find . -type d -name "CVS" -print); do rm -fr $i; done
$ cd ..
$ bunzip2 -c linux-2.4.18.tar.bz2 | tar -xv
$ cd linux-sh
$ cp -fr . ../linux
$ cd ..
$ cd linux-sh-dc
$ cp -fr . ../linux
$ cd ..
```

The next step is correcting a compiler switch in the sh - architecture *Makefile*:

```
$ cat linux/arch/sh/Makefile | \
  sed s/CFLAGS.*\+\=\ \-m4\ \-mno\-implicit\-fp/\
  CFLAGS+=\-m4\ \-m4\-nofpu/ > ./Makefile.old
$ cat ./Makefile.old | \
  sed s/AFLAGS.*\+\=\ \-m4\ \-mno\-implicit\-fp/\
  AFLAGS+=\-m4\ \-m4\-nofpu/ > ./linux/arch/sh/Makefile
```

The last step is patching the Linux kernel with the LAN adapter device driver:

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  lan_adapter-0.0.7-linuxsh-2.4.18.patch.tar.bz2
$ bunzip2 -c lan_adapter-0.0.7-linuxsh-2.4.18.patch.tar.bz2 | \
  tar -xv
$ cd linux
$ patch -p1 < ../lan_adapter-0.0.7-linuxsh-2.4.18/\
  lan_adapter-0.0.7.patch
```

Hopefully, you will have now a linux-sh-dc 2.4.18 synchronized and patched kernel source tree.

## 2.4 Building a cross compiler, linker and bootstrap compiler

Now, we're ready to compile our cross compiler, *linker* and *bootstrap compiler*. The first step is the build of the `binutils-2.11.2`. This package contains programs for archiving (*ar*), archive indexing (*ranlib*) and much more essential:

```
$ cd ../..
$ cd BUILD
$ tar -xvzf ../SRC/binutils-2.11.2.tar.gz -C .
$ patch -p0 < ../SRC/binutils-2.11.2-sh-linux.diff
$ mkdir build-binutils
$ cd build-binutils
$ ../binutils-2.11.2/configure --target=$TARGET \
  --prefix=$PREFIX
$ make all install
$ cd ..
```

Following, we're able to build the *GNU Compiler Collection*:

```
$ tar -xvzf ../SRC/gcc-3.0.1.tar.gz -C .
$ patch -p0 < ../SRC/gcc-3.0.1-sh-linux.diff
$ mkdir build-gcc
$ cd build-gcc
$ ../gcc-3.0.1/configure --target=$TARGET \
  --prefix=$PREFIX --without-headers \
  --with-newlib --disable-shared \
  --enable-languages=c
$ make all-gcc install-gcc
$ cd ..
```

The *configure* - script configures this package for the *SEGA Dreamcast* only with the specified (only the *C* language) compiler and their runtime libraries, `--without-headers`

prohibits the use of cross compiler header files (they actually don't exist!) and the use of the newlib C - library as the target C - library. This library will be replaced by *uClibc* in a later stage.

## 2.5 Setting up the Linux kernel

Now, you've got the first half of a C - compiler. The next step is the setup of the Linux kernel:

```
$ cd ..
$ cd KERNEL/linux
$ make ARCH=sh CROSS_COMPILE=sh4-linux- menuconfig
```

Please consider the dash next to *sh4-linux*.

The following list is a sample configuration for an ADSL software router with firewalling and an ethernet networking adapater:

| **Option** | **Type** |
|---|---|
| CONFIG_EXPERIMENTAL | * |
| CONFIG_MODULES | * |
| CONFIG_KMOD | * |
| CONFIG_SH_GENERIC | (Dreamcast) |
| CONFIG_CPU_SUBTYPE_SH7707 | SH7750 |
| CONFIG_CPU_LITTLE_ENDIAN | * |
| CONFIG_NET | * |
| CONFIG_PCI | * |
| CONFIG_SYSVIPC | * |
| CONFIG_SYSCTL | * |
| CONFIG_KCORE_ELF | (ELF) |
| CONFIG_BINFMT_ELF | * |
| CONFIG_MTD | M |
| CONFIG_MTD_BLOCK | M |
| CONFIG_MTD_VMU | M |
| CONFIG_BLK_DEV_LOOP | * |
| CONFIG_BLK_DEV_RAM | * |
| CONFIG_BLK_DEV_SIZE | (4096) |
| CONFIG_BLD_DEV_INITRD | * |
| CONFIG_PACKET | * |
| CONFIG_NETFILTER | * |
| CONFIG_UNIX | * |
| CONFIG_INET | * |
| CONFIG_IP_MULTICAST | * |
| CONFIG_IP_NF_CONNTRACK | M |
| CONFIG_IP_NF_FTP | M |

| | |
|---|---|
| CONFIG_IP_NF_IRC | M |
| CONFIG_IP_NF_IPTABLES | M |
| CONFIG_IP_NF_MATCH_LIMIT | M |
| CONFIG_IP_NF_MATCH_STATE | M |
| CONFIG_IP_NF_FILTER | M |
| CONFIG_IP_NF_NAT | M |
| CONFIG_IP_NF_TARGET_MASQUERADE | M |
| CONFIG_IP_NF_TARGET_LOG | M |
| CONFIG_NETDEVICES | * |
| CONFIG_NET_ISA | * |
| CONFIG_LAN_ADAPTER | M |
| CONFIG_NET_PCI | * |
| CONFIG_8139TOO | M |
| CONFIG_8139TOO_DREAMCAST | * |
| CONFIG_PPP | * |
| CONFIG_PPP_ASYNC | * |
| CONFIG_PPP_SYNC_TTY | * |
| CONFIG_SLIP | * |
| CONFIG_SLIP_COMPRESSED | * |
| CONFIG_CD_NO_IDESCSI | * |
| CONFIG_SEGA_GDROM | * |
| CONFIG_INPUT | * |
| CONFIG_INPUT_KEYBDEV | * |
| CONFIG_MAPLE | * |
| CONFIG_VT | * |
| CONFIG_VT_CONSOLE | * |
| CONFIG_SH_SCI | * |
| CONFIG_SERIAL_CONSOLE | * |
| CONFIG_UNIX98_PTYS | * |
| CONFIG_UNIX98_PTY_COUNT | (256) |
| CONFIG_MAPLE_KEYBOARD | * |
| CONFIG_INPUT_MAPLE_CONTROL | * |
| CONFIG_MINIX_FS | * |
| CONFIG_PROC_FS | * |
| CONFIG_DEVPTS_FS | * |
| CONFIG_EXT2_FS | * |
| CONFIG_FB | * |
| CONFIG_FB_PVR2 | * |
| (CONFIG_SOUND | *) |
| (CONFIG_SOUND_AICA | *) |

Table 2.2: Sample configuration for an ADSL software router.

We have now a Linux kernel configuration *.config* and some other files set up by the make command mentioned above.

## 2.6 Building a runtime library

This section describes the necessary steps for building a C - library. This library contains many essential functions such as `printf()` for C - sources:

```
$ cd ../..
$ cd BUILD
$ tar -xvzf ../SRC/glibc-2.2.4.tar.gz -C .
$ patch -p0 < ../SRC/glibc-2.2.4-sh-linux.diff
$ mkdir build-glibc
$ cd build-glibc
$ mkdir $PREFIX/$TARGET/include
```

Now, we copy the Linux kernel header files and assembly files to this include directory:

```
$ cp -r ../../KERNEL/linux/include/linux \
  $PREFIX/$TARGET/include
$ cp -r ../../KERNEL/linux/include/asm-sh \
  $PREFIX/$TARGET/include/asm
```

Now, we're ready to configure and compile this C - library. The *configure* - script configures the sources for the *SEGA Dreamcast*, looks for kernel header files in the directory `$PREFIX/$TARGET/include`, dispenses with profiling informationen build in along with debug information and disables examinations of the build programs (you remeber chapter one, common cross compiling problems?):

```
$ CC=sh4-linux-gcc ../glibc-2.2.4/configure \
  --host=$TARGET --prefix=$PREFIX \
  --disable-debug --disable-profile \
  --disable-sanity-checks \
  --with-headers=$PREFIX/$TARGET/include
$ make
```

The next command saves time by bluffing the glibc of having already build unnecessary programs:

```
$ touch iconv/iconv_prog login/pt_chown
$ make install_root=$PREFIX/$TARGET prefix="" install
$ echo "GROUP ( libc.so.6 libc_nonshared.a )" \
  > $PREFIX/$TARGET/lib/libc.so
```

## 2.7 Rebuilding the *cross compiler*

The last necessary step for building the Linux kernel or any other application is the rebuild of the intrinsic compiler. Just change directory und rebuild the whole *gcc-3.0.1* source package:

```
$ cd ..
$ mkdir build-gcc2
$ ../gcc-3.0.1/configure --target=$TARGET \
  --prefix=$PREFIX --enable-languages=c,c++
$ make all install
```

That's all. We have now a complete C - cross compiler.

## 2.8 Building the Linux kernel

Now, we're ready to compile the heart of a Linux distribution: the Linux kernel.

```
$ cd ../..
$ cd KERNEL/linux
$ make ARCH=sh CROSS_COMPILE=sh4-linux- clean dep zImage modules
$ cd ../..
```

You wonder, why we don't complete the Linux kernel build with `modules_install`? The answer is simple: We don't have a target where the modules might go...

So, we round out this build in a later stage, be patient.

From this point, you've got two possibilities to go on. Either, you read on, build a first application and load the whole stuff to your *SEGA Dreamcast* or, you skip the next sections and go directly to section *Building the bootloader*.

I suppose most readers want to see the fruits of their work and thus they read on :-)

## 2.9 Building a shell (and a little bit more...)

This section completes your work and glues everything together.

At first, we need a basic directory structure, containing the binaries. We orientate us to [FHS01], but at this point, we only use some of the proposed directories. This basic directory structure including the binaries will be called *initrd* or *initial ramdisk* because it'll be loaded by the Linux kernel itself directly during the boot process. So, we archive this directory in a later stage into one single file, which we glue with the Linux kernel:

```
$ mkdir INITRD
$ export INITRD=‘pwd‘/INITRD
$ cd INITRD
$ mkdir -p proc dev/pts etc
$ cd ..
```

The `/proc` - directory help us for gathering basic and extend system information. The `/dev` contains all necessary device nodes. Now, we’re building the first application for the *SEGA Dreamcast*:

```
$ cd BUILD
$ tar -xvzf ../SRC/busybox-0.60.5.tar.gz -C .
$ cd busybox-0.60.5
```

For compiling `busybox-0.60.5`, you have to configure your needs in the file `Config.h`. Here’s my proposal:

```
#define BB_BASENAME
#define BB_CAT
#define BB_CHGRP
#define BB_CHMOD
#define BB_CHOWN
#define BB_CLEAR
#define BB_CP
#define BB_DATE
#define BB_DF
#define BB_DIRNAME
#define BB_DMESG
#define BB_DU
#define BB_ECHO
#define BB_FREE
#define BB_GREP
#define BB_HALT
#define BB_HEAD
#define BB_HOSTID
#define BB_HOSTNAME
#define BB_IFCONFIG
#define BB_INIT
#define BB_INSMOD
#define BB_KILL
#define BB_KILLALL
#define BB_KLOGD
#define BB_LOADKMAP
#define BB_LN
#define BB_LOGGER
#define BB_LS
```

```
#define BB_LSMOD
#define BB_MKDIR
#define BB_MKFS_MINIX
#define BB_MODPROBE
#define BB_MORE
#define BB_MOUNT
#define BB_MSH
#define BB_MV
#define BB_PIDOF
#define BB_PING
#define BB_PS
#define BB_PWD
#define BB_REBOOT
#define BB_RESET
#define BB_RM
#define BB_RMDIR
#define BB_RMMOD
#define BB_ROUTE
#define BB_SED
#define BB_SLEEP
#define BB_SYNC
#define BB_SYSLOGD
#define BB_TAIL
#define BB_TAR
#define BB_TEST
#define BB_TIME
#define BB_TOUCH
#define BB_TRACEROUTE
#define BB_TRUE_FALSE
#define BB_TTY
#define BB_UMOUNT
#define BB_UNAME
#define BB_UPTIME
#define BB_VI
#define BB_WGET
#define BB_WHICH
#define BB_WHOAMI
#define BB_YES

#define BB_FEATURE_SH_IS_MSH
#define BB_FEATURE_VERBOSE_USAGE
#define BB_FEATURE_AUTOWIDTH
#define BB_FEATURE_LS_USERNAME
#define BB_FEATURE_LS_TIMESTAMPS
#define BB_FEATURE_LS_FILETYPES
#define BB_FEATURE_LS_SORTFILES
#define BB_FEATURE_LS_RECURSIVE
```

```
#define BB_FEATURE_LS_FOLLOWLINKS
#define BB_FEATURE_LS_COLOR
#define BB_FEATURE_FANCY_PING
#define BB_FEATURE_USE_INITTAB
#define BB_FEATURE_LINUXRC
#define BB_FEATURE_REMOTE_LOG
#define BB_FEATURE_FANCY_TAIL
#define BB_FEATURE_MOUNT_FORCE
#define BB_FEATURE_TAR_CREATE
#define BB_FEATURE_TAR_EXCLUDE
#define BB_FEATURE_SORT_REVERSE
#define BB_FEATURE_SORT_UNIQUE
#define BB_FEATURE_COMMAND_EDITING
#define BB_FEATURE_COMMAND_TAB_COMPLETION
#define BB_FEATURE_SH_FANCY_PROMPT
#define BB_FEATURE_ASH_JOB_CONTROL
#define BB_FEATURE_NEW_MODULE_INTERFACE
#define BB_FEATURE_IFCONFIG_STATUS
#define BB_FEATURE_IFCONFIG_SLIP
#define BB_FEATURE_WGET_STATUSBAR
#define BB_FEATURE_WGET_AUTHENTICATION
#define BB_FEATURE_HUMAN_READABLE
#define BB_FEATURE_FIND_TYPE
#define BB_FEATURE_FIND_PERM
#define BB_FEATURE_FIND_MTIME
#define BB_FEATURE_FIND_NEWER
#define BB_FEATURE_TFTP_PUT
#define BB_FEATURE_TFTP_GET
#define BB_FEATURE_VI_COLON
#define BB_FEATURE_VI_YANKMARK
#define BB_FEATURE_VI_SEARCH
#define BB_FEATURE_VI_USE_SIGNALS
#define BB_FEATURE_VI_DOT_CMD
#define BB_FEATURE_VI_READONLY
#define BB_FEATURE_VI_SETOPTS
#define BB_FEATURE_VI_SET
#define BB_FEATURE_VI_WIN_RESIZE
#define BB_FEATURE_TELNET_TTYPE
```

Everything else found as options in this file might be disabled, using following commenting style:

```
//#define BB_FEATURE_EXTRA_QUIET
```

Now, it’s time for compiling:

```
$ make CROSS=sh4-linux- DOSTATIC=true \
  CFLAGS_EXTRA="-I $PREFIX/$TARGET/include" \
  PREFIX=$INITRD clean all install
```

With the command above, we force `make` into building a *static* program since we don't a dynamic linker let alone a shared C - library on our target platform.

Now, two things are absent: The device nodes and the kernel modules. So, we correct these points by:

```
$ cd ../..
$ cd KERNEL/linux
$ su -c "make ARCH=sh CROSS_COMPILE=sh4-linux- \
  INSTALL_MOD_PATH=/home/christian/Dreamcast/\
  INITRD modules_install"
```

Don't be disturbed by the error message at the end: We don't have a `depmod` utility that handles modules for foreign hardware architectures, so simply ignore it.

Now, change directory to the initial ramdisk and correct the libraries folder:

```
$ cd ../..
$ cd INITRD
$ cd lib/modules/2.4.18-sh-dc
```

Remove every file and directory except the directory *kernel* - it contains our kernel modules.

Following, we create the essential device nodes and the *inittab*:

```
$ cd ../../..
$ cd dev
$ su -c "mknod console c 5 1"
$ cd ..
$ cd etc
```

Edit the (not existing) file `inittab`:

```
# Starts an askfirst shell:

::askfirst:-/bin/sh
```

Finally, leave the initial ramdisk:

```
$ cd ../..
```

## 2.10 Bundle the Linux kernel and the initrd

Now, we are ready to collect everything we've just build and carry that bundle to our *SEGA Dreamcast*. So, we create a single file that will contain the content of the directory `$INITRD`.

But firstly, we have to `chown` everything to root:root for avoiding error messages during the boot process.

```
$ su
# cd INITRD
# chown -R 0.0 *
# cd ..
# dd if=/dev/zero of=initrd.img bs=1k count=4096
# mke2fs -F -v -m0 initrd.img
# mkdir initrd.DIR
# mount -o loop initrd.img initrd.DIR
# (cd INITRD ; tar -cf - .) | (cd initrd.DIR ; tar -xvf -)
# umount initrd.DIR
# gzip -c -9 initrd.img > initrd.bin
# exit
```

The command `mke2fs -f -vm0 initrd.img` forces the formatting process even if we're using a file instead of a block device. The option `-m0` prohibits the creation of the reserved space for the super user. Now, you've archived everything into one single file.

## 2.11 Building the bootloader

Everything's ready to get glued together. The only thing we need is a bootloader that helps us to get our stuff into the memory and execute it:

```
$ cd BUILD
$ tar -xvzf ../SRC/sh-boot-20010831-1455.tar.gz -C .
$ patch -p0 < ../SRC/sh-boot-20010831-1455.diff
```

The next steps create a file kernel-boot.bin that contains the Linux kernel and the initial ramdisk. *So, these steps have to be done every time, you change either the Linux kernel or something within the initial ramdisk (e.g. copy a new program to it or modify the configuration)*:

```
$ cd sh-boot/tools/dreamcast
$ cp ../../../../KERNEL/linux/arch/sh/boot/zImage \
  ./zImage.bin
$ cp ../../../../initrd.bin .
$ make scramble kernel-boot.bin
```

Now, you've got two possibilities.

- Either you burn a CD-R (the *SEGA Dreamcast* can't read CD-RW without any modification) by the following commands (probably as root, depends on your system configuration):

  Edit the file `roast.sh` and change the value CDRECORD for your needs (i.e. the SCSI device id). Afterwards, insert a CD-R and burn *your first embedded Linux distribution* by typing:

  ```
  $ ./roast.sh kernel-boot.bin
  ```

  Insert the disc in your *SEGA Dreamcast* and hit the power button. Hopefully, you'll see the Linux-SH penguin in the upper left corner of your screen while the *SEGA Dreamcast* waits for your login.

- Or you've got the Coder's Cable. Then, you can save money and transfer the image through a *serial loader*.

  First, you have to download the server and client tools for enjoying saving money.

  ```
  $ cd ../../../../SRC
  $ wget http://adk.napalm-x.com/dc/\
    dcload-serial/dcload-1.0.3-1st_read.zip
  $ wget http://adk.napalm-x.com/dc/\
    dcload-serial/dc-tool-serial-1.0.3-linux.gz
  $ gunzip dc-tool-serial-1.0.3-linux.gz
  $ mkdir ../LOADER
  $ mv dc-tool-serial-1.0.3-linux ../LOADER
  $ chmod u+x ../LOADER/dc-tool-serial-1.0.3-linux
  $ mkdir ../BUILD/server.disc
  $ unzip dcload-1.0.3-1st_read.zip -d ../BUILD/server.disc
  $ cd ../BUILD/server.disc
  ```

  Now, you have to create the server disc. Following Marcus Comstedt's HowTo [Comstedt00], a bootable CD-R for the *SEGA Dreamcast* should have two sessions. The first should contain only a normal audio track. The second session should contain a CD/XA data track (mode 2 form 1). This data track ought to contain a regular ISO9660 file system, and in the first 16 sectors a correct bootstrap (IP.BIN).

  First you have to burn the audio session. The simplest option is just to create 4 seconds (the minimum track length) of silence, like:

  ```
  $ dd if=/dev/zero bs=2352 count=300 of=audio.raw
  ```

Next, insert a blank CD-R (the *SEGA Dreamcast* can't read CD-RW without any modification) and burn the audio track. Make sure that you leave the disc open for further sessions, the *-multi* option to cdrecord takes care of that:

```
$ cdrecord dev=0,1,0 -multi -audio audio.raw
```

Please take care that you're allowed to burn the disc and that you're using the right device. Now that the audio track has been burned, it is possible to create the ISO filesystem image. But first, you have to find out the offset where you can start off the data track. To find out this number, run

```
$ cdrecord dev=0,1,0 -msinfo
```

with the disc still in the drive. You should get two numbers separated by comma (for example 0,11700). Remember these numbers. Now create the ISO image with mkisofs:

```
$ mkisofs -l -C x,y -o tmp.iso \
  ./dcload-1.0.3-1st_read/scrambled/1st_read.bin
```

where x,y is the pair of numbers you got with *-msinfo* earlier. Make sure you get them correctly, or the image won't work.

The first 16 sectors of an ISO9660 filesystem are blank, to leave room for bootstraps. This is where IP.BIN (32768 bytes) goes. So, we glue the *tmp.iso* together with the IP.BIN file:

```
$ ( cat ./dcload-1.0.3-1st_read/IP.BIN ; \
  dd if=tmp.iso bs=2048 skip=16 ) > data.raw
```

Finally, you're ready to burn the second session and complete the disc. This track has to be burned as CD/XA with form 1 sectors (2048 bytes per sector). Use the *-xa1* option to cdrecord:

```
$ cdrecord dev=0,1,0 -multi -xa1 data.raw
```

Now, you only have to insert your *serial loader server disc* into your dreamcast and hit the power button. After a while, you'll see the *idle...* prompt. Now, on your host, you have to invoke the loading process by typing:

```
$ cd ../../LOADER
$ ./dc-tool-serial-1.0.3-linux\
  -t /dev/ttyS1 -e -b 115200 -x \
  ../BUILD/sh-boot/tools/dreamcast/kernel-boot.bin
```

Please note, that you probably have to change the serial port `/dev/ttyS?`. Wait a moment and, hopefully, you'll see your *SEGA Dreamcast* is booting your Linux kernel and your initial ramdisk.

*Congratulations*. You overcome the first steps.

## 2.12 Where are we?

Let's summarize what we've done already. We've downloaded the *GNU Compiler Collection* with all needed tools. Then, we've downloaded and plurally patched the Linux kernel 2.4.18. Following, we've build a statically linked application *busybox* (size is about 789 KB) and created a first *initial ramdisk*.

Finally, we've completed the build with the bootloader.

We have right now the tools by the hand we need for compiling and running applications. *But* do you notice the size of the application? It's about 789 KB (up to your setup). Imagine, we compile *pppd*, *pppoe*, *iptables*, *sshd*,... in the same way like *busybox*. Unfortunately, we run out of memory. So, we have to consider an essential thing: We have to get apart from the "big monster" *glibc* and take a leightweight C - library.

This work will be done in the next chapters. First, we discuss a generally directory layout for our *initial ramdisk*. Then, we compile and install the *uClibc*, a minimalistic C - library for our needs. And finally, we try to compile and install the programs, needed by an ADSL software router with firewalling and virtual private networking.

# 3 Setting up an initial ramdisk

Assuming that you have read through the first two chapters, we're ready to set up the layout of our final initial ramdisk. We try to be as close as possible on the proposals of the *Filesystem Hierarchy Standard* [FHS01].

## 3.1 Directory structure

As seen in chapter two, the `make install` of *busybox* has created some directories containing dozens of symlinks. We've only created two directories: */dev/pts* and */proc*.

Now, it's time to set up all the other needed directories.

```
$ cd /home/christian/Dreamcast
$ cd INITRD
$ ls
bin dev etc linuxrc proc sbin usr
$
```

The Filesystem Hierarchy Standard proposes the following directories:

- `/bin` *Essential user command binaries (for use by all users).* For example, our `/bin` contains `cp`, `df`, `rm`,...
- `/boot` *Static files of the boot loader.* We don't need this directory due to the use of a sh-specific, external bootloader.
- `/dev` *Device files.* Here, we create all device nodes, we need to communicate with the kernel.
- `/etc` *Host-specific system configuration.* This folder will contain the configuration files, such as `passwd`, `inittab` among others.
- `/home` *User home directories (optional).* We'll create this directory "on the fly", i.e. if the user plugs in the *VMU*.
- `/lib` *Essential shared libraries and kernel modules.* This directory will only contain the kernel modules caused by the lack of a *dynamic loader* for dynamic libraries.
- `/mnt` *Mount point for a temporarily mounted filesystem.* We need this folder for mounting the *VMU*.

- `/opt` *Add-on application software packages.* We don't need this folder.
- `/root` *Home directory for the root user (optional).* We create this directory for the super user.
- `/sbin` *System binaries.* This directory contains essential system software, such as `insmod`, `ifconfig` and others.
- `/tmp` *Temporary files.*
- `/usr/bin` *Most user commands.* This Folder contains `cut`, `du`,...
- `/usr/sbin` *Non-essential standard system binaries.* This folder will contain `sshd` and other service tools.
- `/var/lib` *Variable state information.* This directory will contain files about *DHCP*.
- `/var/log` *Log files and directories.* This folder contains logging information.
- `/var/run` *Run-time variable data.* Here, some process create some kind of "cookies", often their *process id*.

As you notice, we don't cover all proposed directories due to the limited amount of free inodes.

Let's go and create all needed directories and configuration files. Remember, that you've `chown`'ed all files to the super user, so change your user userid to the super user:

```
$ su
#
```

1. `/dev`

   First, we create the device nodes and set up their mode and ownership:

   ```
   # cd dev

   # chmod 600 console
   # chown 0.1 console

   # mknod gdrom0 b 250 0
   # chmod 640 gdrom0
   # chown 0.2 gdrom0

   # mknod mtdblock0 b 31 0
   # chmod 660 mtdblock0
   # chown 0.2 mtdblock0
   ```

```
# mknod null c 1 3
# chmod 666 null
# chown 0.0 null

# mknod ppp c 108 0
# chmod 660 ppp
# chown 0.3 ppp

# mknod ptmx c 5 2
# chmod 666 ptmx
# chown 0.1 ptmx

# mknod random c 1 8
# chmod 666 random
# chown 0.0 random

# mknod tty c 5 0
# chmod 600 tty
# chown 0.0 tty

# mknod tty0 c 4 0
# chmod 600 tty0
# chown 0.0 tty0

# mknod tty1 c 4 1
# chmod 600 tty1
# chown 0.0 tty1

# mknod tty2 c 4 2
# chmod 600 tty2
# chown 0.0 tty2

# mknod tty3 c 4 3
# chmod 600 tty3
# chown 0.0 tty3

# mknod tty4 c 4 4
# chmod 600 tty4
# chown 0.0 tty4

# mknod ttySC0 c 204 8
# chmod 600 ttySC0
# chown 0.3 ttySC0

# mknod ttySC1 c 204 9
# chmod 600 ttySC1
# chown 0.3 ttySC1
```

```
# mknod urandom c 1 9
# chmod 444 urandom
# chown 0.0 urandom

# mknod zero c 1 5
# chmod 666 zero
# chown 0.0 zero

# cd ..
```

As you notice, we've set different group ids. These ids have to correspond with our entries in `/etc/group` later.

2. `/etc`

   This directory comprises the configuration files. So, be careful with your entries and chmodes!

```
# cd etc

# cat <<. >group
root:x:0:
tty:x:1:
disk:x:2:
dip:x:3:
shadow:x:4:
.
# chmod 644 group

# cat <<. >gshadow
root:*::
tty:!::
disk:!::
dip:!::
shadow:!::
.
# chmod 640 gshadow
# chown 0.4 gshadow

# cat <<. >hosts
127.0.0.1       localhost
192.168.1.1     dreamcast.localdomain  dreamcast
.
# chmod 640 hosts

# mkdir init.d
# cd init.d
```

```
cat <<. > rcS
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

echo "Mounting /proc filesystem..."
mount -t proc none /proc

echo "Mounting /dev/pts filesystem..."
mount -t devpts none /dev/pts

echo "Loading eth0 module..."
modprobe lan_adapter
modprobe mii
modprobe 8139too

echo "Loading mtd modules..."
modprobe mtdcore          2>&1 > /dev/null
modprobe mtdblock         2>&1 > /dev/null
modprobe chipreg          2>&1 > /dev/null
modprobe vmu-flash        2>&1 > /dev/null

echo "Loading netfilter modules..."
for mod in ip_conntrack ip_conntrack_ftp \
 ip_conntrack_irc ip_tables iptable_filter \
 iptable_nat ipt_limit ipt_state ip_nat_ftp \
 ip_nat_irc ipt_LOG ipt_MASQUERADE;
do
   modprobe $mod 2>&1 > /dev/null
done

echo "Setting up routing..."
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward

echo "Setting hostname..."
hostname dreamcast

echo "Configuring lo..."
ifconfig lo 127.0.0.1 up

echo "Configuring eth0..."
ifconfig eth0 192.168.1.1 up

echo "Starting klogd..."
klogd
```

```
echo "Starting syslogd..."
syslogd

#echo "Starting udhcpd..."
#udhcpd

#echo "Starting yaku-ns..."
#/usr/yaku-ns/yaku-ns -c \
# /usr/yaku-ns/yaku-ns.conf -l \
# /usr/yaku-ns/yaku-ns.log -u yaku -d

#echo "Starting sshd..."
#sshd -f /etc/ssh/sshd_config

/bin/sleep 1

echo "Everything's done. Have fun."
.
# chmod 744 rcS
```

The file `/etc/init.d/rcS` starts all initial services. As you notice, some services are disabled by # because we haven't compiled them yet.

```
# cd ..
# cat <<. >inittab
# Initial startup file

::once:/etc/init.d/rcS

# Start four ask first login prompts
tty1::askfirst:/bin/login
tty2::askfirst:/bin/login
tty3::askfirst:/bin/login
tty4::askfirst:/bin/login

::ctrlaltdel:/sbin/reboot

.
# chmod 644 inittab

# cat <<. >passwd
root:x:0:0:root:/root:/bin/sh
.
# chmod 644 passwd

# cat <<. >shadow
root:*:12091:0:99999:7:::
```

```
nobody:*:12091:0:99999:7:::
.
# chmod 640 shadow
# chown 0.4 shadow
```

The `shadow` and `passwd` contain the usernames and passwords. If you need the same usernames and passwords like your host, just copy the files `/etc/passwd`, `/etc/shadow`, `/etc/gshadow` and `/etc/group` to your initial ramdisk.

```
# mkdir ppp

# touch resolv.conf
# chmod 644 resolv.conf

# cat <<. >services
tcpmux      1/tcp              # TCP port service multiplexer
echo        7/tcp
echo        7/udp
ftp-data    20/tcp
ftp         21/tcp
fsp         21/udp  fspd
ssh         22/tcp             # SSH Remote Login Protocol
ssh         22/udp             # SSH Remote Login Protocol
telnet      23/tcp
smtp        25/tcp  mail
domain      53/tcp  nameserver # name-domain server
domain      53/udp  nameserver
www         80/tcp  http       # WorldWideWeb HTTP
www         80/udp             # HyperText Transfer Protocol
pop3        110/tcp pop-3      # POP version 3
pop3        110/udp pop-3
ntp         123/tcp
ntp         123/udp            # Network Time Protocol

.
# chmod 644 services

# mkdir ssh

# cd ..
```

Now, you have a correct configure `/etc` - directory.

3. `/home`

```
# mkdir /home
```

4. `/mnt`

   ```
   # mkdir /mnt
   ```

5. `/root`

   ```
   # mkdir root
   # chmod 700 root
   ```

6. `/tmp`

   ```
   # mkdir /tmp
   ```

7. `/var/lib`

   ```
   # mkdir -p /var/lib
   ```

8. `/var/log`

   ```
   # mkdir /var/log
   ```

9. `/var/run`

   ```
   # mkdir /var/run
   ```

Now, you've got a proper set up initial ramdisk directory layout.

## 3.2 Where are we?

Our initial ramdisk looks a little bit like a *normal* distribution. We've layed the directory structure out and created essential configuration files.

Now, it's time for compiling the new C - library *uClibc* for saving more space on our *SEGA Dreamcast*. Following, we have to recompile `busybox-0.60.5` *against uClib*, i.e. with the new C - library, for shrinking its size. Then, we compile `tinylogin` for allowing that more than one user may login at the same time.

# 4 Setting up the *uClibc* toolchain with two essential applications

In this chapter, you'll set up a *new* toolchain based on the *uClibc* C - library. You wonder, why we're setting up a new one? The simple reason is: The size of *busybox* is now about 789 KB. If you compile *busybox* against the new C - library, its size will shrink to about 290 KB. You save valuable 500 KB! In this and the next chapters, every program will be linked against this new C - library. With this technique, we maximize the functionality of our software router while controlling the growth of the single programs.

## 4.1 Setting up *uClibc*

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `uClibc-0.9.19` | 1.4 MB | LGPLv2 | http://www.uclibc.org/downloads/uClibc-0.9.19.tar.bz2 |

Table 4.1: Source package, size and URL for *uclibc*.

I'm assuming, you'll collect the package sources under `~/Dreamcast/SRC`, so we can unpack the archive with:

```
$ cd Dreamcast/BUILD
$ bunzip2 -c ../SRC/uClibc-0.9.19.tar.bz2 | \
  tar -xv -C .
$ cd uClibc-0.9.19
```

Now, we have to set up some environment variables for getting the correct options in the configuration menu:

```
$ export TARGET_ARCH=sh
$ export NATIVE_CC=gcc
$ export CROSS=sh4-linux-
$ export KERNEL_SOURCE=/home/christian/Dreamcast\
  /KERNEL/linux
```

If you've logged out meanwhile, you have to set up the correct `$PATH` and `$PREFIX`environment settings:

```
$ export PREFIX=/home/christian/Dreamcast\
  /toolchain
$ export PATH=$PREFIX/bin:$PATH
```

Now, we're ready for setting up the *Makefile* for *uClibc*:

```
$ make menuconfig
```

Please select the following options for running programs compiled against *uClibc* on your *SEGA Dreamcast*:

| **Option** | **Type** |
|---|---|
| Target Processor Type | SH4 |
| Target Processor Endianness | Little Endian |
| Target CPU has MMU | * |
| Enable floating point number support | * |
| Please check the correct location of the kernel headers. | |
| POSIX Threading Support | * |
| Malloc Implementation | malloc |
| Shadow Password Support | * |
| Regular Expression Support | * |
| Support only Unix 98 PTYs | * |
| Assume that /dev/pts is a devpts or devfs filesystem | * |
| Remote Procedure Call (RPC) support | * |
| Full RPC support | * |

Table 4.2: Configuration for *uClibc* for running on *SEGA Dreamcast*.

Please check also that the option *uClibc development environment* is set to:

`$PREFIX/$(TARGET)-linux-uclibc`

Any option not mentioned in table 4.2 has to be unselected. Now it's time for compiling and installing:

```
$ make all
$ su -c "make install"
$ cd ..
```

You should have a new toolchain located under:

```
$PREFIX/sh-linux-uclibc/*
```

These last steps are necessary for compiling the following sources without any errors:

```
$ unset TARGET_ARCH
$ unset NATIVE_CC
$ unset CROSS
$ unset KERNEL_SOURCE
$ export UCLIBC=/home/Dreamcast\
  /toolchain/sh-linux-uclibc
```

Next, it's time for testing our new toolchain.

## 4.2 Compiling *busybox-0.60.5*

I don't explain again, why we re-compile the famous package, we just do it :-) Please assure that you're in the correct folder *~/Dreamcast/BUILD*:

```
$ cd busybox-0.60.5
```

First, we clean up the directory from any old *make* pass:

```
$ make clean
```

Now, we're able to re-compile *busybox* against *uClibc*. Please make your choice for any user programs you want *busybox* to have in its configuration file `Config.h` or simply take the ones I've given in chapter two. First, we have to correct a setting in the *Makefile* for using the correct compiler (in our case the *cc - wrapper* of *uClibc*):

```
$ cat Makefile | sed s/CC\ =\ \$\(CROSS\)gcc/\
  CC\ =\ \$\(CROSS\)cc/ > Makefile.new
$ rm -f Makefile && mv Makefile.new Makefile
```

Following, just invoke the compiling process:

```
$ make CROSS=$UCLIBC/usr/bin/ all
$ su -c "make PREFIX=/home/christian\
  /Dreamcast/INITRD install"
$ cd ..
```

That's all! Please compare, if it's possible, the actual size of *busybox* with its old one. You'll probably determine it has shrunken under about 300 KB.

## 4.3 Compiling *tinylogin-snapshot*

*tinylogin* is a similar program to *busybox*: It also bundles some essential tools found in *normal* Linux distribution in one single file and creates tons of symbolic links. *Tinylogin* provides for example `login` or `su`.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `tinylogin-snapshot` | 91 KB | GPLv2 | http://tinylogin.busybox.net/downloads/snapshots/tinylogin-snapshot.tar.bz2 |

Table 4.3: Source package, size and URL for *tinylogin*.

Now, unpack the downloaded archive:

```
$ bunzip2 -c ../SRC/tinylogin-snapshot.tar.bz2 | \
  tar -xv -C .
$ cd tinylogin
```

And finally, compile and install the binaries:

```
$ make CROSS=$UCLIBC/usr/bin/ all
$ su -c "make PREFIX=/home/christian\
  /Dreamcast/INITRD install"
$ cd ..
```

That's all. Easy, huh? But, believe me, tinylogin is at least the problematic source at all. I'm very surprised that I don't have to modify its *Makefile* or patch something. But, why should problems always occur?

## 4.4 Setting up *the cook*

This section describes the creation of a short script that helps us creating the file we can transfer to our *SEGA Dreamcast*. Remeber, you always have to do the same many steps for building this file. So, we just bundle these commands in one single script:

```
$ cat <<. > cook.sh
#!/bin/sh

OLDPW=$(pwd)

cd ~/Dreamcast
```

```
dd if=/dev/zero of=initrd.img bs=1k count=4096
mke2fs -F -vm0 initrd.img
mount -o loop initrd.img initrd.DIR

(cd INITRD ; tar -cvf - .) | (cd initrd.DIR ; tar xvf -)

umount initrd.DIR

gzip -c -9 initrd.img > initrd.bin

cp ./KERNEL/linux/arch/sh/boot/zImage \
./BUILD/sh-boot/tools/dreamcast/zImage.bin

cp initrd.bin ./BUILD/sh-boot/tools/dreamcast

cd ./BUILD/sh-boot/tools/dreamcast
make clean scramble kernel-boot.bin

mv kernel-boot.bin kernel-boot-\
.‘date +"%d.%m.%Y-%X"‘-.bin

cd $OLDPWD
.
$ chmod u+x cook.sh
```

Consider, you run this script as the *super user* due to the use of privileged commands in this script:

```
$ su -c "./cook.sh"
```

The last line in this script creates for every file being transferred to the *SEGA Dreamcast* an own time stamp. That helps, believe me :-)

## 4.5 Where are we?

Our own Linux distribution takes shape: We’ve a resonable C - library for compiling all the programs we need and keep their size small at the same time. Next, we can handle multiple logins to our *SEGA Dreamcast* due to the use of *tinylogin*. Last but not least, we’ve created a small helper script for packing all our work in one single file to execute on the *SEGA Dreamcast*.

# 5 Connecting the *SEGA Dreamcast* to an access concentrator

This chapter describes the steps for connecting the *SEGA Dreamcast* to an access concentrator necessary for using ADSL with our gaming console. Therefore, we compile and configure the source packages `ppp-2.4.1` and `rp-pppoed-3.3`. At the end of this chapter, you'll be able to "surf" the Internet - newly :-) Unfortunately, we don't have any browser, so we just "ping" some hosts or may download some software with "wget". But that stage is absolutely necessary to establish the next steps to complete a router with a firewall.

## 5.1 Some theory

*Why do we need two programs for connecting to the Internet?*

Normally, if you *google* for a HowTo that helps you to connect a personal computer with the Internet over ADSL, you are proposed to use two ethernet networking adapters if you wish to act as an gateway for a small home network. But our *SEGA Dreamcast* only has one single extension slot. So, where should we take this second adapter?

But, RFC 2516 [RFC2516] helps us. That Request for Comments describes how to build PPP sessions and encapsulate PPP packets over Ethernet. That memo describes the theoretical backgrounds necessary for implementing such a protocol. And even that is the key for us. *PPPoE* is "only" a protocol. Thus, nothing contradicts the use of only one adapter due to the use of multiple protocols assigned to that adapter. We just load multiple protocols for this adapter: The *TCP/IP* and the *PPPoE* protocol.

It's up to the protocol and the Linux kernel to distinguish "simple" TCP/IP - packets from PPPoE packets (see figure 5.1). We cause a lot of traffic on that adapter and in the LAN, but ADSL has only a limited downstream of 768 kbps (about 90 kBytes/s), so, those carry no weight.

## 5.2 Compiling *ppp-2.4.1*

As seen in the previous section, we need *ppp* for transferring data packets from one point to another point. An implemention of this protocol is the one, we compile in this section. Additionally, we need a patch for getting the source compiled.

But first, we need to download the appropriate source package:

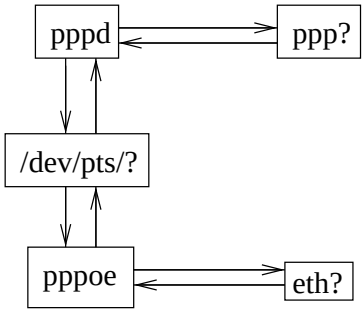


Figure 5.1: Diagram of the (virtual) devices while PPPoE'ing.

| Package | Size | License | URL |
|---|---|---|---|
| `ppp-2.4.1` | 524 KB | BSD/GPL | ftp://cs.anu.edu.au/pub/software/ppp/ppp-2.4.1.tar.gz |

Table 5.1: Source package, size and URL for *ppp*.

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  ppp-2.4.1-uclibc-dreamcast.patch
```

The patch corrects some problems with the `strip` command. This program is used for removing some information needed for debugging a program. Since we need really small programs, we just remove such sections.

Now, we're ready to compile `ppp-2.4.1`:

```
$ cd ../BUILD
$ tar -xvzf ../SRC/ppp-2.4.1.tar.gz -C .
$ cd ppp-2.4.1
$ patch -p1 < ../../SRC/\
  ppp-2.4.1-uclibc-dreamcast.patch
$ ./configure
$ make CROSS=$UCLIBC/usr/bin/ all
$ su -c "make DESTDIR=/home/christian/Dreamcast/\
  INITRD install"
$ cd ../../SRC
```

If you really want to save space, you can delete the programs `pppstats` and `pppdump` from your initial ramdisk. Both programs are needed for debugging a connection or for billing information.

## 5.3 Compiling *rp-pppoe-3.3*

The last source package for an ADSL connection is the implementation of RFC 2516 [RFC2516]. Several implementations of this protocol exist, therefore I've decided to take `rp-pppoe-3.3` because of the recommendation found in the documentation of *uClibc*.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `rp-pppoe-3.3` | 167 KB | GPLv2 | http://www.roaringpenguin.com/pppoe/rp-pppoe-3.3.tar.gz |

Table 5.2: Source package, size and URL for *rp-pppoe*.

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  rp-pppoed-3.3-uclibc-dreamcast.patch
```

The patch creates a script `adsl-connect` with some specific options. Besides, the patch also corrects some problems with the `strip` command and the setting of some environment variables.

Now, we're ready to compile `rp-pppoe-3.3`:

```
$ cd ../BUILD
$ tar -xvzf ../SRC/rp-ppp-3.3.tar.gz -C .
$ cd rp-pppoe-3.3
$ patch -p1 < ../../SRC/\
  rp-pppoed-3.3-uclibc-dreamcast.patch
$ ./configure
$ make CROSS=$UCLIBC/usr/bin/
$ su -c "make PREFIX=/home/christian/Dreamcast/\
  INITRD install"
$ su -c "cp scripts/adsl-connect /home/christian\
  Dreamcast/INITRD/usr/sbin"
$ cd ..
```

That's all. Now, we're ready to connect.

## 5.4 Connecting to an access concentrator

We've now compiled all needed programs for getting a connection to an access concentrator. For doing so, connect the DSL - modem into the so called *UPLINK* port of

your switch or hub, and connect your *SEGA Dreamcast*, if not already done, to your switch or hub, too.

Transfer the new initial ramdisk to your *SEGA Dreamcast*. Please refer to `cook.sh` in chapter four and chapter two.

If you've booted your *SEGA Dreamcast*, don't wonder of some error messages, if you've got the setup of chapter three: Some of the programs invoked by `/etc/init.d/rcS` aren't installed yet. Now, you can check your connection with the access concentrator. Please log in as *root* and type in:

```
root@hercules:~ # pppoe -A -I eth0
Access-Concentrator: BRAX11-erx
Got a cookie: b7 33 1e cf 5e e8 05 23 e0 89 14 90 bc 27 fe 89
------------------------------------------------
AC-Ethernet-Address: 00:90:1a:10:0f:f2
------------------------------------------------
root@hercules:~ #
```

Your output may vary. If you don't get similar messages, please check your cables.

Now, that we've got a cookie from the access concentrator, we've to set up some essential configuration files. The first file is `/etc/ppp/options` and the easiest, too:

```
root@hercules:~ # :> /etc/ppp/options
```

That's all. The documentation of `rp-pppoe` suggest an empty `ppp` configuration file. The next file is `/etc/ppp/pppoe.conf`:

```
root@hercules:~ # cat <<. > /etc/ppp/pppoe.conf
ETH=eth0
# ADSL user name.
USER=your_username
DEMAND=no
DNSTYPE=SERVER
USEPEERDNS=yes
DNS1=
DNS2=
DEFAULTROUTE=yes
CONNECT_TIMEOUT=30
CONNECT_POLL=2
ACNAME=
SERVICENAME=
PING="."
CF_BASE=‘/usr/bin/basename $CONFIG‘
PIDFILE="/var/run/$CF_BASE-adsl.pid"
```

```
SYNCHRONOUS=no
CLAMPMSS=1412
LCP_INTERVAL=20
LCP_FAILURE=3
PPPOE_TIMEOUT=80
PPPOE_EXTRA=""
PPPD_EXTRA=""
.
root@hercules:~ # chmod 640 /etc/ppp/pppoe.conf
```

For further configuration options or information beyond, please read either *README* or the corresponding documentation file of this source package.

Later, we'll set up a configuration with an automatic dynamic local nameserver, but for now, this sample configuration will do its job.

The last file is `/etc/ppp/pap-secrets`:

```
root@hercules:~ # cat <<. > /etc/ppp/pap-secrets
your_username * your_password *
.
root@hercules:~ # chmod 600 /etc/ppp/pap-secrets
```

Please ensure, the username in `pppoe.conf` corresponds to the one found in the configuration file shown above.

Now, it's time for rock'n'roll. On virtual terminal 1 (*ALT-F1*), you start the connection:

```
root@hercules:~ # /usr/sbin/adsl-connect
Connect: ppp0 <--> /dev/pts/0
Local IP address changed to aaa.bbb.ccc.ddd
Remote IP address changed to eee.fff.ggg.hhh
nameserver iii.jjj.kkk.lll
nameserver mmm.nnn.ooo.ppp
...
```

You should get some output here about an IP change of the device `ppp0` as well as the message of the nameserver that should have to be used.

On virtual terminal 2 (*ALT-F2*), you have to set at least one nameserver printed on virtual terminal 1:

```
root@hercules:~ # cat <<. > /etc/resolv.conf
nameserver iii.jjj.kkk.lll
.
```

Now, you should be able to test your connection:

```
root@hercules:~ # ping www.google.com
PING www.google.com (216.239.39.99): 56 data bytes
64 bytes from 216.239.39.99: icmp_seq=0 ttl=53 time=126.5 ms
64 bytes from 216.239.39.99: icmp_seq=1 ttl=53 time=126.2 ms
64 bytes from 216.239.39.99: icmp_seq=2 ttl=53 time=127.8 ms
64 bytes from 216.239.39.99: icmp_seq=3 ttl=53 time=126.8 ms
...
```

If you get a similar output, you're on the way. Congratulations! If you don't get such an output, please check all cables and read carefully the last steps of this article or read the included *README* files in the *ppp-2.4.1* and *rp-pppoe-3.3* source package.

## 5.5 Where are we?

Well, we see the finish line :-) We've got a connection to the access concentrator, so we can already surf the net on the *SEGA Dreamcast* gaming console. But for our aim to share an internet connection, we have to compile some other source packages to determine, which host may send which packets to the internet and vice versa, and for offering network address translation (*NAT*), known as *MASQUERADING*. So, the next chapter introduces `iptables`, the new routing and firewall generation for Linux kernels 2.4.x.

# 6 Setting up routing and firewalling

This chapter describes the compiling and setting up of `iptables`, the new routing tool for Linux kernels 2.4.x. This chapter doesn't want to introduce in the world of building *hard core* firewalling systems nor discuss a reliable und reasonable security level for gateways. For such reasons, please read Oskar Andreasson's great tutorial for `iptables` [Andreasson02]. This chapter merely wants to show the necessary steps for compiling and installing the needed tools for using the tutorial mentioned above. Beyond, we create one simple *firewalling rule* for using the *SEGA Dreamcast* as a *MASQUERADING* router for LANs.

## 6.1 Preparing the Linux kernel

The `iptables` tool controls firewalling rules implemented in the kernel. Therefore, we need a kernel with an appropriate set up. If you don't select the proposed kernel options in chapter two, please adjust your Linux kernel first:

```
$ cd ~/Dreamcast/KERNEL/linux
$ make ARCH=sh CROSS_COMPILE=sh4-linux- menuconfig
```

Table 6.1 points out the necessary options.

If you've selected these options, you have to rebuild your kernel:

```
$ make ARCH=sh CROSS_COMPILE=sh4-linux- \
  clean dep zImage modules
```

Finally, you've to install the kernel modules:

```
$ su -c "make ARCH=sh CROSS_COMPILE=sh4-linux- \
  INSTALL_MOD_PATH=/home/christian/Dreamcast/INITRD modules_install"
```

Don't be disturbed by the error message at the end: We don't have a `depmod` utility that handles modules for foreign hardware architectures, so simply ignore it.

Now, change directory to the initial ramdisk and correct the libraries folder:

```
$ cd ../..
$ cd INITRD
$ cd lib/modules/2.4.18-sh-dc
```

| Option | Type |
|---|---|
| CONFIG_PACKET | * |
| CONFIG_NETFILTER | * |
| CONFIG_UNIX | * |
| CONFIG_INET | * |
| CONFIG_IP_MULTICAST | * |
| CONFIG_IP_NF_CONNTRACK | M |
| CONFIG_IP_NF_FTP | M |
| CONFIG_IP_NF_IRC | M |
| CONFIG_IP_NF_IPTABLES | M |
| CONFIG_IP_NF_MATCH_LIMIT | M |
| CONFIG_IP_NF_MATCH_STATE | M |
| CONFIG_IP_NF_FILTER | M |
| CONFIG_IP_NF_NAT | M |
| CONFIG_IP_NF_TARGET_MASQUERADE | M |
| CONFIG_IP_NF_TARGET_LOG | M |

Table 6.1: Needed configuration for using *iptables*.

Remove every file and directory except the directory *kernel* - it contains our kernel modules. Now, we're ready for compiling the tool `iptables`.

## 6.2 Compiling *iptables-1.2.7a*

This source package contains the essential tool for controlling every packet leaving or arriving the *SEGA Dreamcast*.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| iptables-1.2.7a | 115 KB | GPLv2 | http://www.netfilter.org/files/iptables-1.2.7a.tar.bz2 |

Table 6.2: Source package, size and URL for *iptables*.

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  iptables-1.2.7a-uclibc-dreamcast.patch
```

The patch forces the build of a statically linked version due to the lack of a dynamic loader. Furthermore, it disables the *IPv6* support. Last, but not least, it corrects problems with the *strip* command.

Now, we're ready to compile `iptables-1.2.7a`:

```
$ cd ../BUILD
$ bunzip -c ../SRC/iptables-1.2.7a.tar.bz2 | \
  tar -xv -C .
$ cd iptables-1.2.7a
$ patch -p1 < ../../SRC/\
  iptables-1.2.7a-uclibc-dreamcast.patch
$ ./configure
$ make CROSS=$UCLIBC/usb/bin/ \
       KERNEL_DIR=/home/christian/Dreamcast/KERNEL/linux \
       DESTDIR=/home/christian/Dreamcast/INITRD
```

That's all. Now, copy the file `iptables` to your initial ramdisk and ensure it's executable:

```
$ su -c "cp iptables ../../INITRD/sbin"
$ su -c "chmod 744 ../../INITRD/sbin/iptables"
```

## 6.3 Setting up *IP forwarding* and *MASQUERADING*

Now, we've all needed packages and tools for acting as a router with full IP forwarding and network address translation.

For using these features, we have to set up the configuration of `/etc/init.d/rcS`, if you don't already create this file as suggested in chapter three.

Please edit `/etc/init.d/rcS` of your initial ramdisk and add the following lines:

```
echo "Setting up routing..."
/sbin/iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
```

The last line enables IP forwarding in the Linux kernel, whereas the `iptables` command allows the users of your LAN to use programs such as *IRC* without having installed a proxy or a socks server on your gateway. For further and more complex firewalling rules, please consult the following tutorial [Andreasson02].

For testing your installation, you can create and transfer a new image file for your *SEGA Dreamcast* using *cook.sh*. Please refer to chapter four and chapter two.

## 6.4 Where are we?

Now, we're ready to use our *SEGA Dreamcast* as a router with firewalling for a LAN. But we've got some problems: The given nameservers have to be set up by hand as well as the configuration of every client in the LAN. To make our life easier, we alter the actual configuration and provide an own nameserver together with a *DHCP* server,

that takes over the configuration of the connected clients. These steps are the theme of the next chapter.

# 7 Setting up an own *nameserver* and *DHCP - server*

This chapter helps you to set up your own *nameserver*. In combination with a *DHPC - server*, you as an administrator minimize your effort setting up every client connected to your LAN. Therefore, we download a *nameserver* whose origin bases on source code for an embedded nameserver. At the end of this chapter, we'll modify our start script and ADSL connecting script for using dynamic assigned nameservers.

## 7.1 Compiling *yaku-ns*

Even if the author of this source code warns on his homepage [Sanfilippo02] using this code because it's experimental, I've got no problems so far running the nameserver.

First, we need to download the appropriate source package. Because of its "experimental state", there's no official released source package. So, we have to download the complete source via CVS. CVS will ask for a password, just press enter, no password is required:

```
$ cd ~/Dreamcast/BUILD
$ cvs -d:pserver:anonymous@cvs.hping2.sourceforge.net:\
  /cvsroot/hping2 login
$ cvs -z3 -d:pserver:anonymous@cvs.hping2.sourceforge.net:\
  /cvsroot/hping2 checkout yaku-ns
$ cvs -d:pserver:anonymous@cvs.hping2.sourceforge.net:\
  /cvsroot/hping2 logout
$ cd yak-ns
```

Now, we may remove the CVS control directories:

```
$ for i in $(find . -type d -name "CVS" -print); do rm -fr $i; done
```

Now, we're ready to compile `yaku-ns`:

```
$ make CC=$UCLIBC/usr/bin/cc \
  AR=$UCLIBC/usr/bin/ar CFLAGS="-Os -I. -Wall"
```

Following, we copy the binary to our initial ramdisk:

```
$ mkdir ../../INITRD/usr/yaku-ns
$ cp yaku-ns ../../INITRD/usr/yaku-ns
```

Now, we configure the nameserver. First, we have to create a user and group named *yaku*:

```
$ cd ../../INITRD
$ cd etc
$ su
# cat <<. >>passwd
yaku:x:99:99::/usr/yaku-ns:/bin/sh
.
# cat <<. >>shadow
yaku:*:12091:0:99999:7:::
.
# cat <<. >>group
yaku:x:99:
.
# cat <<. >>gshadow
yaku:!::
.
```

Please ensure that you use unused *user-* and *group-ids*. Next, we create the configuration and set up some *chmod'es*:

```
# cd ../usr
# chown 99.99 /usr/yaku-ns
# chmod 700 /usr/yaku-ns
# chown 0.0 /usr/yaku-ns/*
# chmod 755 /usr/yaku-ns/yaku-ns
# touch /usr/yaku-ns/yaku-ns.log
# chown 99.99 /usr/yaku-ns/yaku-ns.log
# chmod 644 /usr/yaku-ns/yaku-ns.log
```

Now, we set up the configuration file:

```
# cd yaku-ns
# cat <<. > yaku-ns.conf
acl dns.allow 192.168.1.
acl dns.deny $
nameserver aaa.bbb.ccc.ddd
.
# chown 0.0 /usr/yaku-ns/yaku-ns.conf
# chmod 644 /usr/yaku-ns/yaku-ns.conf
# exit
```

As you see, only three lines and you've got your own nameserver. Please change the IP address *aaa.bbb.ccc.ddd* to one provided by the `adsl-connect` script in chapter five.

For further configuration options or deepening information, please read either *README* or the corresponding documentation file of this source package.

Now, you've got to edit the file `/etc/init.d/rcS` for starting up the nameserver automagically at system boot. If you've set up your `/etc/init.d/rcS` like in chapter three, you only have to remove the # at the beginnig of the following lines:

```
 .
 .
 .

#echo "Starting yaku-ns..."
#/usr/yaku-ns/yaku-ns -c \
# /usr/yaku-ns/yaku-ns.conf -l \
# /usr/yaku-ns/yaku-ns.log -u yaku -d

 .
 .
 .
```

Otherwise, simply add the following lines to `/etc/init.d/rcS`:

```
 .
 .
 .

echo "Starting yaku-ns..."
/usr/yaku-ns/yaku-ns -c \
 /usr/yaku-ns/yaku-ns.conf -l \
 /usr/yaku-ns/yaku-ns.log -u yaku -d

 .
 .
 .
```

Now, our nameserver will be started at system boot.

## 7.2 Compiling *udhcp-0.9.8*

This source package contains the client and server tools needed for using *DHCP*.

First, we need to download the appropriate source package:

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

| Package | Size | License | URL |
|---|---|---|---|
| udhcp-0.9.8 | 43 KB | GPLv2 | http://udhcp.busybox.net/downloads/udhcp-0.9.8.tar.gz |

Table 7.1: Source package, size and URL for *udhcp*.

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  udhcp-0.9.8-uclibc-dreamcast.patch
```

The patch corrects problems with the *strip* command and sets some environment variables.

Now, we're ready to compile `udhcp-0.9.8`:

```
$ cd ../BUILD
$ tar -xvzf ../SRC/udhcp-0.9.8.tar.gz -C .
$ cd udhcp-0.9.8
$ patch -p1 < ../../SRC/\
  udhcp-0.9.8-uclibc-dreamcast.patch
$ make CROSS=$UCLIBC/usr/bin/
```

That's all. Now, copy the file `udhcpd` to your initial ramdisk an ensure that it's executable:

```
$ su -c "cp udhcpd ../../INITRD/usr/sbin"
$ su -c "chmod 744 ../../INITRD/usr/sbin/udhpcd"
$ cd ../../INITRD
$ su
#
```

Now, we edit the file `/etc/init.d/rcS` for starting up the *DHCP* - server automagically at system boot. If you've set up your `/etc/init.d/rcS` like in chapter three, you only have to remove the # at the beginnig of the following lines:

```
  .
  .
  .

#echo "Starting udhcpd..."
#udhcpd

  .
  .
  .
```

Otherwise, simply add the following lines to `/etc/init.d/rcS`:

```
 .
 .
 .

echo "Starting udhcpd..."
udhcpd

 .
 .
 .
```

Finally, we have to create the configuration file for `udhpcd`:

```
# cd etc
# cat <<. >udhcpd.conf
start  192.168.1.100
end    192.168.1.254
interface   eth0
lease_file  /var/lib/udhcpd.leases
pidfile     /var/run/udhcpd.pid

option subnet 255.255.255.0
option router 192.168.1.1
option dns    192.168.1.1
option domain localdomain
option lease  864000
.
# chmod 644 udhcpd.conf
# touch ../var/lib/udhcpd.leases
```

For further configuration options or deepening information, please read either *README* or the corresponding documentation file of this source package.

Now, we've got a *DHCP* - server for our LAN.

## 7.3 Setting up automatic DNS

Up to now, we don't have to set up any clients for using our *SEGA Dreamcast* as gateway or as nameserver. But we have to set up the gaming console itself with the given nameserver in `/etc/resolv.conf` and in `/usr/yaku-ns/yaku-ns.conf`. In this section, we'll modify some scripts to let the *SEGA Dreamcast* do this job.

The first file is `/usr/yaku-ns/yaku-ns.conf.in`:

```
# cd ..
# cd usr/yaku-ns
# cat <<. >yaku-ns.conf.in
acl dns.allow 192.168.1.
acl dns.deny $
nameserver __NAMESERVER__
.
# chmod 644 yaku-ns.conf.in
```

As you see, we've change the real nameserver address with a wildcard. Now, we have to edit `/usr/sbin/adsl-connect` for filling up this wildcard. Please add the following lines directly after

```
echo "$!" > $PPPD_PIDFILE
```

in the *while loop*:

```
    .
    .
    .

    # Dynmaic nameserver implementation
    # Read given nameserver
    head -n1 /etc/ppp/resolv.conf | \
     sed s/nameserver\ // > /tmp/nameserver 2> /dev/null
    NAMESERVER=‘cat /tmp/nameserver‘

    # Ensure, $NAMESERVER is non-zero for
    # avoiding to kill the nameserver
    if [ ! -z $NAMESERVER ]; then
        # Alter yaku-ns configuration
        cat /usr/yaku-ns/yaku-ns.conf.in | \
         sed s/__NAMESERVER__/$NAMESERVER/ > \
         /usr/yaku-ns/yaku-ns.conf

        # Exist a yaku-ns process?
        pidof yaku-ns 2>&1 > /dev/null
        if [ "$?" == "0" ]; then
            # Inform yaku-ns of the
            # configuration change.
            kill -SIGHUP ‘pidof yaku-ns‘
        else
    # yaku-ns process doesn’t exist.
    # So, we have to restart the nameserver.
            /usr/yaku-ns/yaku-ns -c \
             /usr/yaku-ns/yaku-ns.conf \
             -l /usr/yaku-ns/yaku-ns.log \
```

```
            -u yaku -d
        fi
    fi

    rm -f /tmp/nameserver

    .
    .
    .
```

This script fragment reads the first given nameserver and replaces the wildcard in the configuration of *yaku-ns*. Finally, it sends a signal to the *yaku-ns* process forcing the local nameserver to re-read its configuration.

For testing your installation, you can create and transfer a new image file for your *SEGA Dreamcast* using *cook.sh*. Please refer to chapter four and chapter two.

## 7.4 Where are we?

Our router is almost complete. We've an own nameserver that's configured automagically by the *SEGA Dreamcast* itself. Additionally, we deliver a valid IP address configuration (*IP address, gateway address, nameserver address,...*) with *DHCP* to any client that wants to surf the Internet.

But one thing is still missing: A remote shell for configuring the *SEGA Dreamcast* without having to burn a new CD or transferring a new image file to the gaming console every time we change something.

Thus, the next chapter describes, how to compile and install a SSH implementation.

# 8 Setting up *SSH*

If you want to administer a host over a connection, you have to use a remote shell. In the past, one has taken `telnet` for this job. But today, it's unavoidable to use an encrypted connection due to the tasks done by the super user *root*. This chapter describes how to compile and install `ossh-1.5.12`, an implementation of the SSH protocol version 1.

Due to the very complicated legal situation about cryptographic software, **PLEASE READ CAREFULLY THE INCLUDED "COPYING" IN THE SOURCE PACKAGE OF `ossh-1.5.12`! I AM NOT RESPONSIBLE FOR ANY LEGAL CONSEQUENCES CONCERNING YOU (THE READER) WHEN USING THE SOFTWARE! PLEASE DECIDE YOURSELF, IF YOU ARE ALLOWED TO USE THIS SOFTWARE! ONCE AGAIN: READ CAREFULLY THE INCLUDED COPYING IN THE SOURCE PACKAGE! I PROVIDE ONLY THE TECHNICAL BACKGROUNDS NECESSARY FOR COMPILING THIS SOURCE!**

## 8.1 Compiling *zlib-1.1.4*

This source package provides some basic compressing and uncompressing algorithms needed by `ossh-1.5.12`.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `zlib-1.1.4` | 176 KB | free | ftp://ftp.info-zip.org/pub/infozip/zlib/zlib-1.1.4.tar.gz |

Table 8.1: Source package, size and URL for *zlib*.

Now, we're ready to compile `zlib-1.1.4` (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ cd ../BUILD
$ tar -xvzf ../SRC/zlib-1.1.4.tar.gz -C .
$ cd zlib-1.1.4
$ prefix=$UCLIBC CC=$UCLIBC/usr/bin/cc \
  AR="$UCLIBC/usr/bin/ar rc" \
  RANLIB="$UCLIBC/usr/bin/ranlib" CFLAGS="-Os"\
   ./configure
$ make all install
$ cd ..
```

That's all. You'll have a library for statically linking called `libz.a` besides a header file `zlib.h` being installed to your `$UCLIBC` toolchain.

## 8.2 Compiling *gmp-2.0.2.tar.gz*

This source package provides "Arithmetic without limitations" following their advertise :-) Well, *gmp* provides a library for doing mathematical operations, needed by *ossh* e.g. for the computing of primes.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `gmp-2.0.2` | 361 KB | LGPLv2 | http://gd.tuwien.ac.at/gnu/gnusrc/gmp/gmp-2.0.2.tar.gz |

Table 8.2: Source package, size and URL for *gmp*.

Following, we have to remove the sh machine definitions for forcing the use of no assembly code. They don't work with our Dreamcast.

```
$ tar -xvzf ../SRC/gmp-2.0.2.tar.gz -C .
$ cd gmp-2.0.2/mpn
$ mv sh sh-old
$ cd ..
```

Now, we're ready to compile `gmp-2.0.2`:

```
$ PATH=$UCLIBC:$PATH ./configure --nfp --host=sh
$ PATH=$UCLIBC:$PATH make
```

The temporary environment variable setting avoids applying a short patch. Now, copy the resulting library to your appropriate toolchain directory, e.g.

```
$ cp libgmp.a $UCLIBC/lib
```

And finally, copy the header - file to your appropriate toolchain directory, e.g.

```
$ cp gmp.h $UCLIBC/include
```

That's all. Now, we're ready to compile the encrypting and decrypting library *openssl*.

| Package | Size | License | URL |
|---|---|---|---|
| `openssl-0.9.6e` | 2,108 KB | OpenSSL / SLeay | http://www.openssl.org/source/openssl-0.9.6e.tar.gz |

Table 8.3: Source package, size and URL for *openssl*.

## 8.3 Compiling *openssl-0.9.6e*

This source package bundles several encrypting algorithms, used by *ossh*.

First, we need to download the appropriate source package:

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  openssl-0.9.6e-uclibc-dreamcast.patch
```

Now, we're ready to configure `openssl-0.9.6e`:

```
$ cd ../BUILD
$ tar -xvzf ../SRC/openssl-0.9.6e.tar.gz -C .
$ cd openssl-0.9.6e
$ patch -p1 < ../../SRC/\
  openssl-0.9.6e-uclibc-dreamcast.patch
$ ./Configure --prefix=$UCLIBC \
  --openssldir=$UCLIBC/openssl \
  no-shared no-asm linux-elf-sh
```

The build process will fail for applications and test programs, because we're on a cross building process. So, we just delete the appropriate folders:

```
$ rm -fr apps test
```

Following, it's time for compiling and installing `openssl-0.9.6e`:

```
$ make CC=$UCLIBC/usr/bin/cc \
  AR="$UCLIBC/usr/bin/ar r" \
  RANLIB=$UCLIBC/usr/bin/ranlib
$ make install
```

Now, you've got the library `libcrypto.a` as well as `libssl.a` besides the header files installed in `$UCLIBC/include/openssl`. At this point, all necessary libraries and header files are present for compiling *ossh*.

## 8.4 Compiling *ossh-1.5.12*

This source package contains the client and server tools needed for using *SSH*.

First, we need to download the appropriate source package:

| Package | Size | License | URL |
|---|---|---|---|
| `ossh-1.5.12` | 350 KB | free | http://gd.tuwien.ac.at/privacy/munitions /software/mirrors/ossh/ossh-1.5.12.tar.gz |

Table 8.4: Source package, size and URL for *ossh*.

Next, fetch the needed patch (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ wget http://www.tu-bs.de/~y0018536/dc/src/\
  ossh-1.5.12-uclibc-dreamcast.patch
```

The patch creates a correct *Makefile* for the *SEGA Dreamcast* besides a correct *Config.h* with some necessary `#define`'s. Furthermore, it disables the *get_canonical_hostname() - function* for connecting hosts that causes very long timeouts even in LANs. If you don't want this to be disabled, don't invoke *make* with the parameter `-DNORESOLVING`.

Now, we're ready to compile `ossh`:

```
$ cd ../BUILD
$ tar -xvzf ../SRC/ossh-1.5.12.tar.gz -C .
$ cd ossh-1.5.12
$ patch -p1 < ../../SRC/\
  ossh-1.5.12-uclibc-dreamcast.patch
$ make CROSS=$UCLIBC/usr/bin/ \
  CFLAGS="-Os -Wall -I. -I$UCLIBC/include \
  -I$UCLIBC/include/openssl -DNORESOLVING"
```

Now, copy at least the file `sshd` to your initial ramdisk and set it executable:

```
$ su -c "cp sshd ../../INITRD/usr/sbin"
$ su -c "chmod 744 ../../INITRD/usr/sbin/sshd"
```

Following, you have to create a so called *host key pair*. Either, you do this step on your *SEGA Dreamcast*, or you use your host to create this pair. I'll describe the latter.

Therefore, you need a SSH implementation running on your host. Just use *ossh* itself (you have to re-compile the source for running on your host), or you use another SSH implementation.

On my host, I found a SSH implementation version 2, your host may vary:

```
OpenSSH_3.4p1 Debian 1:3.4p1-1, \
SSH protocols 1.5/2.0, OpenSSL 0x0090603f
```

So, I don't compile *ossh* again. I just invoke the key creation with:

```
$ cd ../../INITRD
$ su
# cd etc
# mkdir ssh
# cd ssh
# ssh-keygen -f ./ssh_host_key -t rsa -P ""
# chmod 644 ssh_host_key.pub
# chmod 600 ssh_host_key
```

This host key pair identifies the *SEGA Dreamcast* when connecting with *SHH*. Next, we have to create a suitable configuration for SSH:

```
# cat <<. >sshd_config
Port 22
ListenAddress 0.0.0.0
HostKey /etc/ssh/ssh_host_key
RandomSeed /etc/ssh/ssh_random_seed
ServerKeyBits 768
LoginGraceTime 600
KeyRegenerationInterval 3600
PermitRootLogin yes

# Don't read ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
StrictModes yes
QuietMode no
X11Forwarding yes
FascistLogging no
PrintMotd yes
KeepAlive yes
SyslogFacility DAEMON
RhostsAuthentication no

# For this to work you will also need
# host keys in /etc/ssh/ssh_known_hosts
RhostsRSAAuthentication no

RSAAuthentication yes

# To disable tunneled clear text passwords,
# change to no here!
```

```
PasswordAuthentication yes
PermitEmptyPasswords no

# AllowHosts *.our.com friend.other.com
# DenyHosts lowsecurity.theirs.com *.evil.org evil.org
.
# chmod 644 sshd_config
# exit
$ cd ..
```

Last, but not least, you have to enable the *SHH* service. Therefore, you have to edit `/etc/init.d/rcS`. If you've set up your `/etc/init.d/rcS` like in chapter three, you only have to remove the # at the beginnig of the following lines:

```
 .
 .
 .

#echo "Starting sshd..."
#sshd -f /etc/ssh/sshd_config

 .
 .
 .
```

Otherwise, simply add the following lines to `/etc/init.d/rcS`:

```
 .
 .
 .

echo "Starting sshd..."
sshd -f /etc/ssh/sshd_config

 .
 .
 .
```

Now, you're able to connect to your *SEGA Dreamcast* with all of SSH clients supporting SSH version 1. If you want to use *public key authorization*, you have to do the following:

1. Create an identification key pair.

   You simply do the same you've done for the host key pair. Just invoke the key generation:

```
$ ssh-keygen -f ./identity -t rsa
$ chmod 600 identity
```

2. Copy `identity.pub` to your initial ramdisk.

```
$ su -c "mkdir INITRD/root/.ssh && \
  cp identity.pub INITRD/root/.ssh/authorized_keys"
```

3. Log in using `identity`.

   Assuming, you've booted your *SEGA Dreamcast*, you type in (if you're using a SSH implementation version 2):

```
$ ssh -1 -i identity -l root 192.168.1.1
```

Now, you're safe. Let every user existing on your *SEGA Dreamcast* create their key pair and copy every identity.pub in the *.ssh* folder in the appropriate home directory. For example, user *christian* has created his key pair. Then, he copies `identity.pub` to

`/home/christian/.ssh/authorized_keys`

From now on, he logs in with

```
$ ssh -1 -i identity -l christian 192.168.1.1
```

## 8.5 Where are we?

Now, we're able to administer our *SEGA Dreamcast* remote using an encrypted connection as well as allowing encrypted session for our users. At this point, your *SEGA Dreamcast* accepts only encrypted connections and acts as your local gateway with firewalling abilities.

Hint: If you have to wait too long for getting a shell while connecting to your *SEGA Dreamcast*, add the connecting host to `/etc/hosts` or simply compile *ossh* with the parameter `-DNORESOLVING`.

The next chapter describes how to connect several networks through an encrypted connection using *SSH / pppd*.

# 9 Setting up virtual private networking

This chapter describes the setting up for a *virtual private network* between two (geographically) disconnected LANs for sharing services and data over an encrypted logical connection. Figure 9.1 illustrates the interaction of the sent data.

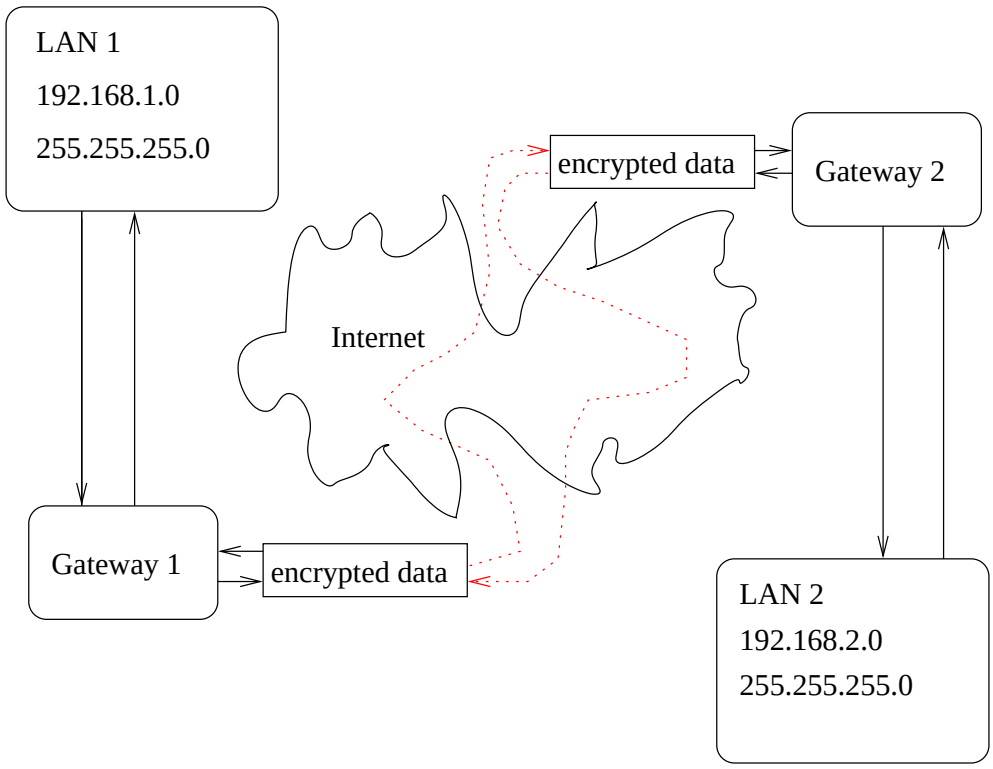


Figure 9.1: Diagram of a virtual private network between two networks over the Internet.

This encrypted logical connection is realized by a redirected point-to-point connection. This point-to-point connection will be redirected through an established SSH connection.

## 9.1 Setting up the *VPN - server*

A virtual private network is composed of at least one client and one server. The client wants to be included in the server's LAN for using services which are offered in that

LAN (e.g. an office LAN). So, this section describes the setting up of the server part. For further information please refer to [Wilson99].

First, we have to correct `/etc/ppp/options`:

```
$ cd ~/Dreamcast
$ su
# cd etc/ppp
# cat <<. >options
ipcp-accept-local
ipcp-accept-remote
proxyarp
noauth
.
# chmod 644 options
# exit
```

But now, we have to change `/usr/sbin/adsl-connect` because this script expects an empty *ppp* options file. So, we simlpy modify the *while - loop*. New lines are marked with `(*)` at the beginning. Please ignore `(*)` while typing in:

```
    .
    .
    .

    while [ true ] ; do
(*)     mv /etc/ppp/options /etc/ppp/options.org
(*)     touch /etc/ppp/options

        $PPPD pty "$PPPOE_CMD" \
                $PPP_STD_OPTIONS \
                $DEMAND \
                $PPPD_SYNC &
        echo "$!" > $PPPD_PIDFILE

(*)     sleep 1
(*)     rm -f /etc/ppp/options
(*)     mv /etc/ppp/options.org /etc/ppp/options

    .
    .
    .
```

Next, we have to create the users, who are allowed for VPN'ing, and a corresponding group:

```
$ cd ~/Dreamcast
$ su
# cd etc
# cat <<. >>passwd
Tux:*:1200:97::/home/vpn-users:/usr/sbin/pppd
.
# cat <<. >>shadow
Tux:*:12091:0:99999:7:::
.
# cat <<. >>group
vpn-users:x:97:
.
# cat <<. >>gshadow
vpn-users:!::
.
# exit
```

Please ensure you use unused *user*- and *group-ids*.

Next, we create the generic home directory as well as a generic *virtual private networking key pair*:

```
$ cd ~/Dreamcast
$ su
# cd home
# mkdir -p vpn-users/.ssh
# chmod -R 755 vpn-users
# chown -R 0.97 vpn-users
# exit
$ cd ../..
$ ssh-keygen -f ./identity -t rsa -P ""
$ su -c "cp identity.pub INITRD\
  /home/vpn-users/.ssh/authorized_keys"
```

At this point, the *VPN - server* is set up.

## 9.2 Setting up the *VPN - client*

If you don't have the tool `pty-redir` on your host, you have to download the appropriate source package first:

Now, we're ready to compile `pty-redir` (assuming your `pwd` points to *~/Dreamcast/SRC*):

```
$ cd ../BUILD
$ tar -xvzf ../SRC/pty-redir-0.1.tar.gz -C .
```

| Package | Size | License | URL |
|---|---|---|---|
| `pty-redir-0.1` | 20 KB | GPLv2 | http://uncensored.citadel.org/pub/unix/pty-redir-0.1.tar.gz |

Table 9.1: Source package, size and URL for *pty-redir*.

```
$ cd pty-redir-0.1
$ make clean
$ make
```

Now, copy the binary to `/usr/sbin`:

```
$ su -c "cp pty-redir /usr/sbin"
$ cd ../..
```

Now, you're able to build a point-to-point encrypted tunnel with *pppd* over SSH with your `identity` - file:

```
$ /usr/sbin/pty-redir /usr/bin/ssh -t -e none -1 \
  -o 'Batchmode yes' -c blowfish -i ./identity \
  -l Tux 192.168.1.1 > /tmp/vpn-tunnel
$ sleep 5
$ /usr/sbin/pppd `cat /tmp/vpn-tunnel` \
  192.168.2.10:192.168.3.10
```

The IP - addresses used above are only an example for testing the tunnel. Normally, you've to check for a unused IP address of LAN 1 and one in LAN 2 between you can establish the encrypted SSH tunnel.

For testing purposes, you may ping the *SEGA Dreamcast* with the given IP address:

```
$ ping 192.168.3.10
PING 192.168.3.10 (192.168.3.10): 56 data bytes
64 bytes from 192.168.3.10: icmp_seq=0 ttl=255 time=6.4 ms
64 bytes from 192.168.3.10: icmp_seq=1 ttl=255 time=5.2 ms
64 bytes from 192.168.3.10: icmp_seq=2 ttl=255 time=4.5 ms
...
```

All sent packets go through the `ppp?` device of your host to the SSH tunnel, where they'll be encrypted. Then, these encrypted packets are sent over the SSH connection to the other point, where the process is invoked vice versa: A received packet will be decrypted and sent to `ppp?`.

## 9.3 Where are we?

Well, our router with firewalling and virtual private networking is quite complete. We've build a complete Linux distribution from scratch, which allows us to use a famous gaming console as a software router. We've compiled everything we need from scratch: Beginnig at the first toolchain and ending up to the source code implementing the SSH protocol.

But, we still miss one thing: A persistent configuration. Every time we're rebooting, we'll loose out configuration until we burn a *final* configuration to a CD-R. But, you'll agree, when is a configuration *final*?

So, the last chapter describes, how to use the *VMU* for storing configuration files.

# 10 Setting up a *persistent configuration*

This chapter rounds out your Linux distribution. You'll use your *VMU* for storing configuration files. Therefore, you've got to change some essential configuration files we've set up so far.

## 10.1 Changes on your *SEGA Dreamcast*

For using your *VMU* to store your configuration files, you've to determine if the *VMU* is plugged in. There are several options to do this job, I'll show the easiest one.

At system boot, we try to mount the *VMU*. Either the *mounting* fails or succeeds. The latter, we look for an executable configuration file on the *VMU* for delivering system control. This configuration file is comparable to `/etc/init.d/rcS`. The tasks of this file are to copy or link the necessary configuration files to the *root file system*.

For using a persistent configuration, simply edit `/etc/init.d/rcS` of your initial ramdisk (your file may vary, please refer to chapter three) and change the following lines (new lines are marked with an asterisk):

```
      .
      .
      .

      echo "Setting up routing..."
      iptables -t nat -A POSTROUTING \
       -o ppp0 -j MASQUERADE
      echo 1 > /proc/sys/net/ipv4/ip_forward

(*)   echo "Trying to load a persistent configuration..."
(*)   mount -t minix -o ro /dev/mtdblock0 /mnt 2>&1 > /dev/null
(*)   if [ -x /mnt/etc/init.d/rcS ]; then
(*)       echo "Persistent configuration found. Invoking..."
(*)       /mnt/etc/init.d/rcS
(*)   else
(*)       echo "No persistent configuration found. \
(*)        Reverting default configuration..."

          echo "Setting hostname..."
```

```
            hostname dreamcast

            echo "Configuring lo..."
            ifconfig lo 127.0.0.1 up

            echo "Configuring eth0..."
            ifconfig eth0 192.168.1.1 up

            echo "Starting klogd..."
            klogd

            echo "Starting syslogd..."
            syslogd

            echo "Starting udhcpd..."
            udhcpd

            echo "Starting yaku-ns..."
            /usr/yaku-ns/yaku-ns -c /usr/yaku-ns/yaku-ns.conf \
              -l /usr/yaku-ns/yaku-ns.log -u yaku -d

            echo "Starting sshd..."
            sshd -f /etc/ssh/sshd_config

            sleep 1

            echo "Everything's done. Have fun."
(*)   fi
```

As you see, your *VMU* is mounted *read only*. This option is given to `mount` for avoiding any file system errors and inconsistencies while loosing power.

Now, your configuration file on the *SEGA Dreamcast* is set up.

## 10.2 Persistent configuration on the *VMU*

The most important file is `/etc/init.d/rc.S` found on the *VMU*. So, we first have to prepare the *VMU* for storing the *persistent configuration*. Boot your *SEGA Dreamcast* and plug in the *VMU*. You'll notice some Linux kernel messages about the found flash memory system. Log in as root and type in:

```
# mkfs.minix /dev/mtdblock0
```

Now, your *VMU* is formatted. Following, you can mount your *VMU* for storing files:

```
# mount -t minix /dev/mtdblock0 /mnt
```

Your flash memory is now accessible under `/mnt`. The first file being created is `/mnt/etc/init.d/rcS`:

```
# mkdir -p /mnt/etc/init.d
# cat <<. >/mnt/etc/init.d/rcS
#!/bin/sh

PATH=/bin:/sbin:/usr/bin:/usr/sbin

echo "Setting up filesystem..."

# First, set up the users.
rm -f /etc/passwd
rm -f /etc/shadow
rm -f /etc/group
rm -f /etc/gshadow
ln -s /mnt/etc/passwd /etc/passwd
ln -s /mnt/etc/shadow /etc/shadow
ln -s /mnt/etc/group /etc/group
ln -s /mnt/etc/gshadow /etc/gshadow

# Then, set up hosts.
rm -f /etc/hosts
ln -s /mnt/etc/hosts /etc/hosts

# Now, set up ADSL connection.
cp /mnt/etc/ppp/pap-secrets /etc/ppp
cp /mnt/etc/ppp/pppoe.conf /etc/ppp

# Following, prepare SSH host_key.
cp /mnt/etc/ssh/ssh_host_key /etc/ssh
cp /mnt/etc/ssh/ssh_host_key.pub /etc/ssh

# Then, we set up the /home directories.
ln -s /mnt/home/christian /home/christian

echo "Setting hostname..."
hostname hercules

# If you have a german keymap
#echo "Setting keymap..."
#loadkmap < /mnt/usr/share/keymaps/qwertz.map

echo "Configuring lo..."
```

```
ifconfig lo 127.0.0.1 up

echo "Configuring eth0..."
ifconfig eth0 192.168.1.1 up

echo "Starting klogd..."
klogd

echo "Starting syslogd..."
syslogd

echo "Starting udhcpd..."
udhcpd

echo "Starting yaku-ns..."
/usr/yaku-ns/yaku-ns -c /usr/yaku-ns/yaku-ns.conf \
 -l /usr/yaku-ns/yaku-ns.log -u yaku -d

echo "Starting sshd..."
sshd -f /etc/ssh/sshd_config

sleep 1

echo "Everything's done. Have fun."
.
# chmod 744 /mnt/etc/init.d/rcS
```

The first few lines remove the generic `passwd` - and `group` - files and replace them with my personal configuration.

The next two lines set up a specific `/etc/hosts` - file.

Following, my personal ADSL account information will be copied to the *root filesystem*.

Furthermore, the SSH specific *host_key pair* files (please see chapter eight) are transferred to the *root file system*.

Last but not least, my personal home directory with a SSH public key is linked to the *root file system*.

Obviously, you have to create or copy every linked or copied file from this script to your *VMU*. I suggest following procedure:

1. Create a special directory on your host.
2. Create or copy every needed file desired to be persistent to this directory.
3. Change both the `/etc/init.d/rcS` on your *SEGA Dreamcast* and the corresponding `/etc/init.d/rcS` for your *VMU* copying or linking every needed file.

4. Change carefully the mode and owner of each file.
5. Archive the complete directory into one single file:

   ```
   $ cd persistent_configuraton
   $ tar -cvpf ../vmufs.tar .
   ```

   The option `-p` is necessary for preserving the modes of the single files.
6. Transfer the file `vmufs.tar` to your *SEGA Dreamcast* using `wget` for example (assuming you're running a local web - or ftp - server):

   ```
   # cd /tmp
   # wget http://192.168.1.11/vmufs.tar
   ```

7. Unpack the archive on the mounted *VMU*:

   ```
   # mount -t minix /dev/mtdblock0 /mnt
   # cd /mnt
   # tar -xvf /tmp/vmufs.tar .
   ```

8. Unmount the *VMU* and reboot you *SEGA Dreamcast* for testing your *persistent configuration.*

This procedure has two advantages: First, you don't need to edit and change your *persistent configuration* directly on your *SEGA Dreamcast*, and second, you've got a backup of your configuration on your host.

## 10.3 Where are we?

Now, everything's complete. You've converted your *SEGA Dreamcast* into a router with firewalling and virtual private networking. Congratulations.

Now, you might set up several firewalling rules using [Andreasson02] or set up some more programs. For example, I've compiled *im-httpd_0.04* for having a simple web server or *ez-ipupdate* for getting a "static" FQDN for my changing IP addresses for using SSH or showing some web pages.

You'll find my *initial ramdisk* under

`http://www.tu-bs.de/~y0018536/dc`

as well as the newest version of this article.

# 11 Acknowledgements

First, I want to thank the whole Open Source Community for their great work.

Special thanks go to Bill Gatliff who inspired my work with his script.

Following, I want to thank my parents for their tolerance for my nightly work :-) and my brother who supports me writing this article.

Furthermore, I want to thank O. Dzubiel for sponsoring the *SEGA LAN Adapter/HIT-0300* for my *SEGA Dreamcast*. Without the ethernet networking adapter, nothing would be possible.

Next, thanks go to C. Groessler for his tips while I was writing the Linux device driver for the *SEGA LAN Adapter/HIT-0300*.

Following, I want to thank J. - C. Treusch for his support in situations I don't see any way out and for his cover picture.

Furthermore, thanks go to `#linuxdc` and all the people there for their hints.

Following, thanks go to Rick Lehrbaum who published my work on his portal *www.linuxdevices.com*.

Last but not least thanks to all people I've forgotten in my short list.

*11 Acknowledgements*

# 12 GNU Free Documentation License

GNU Free Documentation License

Version 1.2, November 2002



0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The ”Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ”you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A ”Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ”Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing

that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ”Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called ”Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ”Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section ”Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ”Acknowledgements”, ”Dedications”, ”Endorsements”, or ”History”.) To ”Preserve the Title” of such a section when you modify the Document means that it remains a section ”Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled ”Acknowledgements” or ”Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled ”Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled ”Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled ”Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section

unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled ”Acknowledgements”, ”Dedications”, or ”History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License ”or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

# 13 Glossary

This short and never complete list is about giving you a short reference of the used terms in this article.

**bootstrap compiler:** A minimal compiler for building runtime libraries and operating system kernels.

**configure:** A script included in a source package for building a *Makefile* from a sequence of templates.

**cross compiler:** A cross compiler is used for building binaries, in common executable files for a platform different from the one you're working on.

**DHCP:** Dynamic Host Configuration Protocol. This protocol allows a central administration of IP addresses among others.

**gdb:** The *GNU debugger*.

**GNU/Hurd:** The kernel of the GNU project, composed of a microkernel and a lot of processes for encapsulating the services e.g. for *filesystems*.

**inittab:** This file controls the processes to be started at system booting time.

**IPv6:** Next generation Internet.

**IRC:** Internet Relay Chat.

**Makefile:** A "recipe" for building a program from source.

**NAT:** Network address translation. Hide several hosts behind one real IP address and all related problems and features.

**linker:** A program for gluing libraries and object files together.

**Linux distribution:** The *Linux* kernel including some essential userland programs.

**process id:** The unique number of a running or sleeping process.

**root file system:** The file system mounted at /.

**serial loader:** A serial loader is a set of two programs: A server and a client. The server waits on your *SEGA Dreamcast* for a connecting client. The client transfers an image file to the server, which the server after completion executes.

**strip:** A tool for removing sections of a compiled program needed for debugging.

**TCO:** Total Cost Of Ownership. These are the costs you have to pay for running something, e.g. power, connection costs, administration etc.

**Toolchain:** A toolchain is the amount of programs for compiling a program.

**uClibc:** A very minimal C - library, especially for embedded devices.

**VMU:** The Visual Memory Unit is a small flash memory card.

# Bibliography


[Beekmans98] *http://www.linuxfromscratch.org*, Gerard Beekmans, 1998-2002

[Gatliff01] *Running Linux on the Sega Dreamcast*, Bill Gatliff, http://linuxdevices.com/articles/AT7466555948.html, Sep 24, 2001

[FHS01] *http://www.pathname.com/fhs*, Filesystem Hierarchy Standard, May 24, 2001

[Comstedt00] *http://mc.pp.se/dc/cdr.html*, Bootable CD-Rs, Nov 08, 2000

[RFC2516] *A Method for Transmitting PPP Over Ethernet (PPPoE)*, Network Working Group, http://www.faqs.org/rfcs/rfc2516.html, 02/1999

[Andreasson02] `iptables` *tutorial*, Oskar Andreasson, http://iptables-tutorial.frozentux.net, 2002

[Sanfilippo02] *http://www.kyuzz.org/antirez/ens.html*, Salvatore Sanfilippo, Sep 06, 2002

[Wilson99] *VPN HOWTO*, Matthew D. Wilson, http://www.ibiblio.org/pub/Linux/docs/howto/other-formats/ps/VPN-HOWTO.ps.gz, 12/1999

# Index