CHAPTER 1

INTRODUCTION TO THE INTERNET LAB

Lab 1 is an introduction to the hardware and software environment of Internet lab. The Internet lab can be thought of as a miniature version of the Internet. Instead of millions of hosts and routers, the Internet lab has only four routers, four personal computers (PCs), and a few Ethernet hubs. Still, the available equipment is sufficient to emulate many traffic scenarios found on the real Internet.

This chapter has four sections. Each section covers material that you need to run the lab exercises in this book. The first section gives an overview of the Internet lab equipment. Then, we give an introduction to the Linux operating system. The last section discusses the application and traffic analysis tools used in Lab 1.

TABLE OF CONTENTS

1. Overview of the Internet Lab

The Internet Lab consists of a set of routers, PCs, and Ethernet hubs which. The setup of the Internet lab equipment should be similar to that in Figure 1.1. Figure 1.2 shows photographs of actual Internet labs.

With four PCs and four routers, the Internet Lab is a small network. However, with the small equipment pool it is feasible to complete non-trivial lab exercises in a reasonable amount of time, and it is practical to give you exclusive access to the equipment.

The Internet lab is completely isolated from the Internet. Therefore, you can perform tasks that would cause significant disruptions in an operational network (such as unplugging network cables) or would raise security concerns (such as capturing and studying network traffic). In an operational network, the majority of tasks that you will perform in the lab would be restricted to experienced network engineers and system administrators.

1.1. Description of the Hardware

The Internet lab has four PCs, which are labeled as PC1, PC2, PC3, and PC4. All PCs have the Linux Red Hat 7.1 operating system installed. All necessary measurement and configuration software needed in the labs are already configured. Each PC has at least a floppy drive, a serial port, and two 10/100 Mbps Ethernet interface cards. The back of each PC should be similar to the sketch in Figure 1.3. Each PC has connectors for power supply, and ports to connect a keyboard, a mouse, a monitor, parallel devices, serial devices, audio devices, and Ethernet network devices. In Figure 1.4, the network interface cards (or *NICs*) are labeled as *eth0* and *eth1*, and the serial ports are labeled as *ttyS0* and *ttyS1*. These label refer to the names Linux uses to identify the Ethernet cards or serial ports. For example, when you assign an IP address to an Ethernet interface card in Linux, you need to specify the name of the interface.

> **Equipment of the Internet Lab:**
>
> 1. 19" equipment rack
> 2. 4 Cisco Routers
> 3. 4 Ethernet Hubs
> 4. 4 hosts
> 5. 1 Monitor, 1 keyboard, 1 mouse, 1 KVM switch
> 6. Cables and connectors:

All PCs are controlled from a single KVM (Keyboard-Video-Mouse) switch, which connects a keyboard, monitor, and mouse to the PCs. The KVM switch gives you control over all four PCs from single keyboard, monitor and mouse set. However, with a KVM switch you can access only one PC at a time.
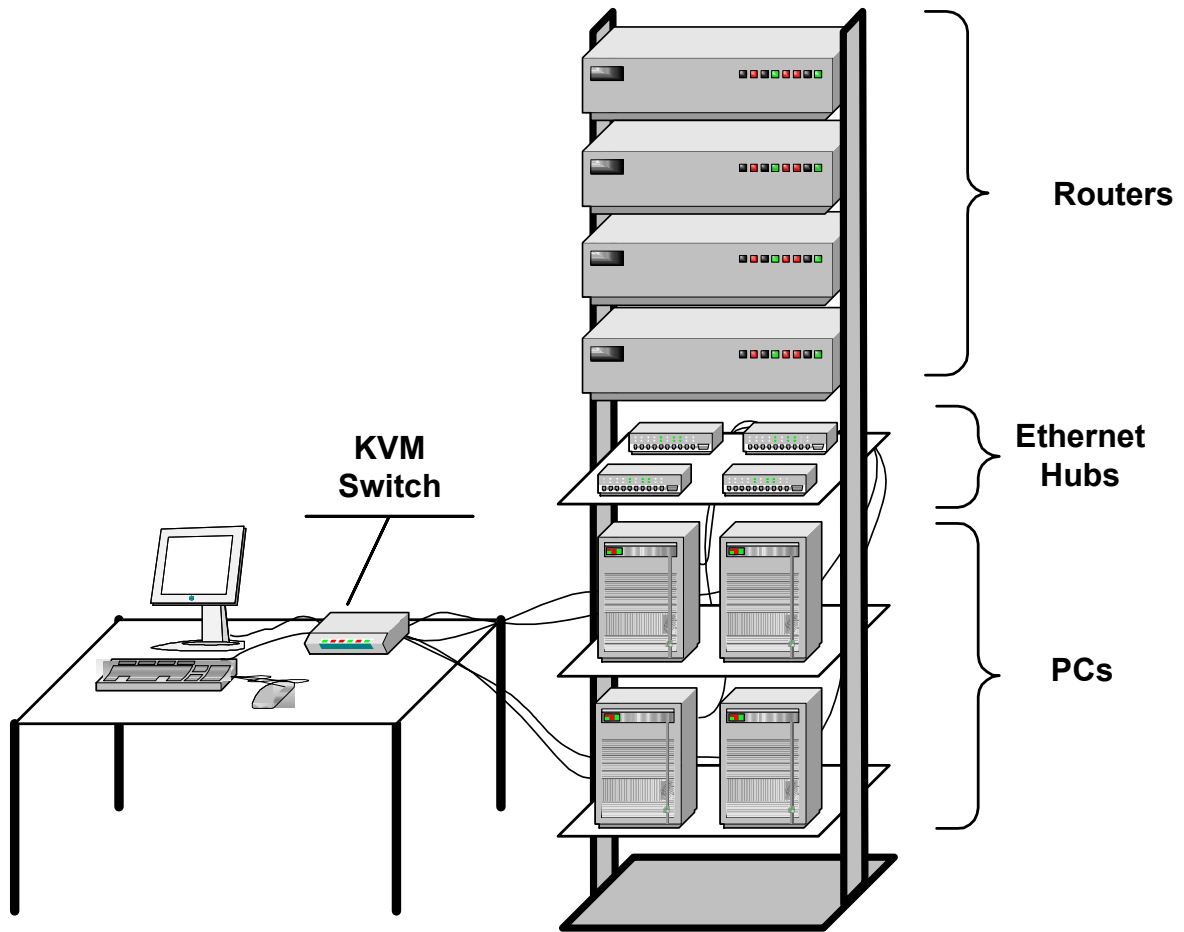
3

Figure 1.1. Illustration of the Internet lab equipment.

Internet Lab at UC Irvine
with Cisco 2514 routers

Internet Lab at UVA.
with Cisco 2514

Internet Lab at UVA.
with Cisco 7010 routers

Figure 1.2. Examples of Internet labs.

Power switch

Power plug

PS/2 ports (Mini DIN 6)
for mouse and keyboard

Parallel ports (DB25)

Serial ports
(DB9)

ttyS0

ttyS1

PC1

VGA/SVGA Port (HD15)
for monitor

Two Ethernet interface cards
with RJ-45 connectors

eth0

eth1

Audio Jacks for Line-in,
microphone, and output

USB ports
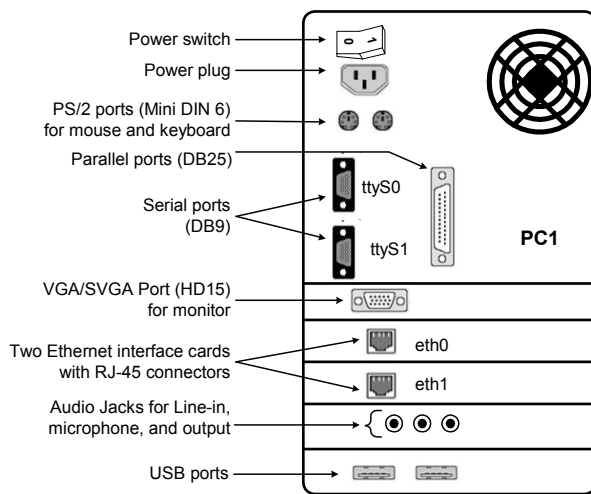
Figure 1.3. Ports on a PC. The serial ports and Ethernet interface cards are labeled.

Ehternet AUI ports
(DB-15)

Synchronous Serial
ports (DB-60)

Console
port

Auxiliary
port

Power
switch

Power
plug
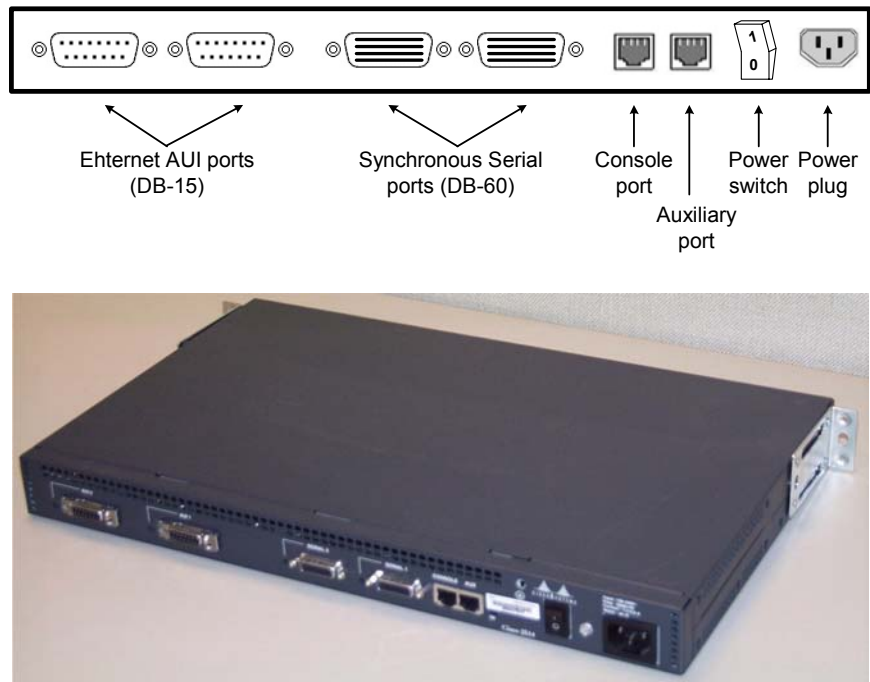
Figure 1.4. Back of a Cisco 2514 router with two Ethernet ports and two serial ports.

Four RJ-45 connectors

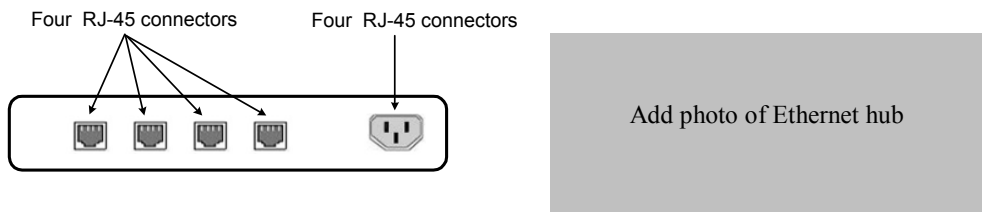Four RJ-45 connectors

Add photo of Ethernet hub

Figure 1.5. Ethernet Hub with four ports.

The Internet lab requires four Cisco routers, labeled as Router1, Router2, Router3, and Router4. Each router has two Ethernet interfaces and at least one serial port. The Ethernet interfaces operate at 10 or 100 Mbps and the serial interfaces have a rate of up to 2 Mbps. The requirement for routers from a specific vendor is due to the fact that vendors of commercial routing equipment generally provide a proprietary command language configuring a router. The labs in this book use the command language IOS (Internet Operating System), which is the command and configuration language for routers from Cisco Systems. The lab exercises use IOS version 12.0. There are

different types of Cisco routers. Some have pre-configured interfaces, such as the Cisco 25xx or Cisco 26xx series, and some have configurable slots, such as the Cisco 36xx or Cisco 7xxx series. Figure 1.4 shows the back of a Cisco 2514 router, which has two Ethernet ports and two serial ports. The routers in the Internet lab are labeled Router1, Router2, Router3, and Router4.

There are four Ethernet hubs in the lab, each with at least four ports (see Figure 1.5). The hubs should be dual-speed, that is, support data transmissions at both 10 and 100 Mbps. Hosts or routers that connect to the same Ethernet hub build an Ethernet local area network (LAN). For example, the network configuration for the lab exercises in Lab 1 consists of a single Ethernet segment, where four PCs are connected to a single Ethernet hub.

Lastly, the Internet lab has different kinds of cables and connectors. There are cables to connect the keyboard, video, and mouse ports of the PCs to the KVM switch. Then there are cables which connect the Ethernet interface cards of PCs and routers to the Ethernet hubs. We will call these cables *Ethernet cables*. Lastly, there are two types of serial cables. One type of serial cables is used to connect a PC to the console port of a router. The other type of serial cables connects two serial interfaces of the routers. We will discuss serial cables in more detail in Lab 3.

1.2.   Wiring a Twisted Pair Ethernet network

Most network experiments in the Internet lab are done over Ethernet, which is the dominant networking technology for local area networks (LANs). Ethernet LANs are discussed in detail in Chapter/Lab 2. The following discussion presents an overview of setting up the wiring of an Ethernet network.

The Ethernet equipment in the Internet Lab is based on the physical layers 10BaseT and 100BaseTX, which are currently the most widely used physical layers for Ethernet. 10BaseT and 100BaseTX have a data rate of 10 Mbps and 100 Mbps, respectively. Network interface cards with 10BaseT or 10Base100TX are connected with unshielded twisted pair (UTP) cables with RJ-45 connectors, to an Ethernet hub or an Ethernet switch.[1] Therefore, the resulting network has a start configuration with the hub (or switch) in the center.

Setting up a 10BaseT and 100BaseT Ethernet network configuration is relatively easy. Given a set of PCs (as shown in Figure 1.3) and Ethernet hubs (as shown in Figure 1.5), you simply need to connect PCs and hubs with UTP cables that have RJ-45 connectors, as shown in Figure 1.6. We will often to refer to these cables as Ethernet cables.

_____

[1] The difference between hubs and switches is the subject of Lab 5. For the time being, we assume that we only have hubs.
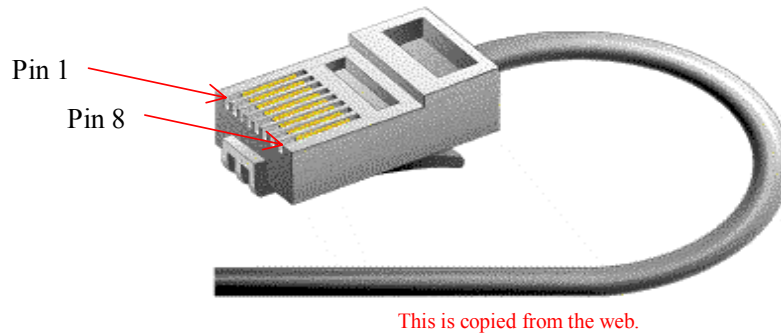
Pin 1

Pin 8

Figure 1.6. RJ-45 connector.

However, there are a few things to consider when wiring an Ethernet network. The first is the quality of cables used. UTP cables come in different quality grades, from low telephone-grade cables to cables suitable for high-speed transport. Using UTP cables with an insufficient quality grade results in a large number of transmission errors, which degrade performance or make communication impossible. For specifying the quality grade of UTP wires, one generally uses the rating scheme from the ANSI/EIA/TIA-568[2] standard for wiring commercial buildings. For 100BaseTX Ethernet, the twisted pair cable must satisfy the Category 5 (Cat 5) rating. A Cat 5 cable has four pairs of twisted wires. It is noteworthy that 10BaseT and 100BaseTX Ethernet only uses two of the four wire pairs of a Cat 5 cable. Referring to Figure 1.6, an RJ-45 only uses pins 1,2, 3 and 6.

A second issue to consider when wiring an Ethernet network, is that Ethernet cables come in two types: *straight-through cables* and *crossover cables.* The difference between these two types is the assignment of pins of the RJ45 jack to the wires of the Cat 5 UTP cable. In Figure 8, we show an RJ-45 plug with 8 pins. The assignment of pins to the wires is shown in Figure 1.7. In a straight through cable, the pins of the RJ-45 connectors at both ends of the cables are connected to the same wire. A crossover cable switches pins 1 and 6, and pins 2 and 3, which corresponds to switching the transmit and receiver pins. Since it is easy to confuse straight-through cables and crossover cables, it is good practice to clearly label the different types of cables.

To determine if a cable is straight-through or crossover, simply hold the RJ-45 connectors of both cable ends next to each other (with the connector end pointing away from you) and compare how

---

[2] ANSI = American National Standards Institute,
EIA = Electronics Industries Association
TIA = Telecommunication Industry Association

the colored wires are connected to the pins of the RJ-45 connectors. Since the plastic shielding of each wire has a different color, you can quickly determine the difference between a straight-through cable and a crossover cable. In a straight-through cable, colors appear in the same sequence. In a crossover cable the positions of pins 1 and 3, and 2 and 6 are reversed.
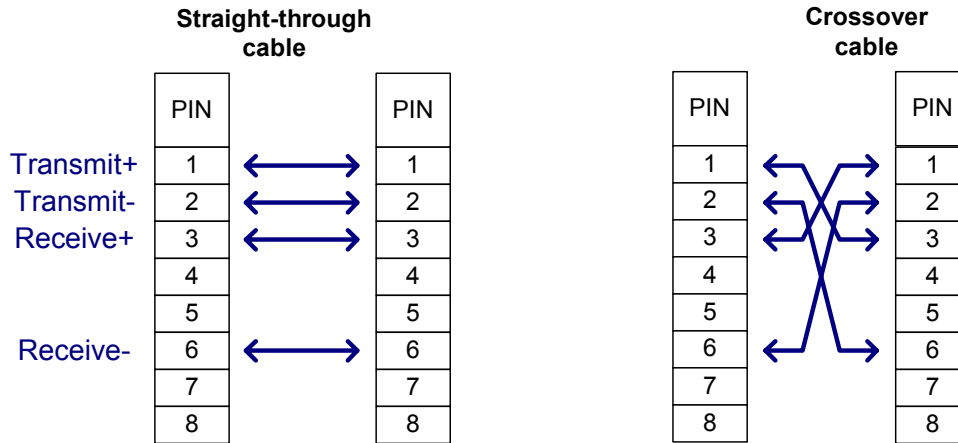

Figure 1.7. Pin connection for straight through and crossover Ethernet.

To connect a PC or router to an Ethernet hub, you need to use a straight-through cable. When connecting two PCs with an Ethernet cable or a PC to a router, or two routers directly without a hub, you must use a crossover cable. Likewise, when connecting two hubs you must use a crossover cable. There is one special case: some hubs have a port which is labeled as "uplink port". If you connect on uplink port of one hub to a regular port of another hub you need to use a straight-through cables. Figure 1.8 illustrates these scenarios. In Figure 1.8(a) devices are connected to a hub using a straight-through cables, in Figure 1.8(b) devices are connected directly with crossover cables, and in Figure 1.8(c) a configuration is shown where hubs are connected to each other, with and without an uplink port.
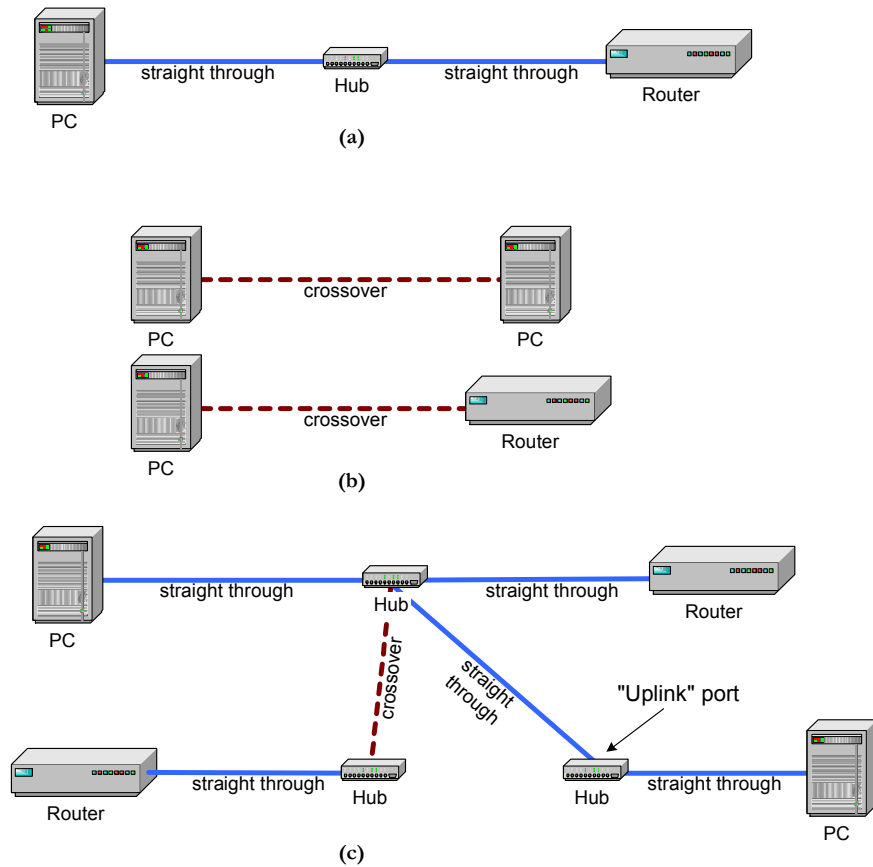
Figure 1.8. Wiring an Ethernet network with straight-through and crossover cables.

There is, of course, the question why Ethernet needs different types of cables. The answer has to do with different wirings of RJ-45 ports at hubs and other devices. For illustration, consider two devices that each have a communication port with two pins. The ports at both devices are identical: the first pin is used to transmit data and the second pin is used to receive data. (For the sake of the illustration, we ignore the need to close electrical circuits.) This situation is illustrated in Figure 1.9(a). When connecting the two devices we need to ensure that the first pin of one device is connected to the second pin of the other device., so that the transmission pin at one end connects to the receiving pin at the other end. This crossing of wires is accomplished by a crossover cable. A straight through cable only works, if the pins of the two devices have a reversed order, as shown in Figure 1.9(b). Ethernet networks exhibit both scenarios shown in Figure 1.9. In 10BaseT and 100BaseTX, the ports of a hub have a different configuration than the ports of Ethernet end systems, such as the PCs and routers. There are technical terms for these configurations: Ethernet ports at hubs are called DCE (Data Communications Equipment) devices and Ethernet ports of PCs

and routers are called as DTE (Data Terminal Equipment) devices. Now we can also explain the uplink port found on some Ethernet hubs. It is simply a port on a hub with a DTE configuration.

(a) Crossover cable needed for identical ports

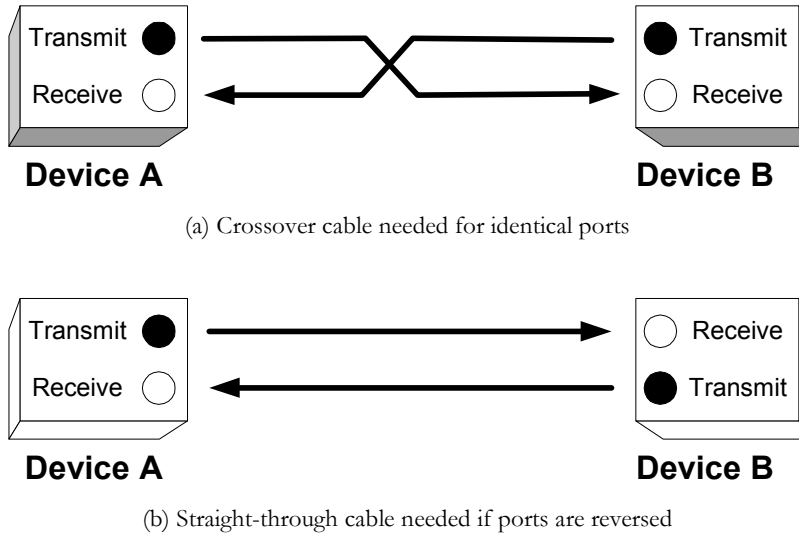(b) Straight-through cable needed if ports are reversed

Figure 1.9. Crossover and straight-through cables.

The last issue to consider is that some Ethernet interfaces may not have RJ-45 connectors. Some Ethernet interfaces, e.g., those on Cisco 2500 or Cisco 7000 series routers, have 15-pin AUI ports on their Ethernet interfaces. In that case, you need an AUI/RJ-45 transceiver, shown in Figure 1.10, which is a small device that has an AUI interface on one side and an RJ-45 port on the other side.

A transceiver (sometimes also called Media Access Unit or MAU) implements the physical layer of an Ethernet interface. All Ethernet interfaces have transceivers, but they are usually integrated on the network interface card. Transceivers with AUI interfaces are a relict from the early days of Ethernet. External transceivers are helpful, when different physical layers are used in the same Ethernet network. For example, a few years ago, when coaxial Ethernet media were replaced by UTP, systems with AUI Ethernet interfaces could transition to twisted pair cabling simply by replacing the transceiver.

Figure 1.10. Two AUI/RJ-45 transceivers – the top one shows the RJ-45 end, the bottom one shows the AUI end.

2. An Overview of Linux

All network experiments in the Internet lab are controlled from the PCs in the Internet Lab. Since the PCs run the Red Hat Linux operating system, you need to know how to use and navigate the Linux operating system. This section gives a brief overview of Linux for newcomers to Linux. If you have worked on Linux before, this section may not provide new information, and you may want to quickly browse this section, or skip it entirely.

We emphasize that our overview of Linux is very selective and touches only on a limited set of topics relevant to the lab experiments.

Linux and Unix

Linux is a clone of the Unix operating system. Unix was developed in 1969 by Dennis Ritchie and Kevin Thompson at Bell Laboratories. Different from most operating systems at the time, Unix was designed to be easily portable to different hardware platforms. One factor that contributes to the portability of Unix is that most of the Unix operating system is written in the high-level programming language C, which was specifically developed for the purpose of implementing Unix.

A Unix operating system consists of a kernel and a set of common utility programs. The kernel is the core of the operating system, which manages the computer hardware, provides essential facilities, such as the control of program executions, memory management, a file systems, and

mechanisms for exchanging data with attached devices, other programs, or a network. The utility programs provide user level commands, such as those to create and edit files.

Since its inception, many different versions of Unix-like operating systems have emerged. Most computer companies have developed their own flavor of Unix, with names such as AIX (IBM), HP-UX (Hewlett Packard), SunOS and Solaris (Sun Microsystems), Ultrix (DEC), Xenix (Microsoft Systems), and many others. By the early 1990s, when PCs had become powerful enough to run a full Unix-like operating systems, Unix versions for PCs started to emerge, including, FreeBSD, NetBSD, OpenBSD, and Linux. All *BSD systems are based on BSD4.x, an influential version of Unix from the University of California at Berkeley. Differently, Linux constitutes a new branch of Unix, which was created by the Finnish computer science student Linus Torvalds. Most of the Unix versions for PCs were distributed freely and with source code. Bundled with other free software, particularly, the *GNU* software, which includes editors, compilers, and applications, and the *X Windows* graphical user interface, Unix PCs have become an alternative to the Microsoft Windows platform.

The Linux operating system is distributed by organizations that package Linux with other, sometimes proprietary, software. Popular Linux distributions include Red Hat Linux, Caldera Linux, and Debian Linux. For the most part, the different distributions of Linux are quite similar. However, there are differences in the configuration files, that is, the files that contain system parameters, which are read when the system or a server on the system are started. Since many lab exercises deal with changing configuration files, the lab experiments are bound to a certain distribution. We have selected Red Hat release version 7.1, which runs the Linux kernel version 2.4. At the time of this writing, Red Hat 7.1 is the latest version of Red Hat which can run all of the software utilities used in the Internet lab.

We want to point out that most of the description of Linux features in this chapter, such as the file system and the commands, applies to other Unix systems.

What is X?

On most Unix systems, a user interacts with the operating system via a graphical window system, that takes input from the keyboard and the mouse, and displays output on windows on the monitor. Virtually all window systems for Unix systems are based on the *X Windows system*, sometimes simply called *X* or *X11*, which was developed at MIT in the mid-1980s. The layout and appearance of windows in *X11* is set by a separate component, called *window manager*. In recent years, window managers for Unix systems have been enhanced to *desktop environments*, which add software tools and applications to the window manager.

In this book we will use the *Gnome desktop* with the *Enlightenment window manager*, one of the popular desktop environments for Linux systems. The lab exercises do not require a specific

window manager or desktop environment. All labs can be completed with any window manager or desktop environment for X11.

In the following, we assume that Red Hat Linux and an X11 window manager are installed on all PCs in the Internet Lab. If you need to install or customize your Linux system, refer to the instructions in Appendix **A**.

2.1.   Logging in

Linux is a multiuser operating system where multiple users can work on the same system at the same time. Linux uses accounts to administer access to the system. Before you can work on a Linux system you must provide an account name (*login name*) and a password. This process is referred to as "logging in". Each Unix system has a special account, with login name *root*. The root account is reserved for administrative tasks and has the highest privileges. The root user, often called *root*, can access all files and all programs, delete all files, create or delete accounts, and change configuration files. In short, the root user can do anything on the system.

Most lab exercises require you to make changes to the configuration of the Linux system, or run programs which require the privileges of the root user. Therefore, whenever you access the PCs in the Internet lab, you log in as the root user. A risk of logging in as root is that a single inadvertent action may render the system useless and require a new installation of the operating system. Some actions may even damage the hardware of the system. The most probable cause of problems is the removal of a file that is required by the operating system. Therefore, whenever you are logged in as root, exercise caution when deleting files.

Before you login to a Linux Red Hat that has the Gnome desktop environment installed, you see a window as shown in Figure 1.11. Other X11 window managers or desktop environments show a similar window. To login as root, type **root** in the login field and press the enter key. When prompted for the password, type the root password and press the enter key. If the password is correct, you will see a desktop similar to the desktop shown in Figure 1.12.

Clearly, you must know the root password to log in as root user. If you have installed Linux on the system, you have selected a root password during the installation procedure. Otherwise, someone must have given you the root password.
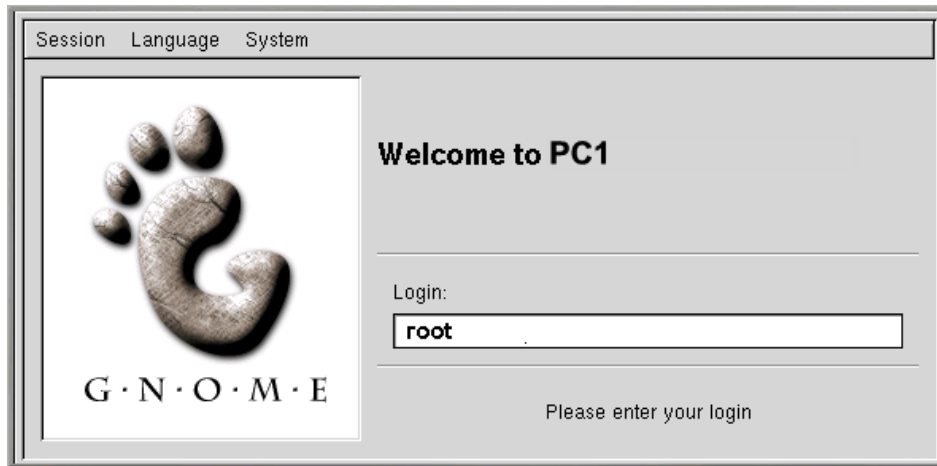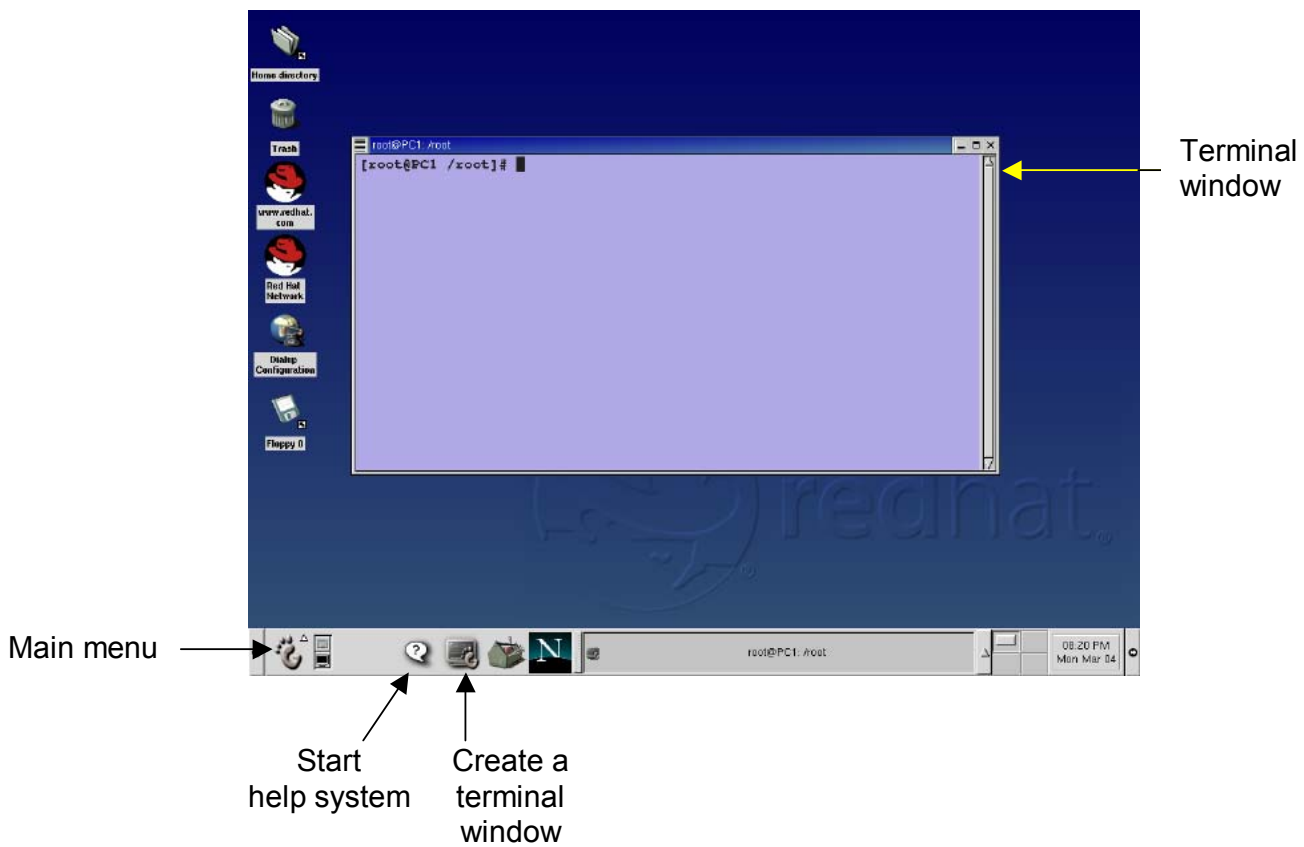
Figure 1.11. Login window.

Terminal window

Main menu

Start help system

Create a terminal window

Figure 1.12. Snapshot of a Gnome desktop.

2.2.   Navigating the Desktop

There are only a few things that you need to know about the Gnome desktop environment. Here is a description of tasks that you may find useful.

- **Opening a terminal window:** To use the command line interface of Linux you need to create a terminal window. A new terminal window is created by clicking on a button in the panel at the bottom on the desktop, as shown in Figure 1.12. The center of the desktop in Figure 1.12 shows a terminal window.

- **Working with windows on the desktop:** You move a window on the desktop by selecting the top bar of a window and dragging the window to its new position. You can hide, maximize, and destroy a window by clicking in one of the buttons in the top bar of a window.

- **Cutting and pasting text:** Most X11 windows managers, including the windows manager of the Gnome desktops, have a simple feature for copying and pasting text. Select text with the left mouse button, move the mouse to the desired position in the same or a different window, and paste the copied text by pressing the middle mouse button. With a two-button mouse, you can paste by pushing both buttons simultaneously.

- **Logging out:** At the end of each lab session, you should log out of the root account. In the Gnome desktop, you log out by clicking on the main menu button (see Figure 1.12). In the displayed menu, shown in Figure 12(b), select *Log out.* This will display a window on the desktop as shown in Figure 12(b).   Selecting the *Logout* button logs you out of the root account.

- **Getting help:** The Gnome desktop has an online help system. The system is started by clicking on the question mark button in the panel of the Gnome desktop (see Figure 1.12).

(a)   (b)

Figure 1.13. Logging out in Gnome.

2.3.   The Linux File System

Just like any modern operating systems, Linux organizes files as a hierarchical tree of directories. Figure 1.14 shows a snapshot of a hierarchy of directories in Linux. Linux uses a single hierarchy, and the directory at the top of the tree is called the *root directory*.
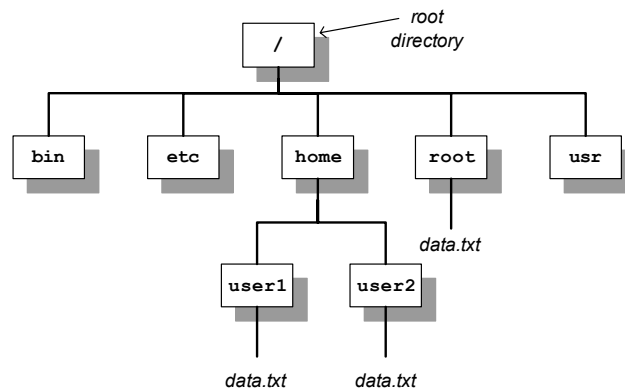

Figure 1.14. Snapshot of a Linux directory hierarchy.

Each file and directory in a Linux file system can be uniquely identified by a *pathname*. Pathnames can be absolute or relative. Absolute pathnames start at the root directory. The absolute pathname of the root directory is a slash *(/)* character. In the file hierarchy in Figure 1.14, the absolute

17

pathname of directory *home* in the root directory is */home*, that of directory *user1* in */home* is */home/user1*, and the absolute pathname of file *data.txt* in */home/user1* is */home/user1/data.txt*.

Pathnames that do not begin with a slash are relative pathnames and are interpreted relative to a *current (working) directory*. For example, if the current directory is */home*, then the pathname *user1/data.txt* refers to the absolute pathname */home/user1/data.txt*.

When using relative pathnames, a single dot (.) denotes the current directory and two dots (..)denote the *parent directory*, which is the directory immediately above the current directory. With the parent directory, it is feasible to identify each file with relative pathnames. Using the example in Figure 1.14, if the current directory is */home/user1*, the relative pathname .. refers to directory */home,* the pathname ../.. refers to the root directory, and the pathname *../user2/data.txt* refers to the file */home/user2/data.txt.*

Each Linux account has a *home directory*. For regular accounts (that is, accounts which different from the root account) the home directories are located in */home*. So, */home/user1* is the home directory for an account with login *user1*. The home directory of the root account is */root*. Whenever a new terminal window is created, the current directory in the terminal window is the home directory. Since you log in as root in the Internet lab, this is directory */root*.

A more complete list of the top level in a Linux file systems is shown in Figure 1.15. Most of these directories and files can only be modified by root. Linux configuration files are located in directories /etc, /usr/etc, /var and their subdirectories. Whenever you modify the configuration of a Linux system, you will work on files in these directories.

Each file and each directory has an owner. A regular user only owns the home directory and all files that are created by the user. The root is the owner of all other files on the system.

In Linux, each file has a set of access permissions. The permissions are *read ("r"), write ("w"),* and *execute ("x"),* and give, respectively, permission to read the contents of a file, modify the file, or execute the file as a program. Permissions are set by the owner of a file. Linux specifies access permissions separately for the owner of the file, a user group which is associated the file, and the set of all users. So, the owner of a file can set the permissions so that all users can read the files, but only the owner can modify the file.

The root user can ignore all access permissions and can even change the ownership of files. Since all lab experiments are run from the root account, access permissions are not important for running experiments in the Internet Lab. The downside of not having to worry about access permissions is that there is no protection against accidentally deleting or corrupting files.

When using a floppy disk or a CD-ROM on a Linux system, the media (floppy disk or CD-ROM) can be attached to the directory tree of the Linux system. Linux expects that the external media be

formatted as a hierarchical file system that is recognized by Linux, complete with root directory.[3] The process of adding an external file is illustrated in Figure 1.16. The figure shows a Linux system on a floppy disk which is mounted to an existing Linux file system. After mounting the floppy disk, the files on the floppy disk are available through the pathname */mnt/floppy*. The commands for mounting a floppy disk are discussed in a lab exercise.
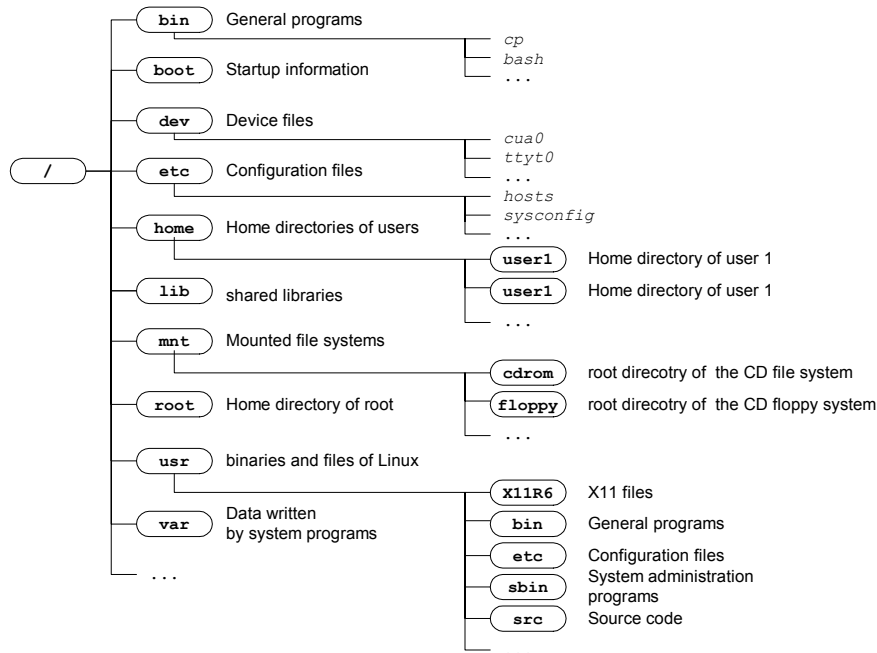

Figure 1.15. Main directories of the hierarchical Linux directory structure.
Directories are shown as boxes and file names are written in italics

---

[3] There are numerous file system formats for Unix-like systems. In addition, Linux recognizes other file systems formats, e.g., for MSDOS, Music CDs, and many more.
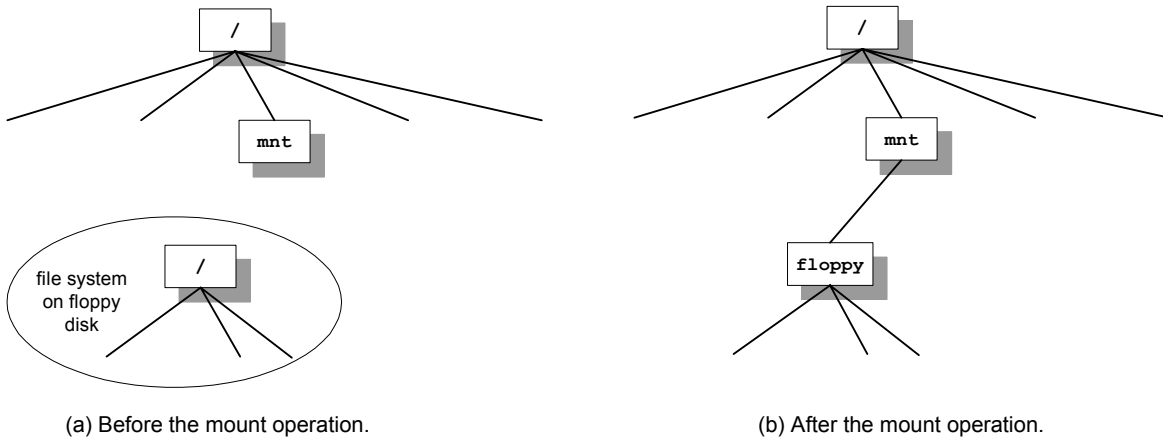
|(a) Before the mount operation.|(b) After the mount operation.|

Figure 1.16. Mounting a file system on a floppy disk.

2.4.   Linux Devices and Network Interfaces

In Linux, hardware devices, such as, disks, keyboard, and the mouse, are represented by files, called device files, that reside in the directory */dev*. For example, the mouse of a PC is represented by the device file */dev/mouse*. With device files, communication with an external device is similar to reading and writing from and to a file. When data is written to or read from a device file, Linux communicates with a device driver that is associated with the device file. The device driver communicates and controls a hardware device. In the Internet Lab, you will work with a number of different device files. For example, you will access the serial ports of the PCs via device files */dev/ttyS0* or */dev/ttyS1* and the floppy disk drive will be accessed via */dev/fd0*.

The software abstraction through which the Linux kernel accesses networking hardware is that of a *network interface.* For example, when assigning an IP address to an Ethernet interface cards, one manipulates the configuration parameters of the network interface which represents the Ethernet card. Just like other devices, each network interface is associated with a device driver. In most Unix-like operating systems, a network interface is implemented as a device files. This is different in Linux, where network interfaces are internally defined in the kernel. As a result, networking hardware is handled slightly differently from other hardware. In Linux, the names of network interfaces for Ethernet hardware are *eth0* for the first Ethernet interface card, and *eth1* for the second Ethernet interface card. There is a special network interface, the *loopback interface*, with name *lo*. The loopback interface is not connected to a real device, but is a virtual interface, which allows a PC to send messages to itself. In , the serial ports and ports of the Ethernet cards are

20

labeled with the interface names or names of device files that internal names that Linux uses to communicate with these ports.

2.5.   Linux Shell and commands

The command line user interface of the Linux operating system is called a *shell.* A shell is a program that interprets and executes Linux commands which are typed in a window terminal. Whenever you create a new terminal window, a shell is started. The shell displays a prompt at which the user can type in commands. The prompt can be as simple as

```
%
```

or the prompt can be set to provide additional information. For example, in the terminal in Figure 1.12, the prompt

```
[root@PC1 /root]#
```

displays the user, the name of the computer, and the current directory. Throughout this book, we use the prompts "%", or "PC1%" if we want to indicate that this is a shell prompt at PC1. When you type a command at the prompt, and press the enter key, the shell interprets the command, and, if it is a valid Linux command, executes the command.

A shell is terminated by typing **exit** at the command prompt. If the shell is running in a terminal window, the terminal window disappears.

Linux offers a variety of shell programs with names such as *sh, csh, ksh, tcsh*, or *bash*. For the purposes of the material covered here, the differences between these shell programs are not relevant.

Next, we review some basic and commonly used Linux commands which are issued at a shell prompt. Commands in Linux have a common format: a command name, which may be followed by a set of options and arguments. For example, in the command **ls –l** *data.txt*, **ls** is the command, **-l** is an option which further specifies the command , and data.txt is an argument, which is often a file name. Options are generally preceded by a – (dash), and more multiple options can be specified in a single command.

The only built-in help feature of a Linux system are the online manual pages for Linux commands, called the *man pages*. The man pages offer detailed information on a command, however, are not always helpful unless you are familiar with the command, and need to lookup some details. Desktop environments, such as Gnome, may have additional documentation.

When you login to a PC in the Internet lab you may find that changes to the Linux system from a previous lab are still in effect. Restarting ("*rebooting*") the operating system removes all temporary configuration changes. Therefore, at the beginning of each lab you should reboot the Linux PCs.

Since all files in Linux are organized as a tree of directories, you need to become familiar with navigating and manipulating the directory tree. The following are essential commands that relate to Unix directories.

Directory commands:

pwd   Prints the current directory.

cd *dirpath*
cd

      Changes the current directory to the relative or absolute pathname of the directory *dirpath*. If no directory is given, the command changes the current directory to the home directory.

      For example, the command **cd** */usr/bin*
      changes to directory */usr/bin*, the command **cd ..** changes to the parent directory, and the command **cd** without a parameter changes, if you are logged in as root, to directory */root*.

      mkdir *dirname*

      Creates a new directory with name *dirname* in the current directory.

      For example, the command **mkdir** *xyz* creates a new directory with name *xyz*.

rmdir *dirname*

      Deletes the directory *dirname* from the current directory. A directory must not contain any files when it is deleted, otherwise an error message is displayed.

Before discussing the commands to list and manipulate files, we introduce the *wildcard characters* * (star) and ? (question mark). The wildcard character * matches any sequence of zero or more characters, and ? matches any single character. Wildcard characters are useful to describe multiple files in a concise manner. For example, The text string *A*.txt* matches all file names that start with an *"A"* and end with *".txt",* e.g., *ABC.txt, A.txt,* and *Ab.txt* The text string A? matches all filenames that are two characters long and start with *"A",* e.g., *Ab.txt* and *A1.txt*

File commands:

ls
ls *dirname*

> Lists information about files and directories in the current directory. If the command has a directory name as argument (**ls** *dirname*), then the command lists the files in that directory.

> The **ls** command has several options. The most important is **ls –l**, which includes extensive information on each file, including, the access permissions, owner, file size, and the time when the file was last modified.

> For example, *ls /* lists all files and directories in the root directory; **ls** *AB** lists all files and directories in the current directory that start with *AB*; **ls –l** .. prints detailed information on each file and directory in the parent directory of the current directory.

mv *fname newfile*
mv *fname dirname*

> Renames or moves a file. Here, the file or directory *fname* is renamed as *newfile*. If the destination file (*newfile*) exists, then the content of the file is overwritten, and the old content of *newfile* is lost. If the first argument is a file name and the second argument is a directory name (*dirname*), the file is moved to the specified directory.

> For example, **mv** *data.txt text.txt* simply renames file *data.txt*, and *mv * /root* moves all files from the current directory to directory */root* (and gives an error message if the current directory is */root*).

cp *fname newfile*
cp *fname dirname*

> Copies the content of file *fname* to *newfile*. If a file with name *newfile* exists the content of that file is overwritten. If the second argument is a directory, then a copy of *fname* is created in directory *dirname*.

> For example, **cp** **.txt /tmp* creates a copy of all files that end with *".txt"* in directory */tmp*.

rm *fname*

> Removes a file. Once removed, the file cannot be recovered. For example, **rm** ***
> removes all files in the current directory.

Note: Linux may not issue a warning when a file is overwritten or when a file is removed. When
> you use the option **–i**, Linux asks for confirmation before overwriting or deleting files. It
> is strongly recommend that you use **cp –i** instead of **cp**, **mv – i** instead of **mv**, and **rm –i**

instead of **rm**. Many shells are configured to always use the **-i** option.

An important thing to have in mind is that Linux does not have an *undo* command that reverses the effects of a previously issued command.

In many lab exercises you need to modify the content of configuration files. For this, the following commands are helpful.

<u>Commands to view and modify the text files:</u>

more *fname*

> Displays the contents of file *fname*, one page at a time. The display can be scrolled with the '↑' and '↓' keys, and the 'Page Up' and 'Page Down' keys. Keyboard controls are space bar or *'f'* for the next page, *'b'* for the previous page, and *'q'* to end the display.

cat *fname*

> This is similar to the more command, but the file is displayed without stopping at the end of each page.

gedit *fname*
gedit

> This command opens the file *fname* in the text editor **gedit**. A new *textfile* can be written by running **gedit** without an argument. A text editor is used to view or modify the content of a text file.

We note that Linux has a wide variety of editors available that can be used to modify text files. Some of the most popular text editors are **vi, emacs, pico,** and **gedit**. We recommend the **gedit** editor if you have never worked with a text editor on a Unix-like system. To edit the file */etc/hosts* with **gedit**, simply type:

```
PC1% gedit /etc/hosts
```

The **gedit** application is shown in Figure 1.17. To modify the file simply click on a location n the text window and type text. You can type **Ctrl-c** for copying highlighted text and **Ctrl-v** for pasting text. To save the changes, press the *Save* button. To terminate the exit the application press the *Exit* button. If you have push *Exit* and have not saved changes to the file, the editor will ask whether you want to save the changes.
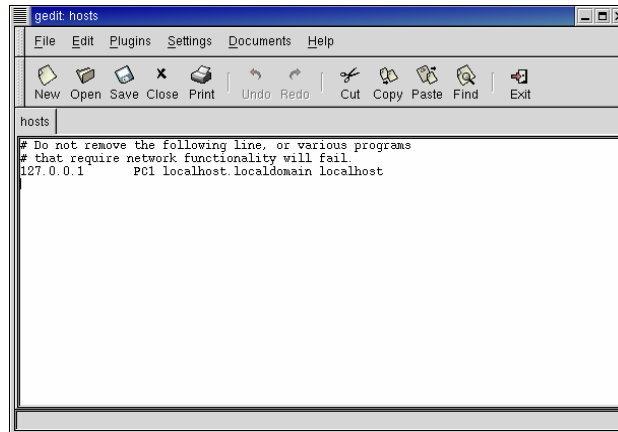
Figure 1.17. The gedit text editor.

Many lab experiments ask you to save data that is displayed in a terminal window to a file. The following commands show how to redirect the output of a terminal window to a file.

Redirecting the output of programs

cmd > fname

> The output of *cmd* is written to file *fname*. The file is created if it doesn't already exist, and the contents is overwritten if the file exists. For example, the command *ls > mlist.txt* writes a listing of the current directory in file *mylist.txt*.

cmd >> fname

> The >> operator appends the output of command *cmd* to the end of file *fname*.

> For example, the command, *ls >> mlist.txt* appends a listing of the current directory to file *mylist.txt*.

*cmd | **tee** fname*
*cmd > fname **& tail –f** fname*

> Both commands have the effect that the output of command *cmd* is displayed in the terminal window, and also written to file *fname*. The file is created if it doesn't exist, or the content is overwritten if the file exists.

> For example, the command **ls | tee** *mylist.txt* displays the listing of the current directory on the screen, and writes a listing of the current directory to file *mylist.txt*.

26

In Linux, each terminal window can run multiple commands at the same time. Also, it is possible to stop a command temporarily and resume it at a later time. In each terminal window, one command can be run as a *foreground process* and multiple command can be run as *background processes*. When a command is issued from the prompt, say

```
% gedit
```

the command **gedit** is started in the foreground. When a command is running in the foreground, no shell prompt is displayed until the command is finished. The same command can be run *in the background* by adding an ampersand ('&') at the end of the command, as follows:

```
% gedit &
```

If a command is executed in the background, the shell prints a prompt for the next command without any delay. Using background commands you can run multiple commands from a single terminal window.

You can switch a command that is running in the foreground to the background and vice-versa. Switching a command from the foreground to the background is done as follows:

```
% gedit
Ctrl-z
% bg
%
```

Here, gedit is the command in the foreground. Typing **Ctrl-z** stops the command, and **bg** resumes the stopped command in the background. To switch a command from the background to the foreground, type

```
% jobs
```

The command **jobs** lists all commands that are currently running in the background or are stopped, e.g., with Ctrl-z. The command

```
% fg %1
```

resumes the first command in the foreground. The following are a set of Linux commands that control the execution of commands.

Control of commands:

Ctrl-c

> Pressing **Ctrl-c** (CTRL + C) terminates the command running in the foreground.

Ctrl-z

> Pressing the **Ctrl-c** stops the commands in the foreground.

cmd **&**

> Executes the command *cmd* in the background.

jobs

> Lists all background and stopped commands of the current user, and assigns a number to each command.

fg *%n*
fg   Resumes the *n*-th command of the user that is either stopped or running in the in the foreground. The numbers are as displayed by the jobs command. If no number is given the command refers to the command that was last running started or stopped.

bg *%n*
bg   Resumes the *n*-th command of the user that is either stopped in the background. If no number is given the command refers to the command that was last running started or stopped.

kill *%n*

> Terminates the *n*-th command, where the number is as displayed by the command **jobs**.

pkill *cmd*

> Terminates a process that executes the command with name *cmd*.

3. Applications, Utilities, and Tools

We next describe some of the software tools and applications that are used in Lab 1. The lab uses the remote login protocol **Telnet** and the file transfer protocol **FTP,** which were introduced in Chapter 0. **Ping** is a simple, but very powerful debugging utility. Finally, the lab features two protocol analyzers, **tcpdump** and **ethereal.** Both tools capture network traffic, and display the protocol headers of captured traffic.

3.1. Running a Telnet Session

To establish a Telnet session to a host with name PC2 at IP address 10.0.1.12, you simply type the command **telnet 10.0.1.2.** Assuming that a Telnet server is running at PC2, you are prompted for a login name and a password. If the login is successful, you see a shell prompt from PC2, and can issue Linux commands. A Telnet session is terminated by typing **exit** at the command prompt. Figure 1.18 shows the output from a short Telnet session from PC1 to PC2.

```
Issue telnet command on PC1 ──────   PC1% telnet 10.0.1.12

                                     Trying 10.0.1.12...
                                     Connected to 10.0.1.12 (10.0.1.12).
                                     Escape character is '^]'.
Login to remote host                 Red Hat Linux release 7.1 (Seawolf)
                                     Kernel 2.4.2 on an i686
                                     login: root
                                     Password:

Shell prompt at PC2 ──────           Last login: Fri Mar  8 21:18:32 from 10.0.1.11
                                     PC2%

End telnet session ──────            PC2%  exit
                                     Connection closed by foreign host.

Shell prompt at PC1 ──────           PC1%
```

Figure 1.18. Telnet session.

```
Issue ftp command on PC1 ──────      PC1% ftp 10.0.1.12

                                     Connected to 10.0.1.12.
                                     220 PC2 FTP server (Version wu-2.6.1-16) ready.
Login to                             Name (10.0.1.12:root): root
remote ftp server                    Password:
                                     Remote system type is UNIX.
                                     Using binary mode to transfer files.

                                     ftp> get x.txt
Download file x.txt                  local: x.txt remote: x.txt
                                     226 Transfer complete.
                                     61 bytes received in 0.00062 seconds (96 Kbytes/s)

End ftp session ──────               ftp> quit

Shell prompt at PC1 ──────           PC1%
```

Figure 1.19. FTP session. (Some system messages are omitted.)

29

3.2.   Running an FTP Session

An FTP session from PC1 to PC2 (where PC2 has IP address 10.0.1.12) is initiated by typing the command **ftp 10.0.1.2.** Similarly as in Telnet, the user at PC1 is prompted by PC2 for a login name and a password. If the login is successful, the user at PC1 sees a command prompt: **ftp>**. The FTP prompt accepts a limited set of commands which can be used to download files from PC2 to PC1 or to upload files from PC1 to PC2. The command **get** is used to download a file, and the command **put** is used to upload a file. The main commands that can be entered at the FTP prompt are summarized in the following list.

FTP commands:

ls   Lists the content of the current directory on the remote FTP server. After
                         logging in, the current directory is the home directory of the user.

!ls   Lists the content of the current directory on the local system.

cd *dirname*   Changes the current directory at the remote system to *dirname*

lcd *dirname*   Changes the current directory at the local system to *dirname*

binary
ascii   FTP transfers files either as text files or as binary files. The default mode of
                         data transfer is ASCII, which is suitable for transferring text files. Before
                         transferring a file that is not a text file, e.g., a JPEG image or a compiled
                         program, the transfer mode must be switched to binary mode with the
                         command **binary**. The command **ascii** switches back to text mode.

get *fname*
get *fname fname2*   Downloads the file with name *fname* from the current remote directory to the
                         current local directory. If a file with name *fname* exists in the local directory,
                         it is overwritten without issuing a warning. If the command has a second
                         filename as argument (*fname2*), the downloaded file is renamed as *fname2* on
                         the local system.

mget *fname*   Downloads multiple files if *fname* uses wildcard characters. For example, the
                         command **mget** *.txt downloads all files that end with *.txt*.

put *fname fname2*   Uploads file *fname* from the current local directory to the current remote
                         directory. If a file with name *fname* exists in the remote directory, it is
                         overwritten without issuing a warning. If the command has a second filename
                         as argument (*fname2*), the downloaded file is renamed as *fname2* on the
                         remote system.

30

mput *fname*	Uploads multiple files if *fname* uses wildcard characters. For example, the command **mput** **.txt* uploads all files that end with *.txt*.
quit	Ends the FTP session.
help	Lists all available commands.

3.3.   Ping

One of the most simple, but also most effective, tools to debug IP networks is the ping utility. Ping, which is sometimes said to be a short for Packet Internet Groper, but is actually named after the sound of a sonar tracking system, simply tests whether a given Internet host is reachable. Ping sends a short packet to an IP address and waits for response from that IP address. The packets that are issued during a ping are an ICMP Echo Request and an ICMP Echo Response message.[4] The ping command sends an ICMP Echo Request datagram to an interface, and expects an ICMP Echo Response datagram in return.
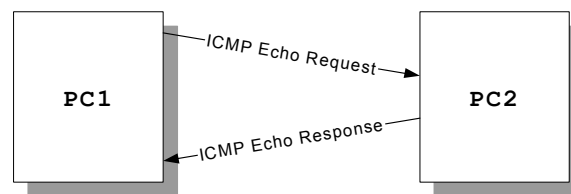

Figure 1.20. Ping command

When issuing a ping command, a Linux system measures and displays the time between the transmission of the ICMP Echo Request and the return of the ICMP Echo Response. However, the main information provided by ping is not the time delay, but whether a certain host is reachable at all. In most cases, if a ping command between two machines is successful, most Internet applications are likely to run without problems.

---

[4] ICMP packets are covered in Lab 2.

31

Figure 1.1

Figure 1.21. Running ping

Ping is the single most important tool of a network engineer to troubleshoot problems in a network configuration. Whenever you change the network setup in the lab, you should use ping to determine correctness of or troubleshoot the network configuration. Using ping effectively can sometimes extract a lot of information from a network.

ping *IPaddress*
> Issues a ping command for the host with the given IP address. The system will issues one ICMP Echo Request packet with a size of 56 bytes every second. The command is stopped by typing **Ctrl-c**.

ping -c *num IPaddress*
> The command stops after sending *num* ICMP Echo Requests and receiving *num* ICMP Echo Response packets, where *num* is a number

ping -f *IPaddress*
> Transmits ICMP Echo Response packets as quickly as possible.

ping -i *num IPaddress*
> The sender waits for *num* seconds between transmissions of ICMP Echo Request packets. The default value is 1 second.

ping -n *IPaddress*
> With this option, the output uses numeric IP addresses and does not display symbolic names of hosts.

ping -R *IPaddress*
> With this option, the traversed route of the ICMP packets is displayed. The display is limited to the IP addresses of the first nine hops of the route.

ping -s *num IPaddress*
> The number of data bytes in the ICMP Echo Request is set to *num* bytes. The default value is 56 bytes.

ping -v *IPaddress*
> Displays a verbose output.

3.4.   Network Protocol Analyzers

Since the all labs require to make observations of the behavior of network protocols, we need to have tools that can monitor the traffic that is generated by these protocols, and present it in a human readable form. In the labs we will use two tools to monitor and display network traffic: **tcpdump** and **ethereal**. Tools, such as tcpdump and ethereal, that capture and display traffic on a network interface are referred to as network protocol analyzers or *packet sniffers.*

Network protocol analyzers set the network interface card into a mode where the card captures all traffic that passes by the interface card. This mode is called a *promiscuous mode*. On an Ethernet segment, an Ethernet interface in promiscuous mode can capture the traffic transmitted by all stations on the segment. Because of the involved security issues, network traffic captures are restricted to the root user.

The software architecture of a network protocol analyzer on a Linux system with an Ethernet card is shown in Figure 1.22. The network protocol analyzer is running as an application which communicates with a component in the Linux kernel, called the *Linux socket filter*. The Linux socket filter acts as an agent between the protocol analyzer and the Ethernet device driver. It sets the Ethernet device driver in promiscuous mode and obtains a copy of all incoming traffic from network and all outgoing traffic to the network. The socket filter processes the traffic and passes it to the network protocol analyzer, which displays the traffic to the user.

It is important to be aware that network protocol analyzers such as *tcpdump* and ethereal do not work perfectly, and may miss a fair fraction of packets. On a lightly loaded system and on a network with low traffic volume, *tcpdump* and ethereal will rarely miss a packet. However, if the system on which the protocol analyzer is running is highly loaded or if the traffic volume on the network is high, *tpcdump* and ethereal may not be able to catch up and skip packets.
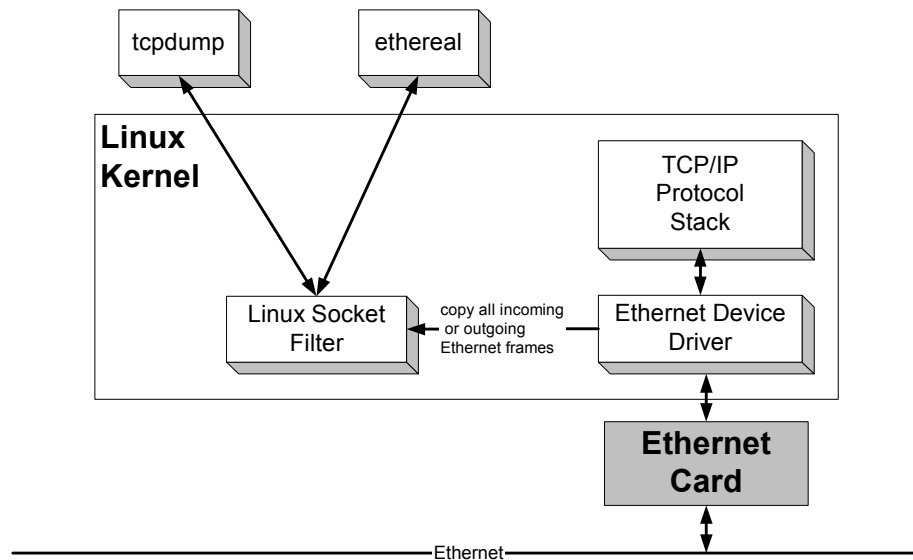
Figure 1.22. Implementation of network protocol analyzers in Linux.

3.4.1. Tcpdump

Tcpdump and the BSD packet filter, on which the Linux socket filter is based, were developed in the early 1990s at the Lawrence Berkeley National Laboratory. *Tcpdump* is started by running the command

```
%tcpdump
```

An example of the output of *tcpdump* is shown in Figure 1.23. The figure depicts the output for the FTP example from Chapter 0. The output from *tcpdump* is displayed in the terminal window where the program was started.

Tcpdump displays one line for each
transmitted or received Ethernet. In each line, *tcpdump* displays a timestamp and information that is derived from the protocol headers contained in the Ethernet frame. The timestamp "*16:54:51.340712*" corresponds to 4:54 PN and 51.340712 seconds. The fractions of second after the second digit may not be very accurate, since the system clocks on most PCs are reliable only for times that exceed 10-50 milliseconds. If the Ethernet frame is an IP datagram with UDP or TCP payload, then *tcpdump* displays information on the source and the destination of the frame. For example, the entry in Line 1 of Figure 1.23 *"128.143.137.144.1555 > 128.143.137.11.53"* indicates that the sender of the IP datagram is IP address 128.143.137.144 at port 1555 and the destination is *128.143.137.11* at port *53.* Even if a frame does not contain an IP datagram, *tcpdump*

attempts to interpret the payload. For example, in Figure 1.23, Lines 7 and 8 show that the payload of the frame is an ARP packet. After the IP addresses, *tcpdump* displays information from other protocol headers, such as, TCP, UDP, routing protocols, and other protocols. In Figure 1.23, Lines 1—6 the displayed information includes header information from DNS messages, and Lines 9—13 display information from TCP segment headers.

By default, *tcpdump* does not display the payload of a packet

```
%tcpdump
```

1. `16:54:51.340712 128.143.137.144.1555 > 128.143.137.11.53: 1+ A? neon.cs. (25)`
2. `16:54:51.341749 128.143.137.11.53 > 128.143.137.144.1555: 1 NXDomain* 0/1/0 (98) (DF)`
3. `16:54:51.342539 128.143.137.144.1556 > 128.143.137.11.53: 2+ (41)`
4. `16:54:51.343436 128.143.137.11.53 > 128.143.137.144.1556: 2 NXDomain* 0/1/0 (109) (DF)`
5. `16:54:51.344147 128.143.137.144.1557 > 128.143.137.11.53: 3+ (38)`
6. `16:54:51.345220 128.143.137.11.53 > 128.143.137.144.1557: 3* 1/1/2 (122) (DF)`
7. `16:54:51.350996 arp who-has 128.143.71.21 tell 128.143.137.144`
8. `16:54:51.351614 arp reply 128.143.71.21 is-at 0:e0:f9:23:a8:20`
9. `16:54:51.351712 128.143.137.144.1558 > 128.143.71.21.21: S 607568:607568(0)`
   `                                              win 8192 <mss 1460> (DF)`
10. `16:54:51.352895 128.143.71.21.21 > 128.143.137.144.1558: S 3964010655:3964010655(0)`
    `                                              ack 607569 win 17520 <mss 1460> (DF)`
11. `16:54:51.353007 128.143.137.144.1558 > 128.143.71.21.21: . ack 1 win 8760 (DF)`
12. `16:54:51.365603 128.143.71.21.21 > 128.143.137.144.1558: P 1:60(59)`
    `                                              ack 1 win 17520 (DF) [tos 0x10]`
13. `16:54:51.507399 128.143.137.144.1558 > 128.143.71.21.21: . ack 60 win 8701 (DF)`

Timestamp   Source IP address and destination IP address   Packet headers from other protocols

Figure 1.23. Output of **tcpdump**.

The following list shows different uses of the *tcpdump* command. In the next chapter, we discuss how to specify expressions in the *tcpdump* command line so that *tcpdump* only captures a certain type of frames, e.g., with a given IP address or a certain protocol.

tcpdump -i *interface*
> Specifies that tcpdump is started on the given interface. This option should be used on systems with multiple network interfaces. For example, **tcpdump –i** *eth0* starts tcpdump on interface *eth0*.

tcpdump -n
> With the **–n** option, **tcpdump** does not print host names, but prints the IP addresses in the packet. We recommend to always set the **–n** option, since resolving host names from the IP addresses may have the undesirable effect that **tcpdump** sends DNS messages, that is, **tcpdump** may generate traffic on its own.

tcpdump -x

> With this option, the first 68 bytes of the captured packet are displayed in hexadecimal form.

tcpdump -t

> This option suppresses the display of a timestamp at the beginning of each line.

tcpdump -vv

> Prints additional information on each packet.

tcpdump –l

> Buffers the output to the terminal window and enables to save output to a file. When saving the output of tcpdump to file *fname* use the command
> **tcpdump -l | tee** *fname*
> or
> **tcpdump -l >** *fname* **& tail -f** *fname*

Note: As always, multiple options can be used in the same command line. For example, the command tcpdump –i eth0 –n –x – t –vv enables all of the above options.

3.4.2. Ethereal

Ethereal is a protocol analyzer with a graphical user interface and can be run on Unix-like operating systems as well as other operating systems. Ethereal recognizes a large number of protocols, and new protocols are constantly being added. Ethereal is be the main tool for capturing traffic in the Internet Lab exercises.

Ethernet is started from a terminal window with the command

```
% ethereal
```

The command displays a window as shown in Figure 1.24. The traffic capture is started by selecting *Capture*:*Start* in the main menu of the window. Once the traffic capture is started, the ethereal window displays the traffic in three views. The first view shows a summary of the captured packets. There is one line for each packet and one of these packets can be highlighted. In Figure 1.24, the first packet, an Echo Request ICMP message, is highlighted. The second view shows the protocol header details from the highlighted packet. Packet headers can be expanded and hidden. In Figure 1.24, the Ethernet header is expanded and the other header are hidden. The third

view shows the hexadecimal and ASCII representation of the packet header in the second view. The traffic captured by ethereal can be saved to a file by selecting *File:Print* in the main menu.

With Ethereal, one can specify the traffic to be captured by selecting a *capture filter*. Also, it is feasible to display a subset of the captured traffic by specifying a *display filter*. Setting capture filters and display filters is discussed in Lab 2.
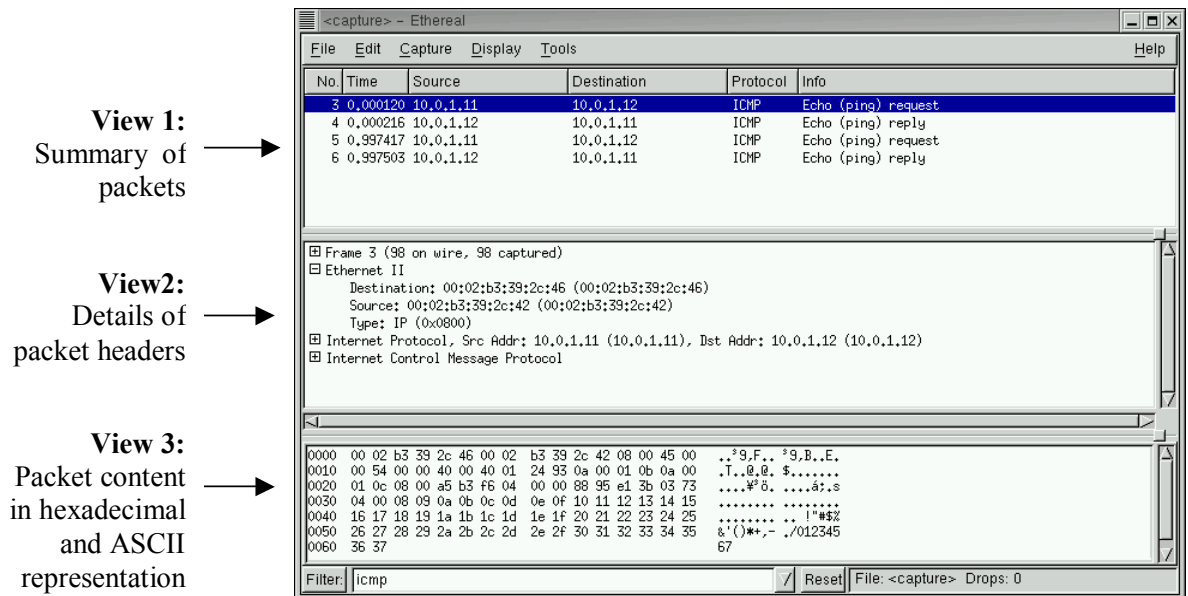

Figure 1.24. Ethereal window.

BIBLIOGRAPHY

W. Richard Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.

Richard Sharpe, Ed Warnicke, Ethereal User's Guide V1.1 for Ethereal 0.8.19,
http://www.ethereal.com/docs/user-guide/, 2001.

Steven McCanne and Van Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture, Proceedings of the 1993 Winter USENIX Conference, Pages 259-270, 1993.

Olaf Kirch, Terry Dawson, Linux Network Administrators Guide, Second Edition, O'Reilly and Associates, 2000.

Evi Nemeth, Garth Snyder, Scott Seebass Trent R. Hein, Unix System Administration Handbook, Third Edition, Prentice Hall, 2000.

Bill McCarty, Learning Red Hat Linux, Second Edition, O'Reilly and Associates, 2002.