CHAPTER 6

Network Sniffers

You can now properly secure and harden your systems and test your network for security vulnerabilities using proactive tools that help to keep your network healthy and secure. Now we will look at some tools that help you to act and react if you have a computer attack or security issue on your network in spite of all your preparations. Network sniffers fit into this category along with intrusion detection systems and wireless sniffers.

Chapter Overview

Concepts you will learn:

- Network sniffer fundamentals
- Ethernet history and operation
- How to do safe and ethical network sniffing
- Sample sniffer configurations
- Network sniffer applications

Tools you will use:

Tcpdump, WinDump, and Ethereal

Simply put, a **network sniffer** listens or "sniffs" packets on a specified physical network segment. This lets you analyze the traffic for patterns, troubleshoot specific problems, and spot suspicious behavior. A **network intrusion detection system** (NIDS) is nothing more than a sophisticated sniffer that compares each packet on the wire to a database of known bad traffic, just like an anti-virus program does with files on your computer.

Sniffers operate at a lower level than all of the tools described thus far. Referring to the OSI Reference model, sniffers inspect the two lowest levels, the physical and data link layers.

OSI Layer Number	Layer Name	Sample Protocols
Layer 7	Application	DNS, FTP, HTTP, SMTP, SNMP, Telnet
Layer 6	Presentation	XDR
Layer 5	Session	Named Pipes, RPC
Layer 4	Transport	NetBIOS, TCP, UDP
Layer 3	Network	ARP, IP, IPX, OSPF
Layer 2	Data Link	Arcnet, Ethernet, Token Ring
Layer 1	Physical	Coaxial, Fiber Optic, UTP

The physical layer is the actual physical cabling or other media used to create the network. The data link layer is where data is first encoded to travel over some specific medium. The data link layer network standards include 802.11 wireless, Arcnet, coaxial cable, Ethernet, Token Ring, and many others. Sniffers are generally specific to the type of network they work on. For example, you must have an Ethernet sniffer to analyze traffic on an Ethernet LAN.

There are commercial-grade sniffers available from manufacturers such as Fluke, Network General, and others. These are usually dedicated hardware devices and can run into the tens of thousands of dollars. While these hardware tools can provide a much deeper level of analysis, you can build an inexpensive network sniffer using open source software and a low-end Intel PC.

This chapter reviews several open source Ethernet sniffers. I chose to feature Ethernet in this chapter because it is the most widely deployed protocol used in local area networks. The chances are that your company uses an Ethernet network or interacts with companies that do.

It used to be that the network world was very fragmented when it came to physical and data link layer transmission standards; there was no one dominant standard for LANs. IBM made their Token Ring topology standard for their LAN PCs. Many companies that used primarily IBM equipment used Token Ring because they had no other choice. Arcnet was popular with smaller companies because of its lower cost. Ethernet dominated the university and research environment. There were many other protocols, such as Apple's AppleTalk for Macintosh computers. These protocols were usually specific to a particular manufacturer. However, with the growth of the Internet, Ethernet began to become more and more popular. Equipment vendors began to standardize and focus on low-cost Ethernet cards, hubs, and switches. Today, Ethernet has become the de facto standard for local area networks and the Internet. Most companies and organizations choose it because of its low cost and interoperability.

A Brief History of Ethernet

Bob Metcalfe invented Ethernet in 1973 while at the Xerox Palo Alto Research Center. (This same innovative place also fostered the invention of the laser printer and the graphical user interface, among other things.) Bob and his team developed and patented a "multipoint data connection system with collision detection" that later became known as Ethernet. Bob went on to form a company specifically dedicated to building equipment for this new protocol. This company eventually became 3Com, one of the largest network companies in the world. Luckily, Ethernet was released into the public domain so other companies could build to the specification. This was not true of Token Ring and most of the other network protocols of the day. If Ethernet had been kept proprietary or limited to only one company's hardware, it probably wouldn't have developed into the dominant standard it is today. It was eventually adopted as an official standard by the International Electrical and Electronic Engineers (IEEE), which all but assured it wide acceptance by corporate and government users worldwide. Other standards have been developed based on Ethernet, such as Fast Ethernet, Gigabit Ethernet, and Wi-Fi.

Ethernet handles both the physical media control and the software encoding for data going onto a network. Since Ethernet is a broadcast topology, where every computer can potentially "talk" at once, it has a mechanism to handle collisions—when data packets from two computers are transmitted at the same time. If a collision is detected, both sides retransmit the data after a random delay. This works pretty well most of the time. However, this is also a downside to the Ethernet architecture. All computers attached to an Ethernet network are broadcasting on the same physical wire, and an Ethernet card on the network sees all the traffic passing it. The Ethernet card is designed to process only packets addressed to it, but you can clearly see the security implication here.

Imagine if the way the postal system worked was that a bag containing all the mail was dropped off at the end of the street and each resident picked through it for their mail and then passed it along. (It might be interesting to see who subscribed to *Playboy* and who was getting the past due notices.) This fictional system is not very secure nor does it make efficient use of everyone's time, but that is essentially how Ethernet was designed.

Nowadays, most Ethernet networks are switched to improve efficiency. This means that instead of each Ethernet port seeing all the traffic, it sees only traffic intended for the machine plugged into it. This helps alleviate some of the privacy and congestion issues, but plenty of broadcast traffic still goes to every port. Broadcast traffic is sent out to every port on the network usually for discovery or informational purposes. This happens with protocols such as DHCP, where the machine sends out a broadcast looking for any DHCP servers on the network to get an address from. Machines running Microsoft Windows are also notorious for putting a lot of broadcast traffic on the LAN.

Other broadcast types are often seen on Ethernet LANs. One is **Address Resolution Protocol** (ARP); this is when a machine first tries to figure out which MAC address relates to which IP address (see the sidebar on MAC addresses in Chapter 3). Ethernet networks use an addressing scheme called **Medium Access Control** (MAC) addresses. They are 12-digit hexadecimal numbers, and are assigned to the card at the factory. Every manufacturer has its own range of numbers, so you can usually tell who made the card by looking at the MAC address. If a machine has an IP address but not the Ethernet address, it will send out ARP packets asking, "Who has this address?" When the machine receives a reply, it can then send the rest of the communication to the proper MAC address. It is this kind of traffic that make Ethernet LANs still susceptible to sniffer attacks even when they use switching instead of broadcasting all traffic to every port. Additionally, if hackers can get access to the switch (these devices are often poorly secured), they can sometimes turn their own ports into a "monitor" or "mirror" port that shows traffic from other ports.

Considerations for Network Sniffing

In order to do ethical and productive sniffing, you should follow the following guidelines.

Always Get Permission

Network sniffing, like many other security functions, has the potential for abuse. By capturing every transmission on the wire, you are very likely to see passwords for various systems, contents of e-mails, and other sensitive data, both internal and external, since most systems don't encrypt their traffic on a local LAN. This data, in the wrong hands, could obviously lead to serious security breaches. In addition, it could be a violation of your employees' privacy, depending on your company policies. For example, you might observe employees logging into their employee benefits or 401(k) accounts. Always get written permission from a supervisor, and preferably upper management, before you start this kind of activity. And you should consider what to do with the data after getting it. Besides passwords, it may contain other sensitive data. Generally, network-sniffing logs should be purged from your system unless they are needed for a criminal or civil prosecution. There are documented cases of well-intentioned system administrators being fired for capturing data in this manner without permission.

Understand Your Network Topology

Make sure you fully understand the physical and logical layout of your network before setting up your sniffer. Sniffing from the wrong place on the network will cause you either to not see what you are looking for or to get erroneous results. Make sure there is not a router between your sniffing workstation and what you are trying to observe. Routers will only direct traffic onto a network segment if it is addressed to a node located there. Also, if you are on a switched network, you will need to configure the port you are plugged into to be a "monitor" or "mirror" port. Various manufacturers use different terminology, but basically you need the port to act like a hub rather than a switch, so it should see all the traffic on that switch, not just what is addressed to your workstation. Without this setting, all your monitor port will see is the traffic addressed to the specific port you are plugged into and the network's broadcast traffic.

Use Tight Search Criteria

Depending on what you are looking for, using an open filter (that is, seeing everything) will make the output data voluminous and hard to analyze. Use specific search criteria to narrow down the output that your sniffer shows. Even if you are not exactly sure what you are looking for, you can still write a filter to limit your search results. If you are looking for an internal machine, set your criteria to see only source addresses within your network. If you are trying to track down a specific type of traffic, say FTP traffic, then limit the results to only those on the port that application uses. Doing this will make your sniffer results much more usable.

Establish a Baseline for Your Network

If you use your sniffer to analyze your network during normal operation and record the summary results, you will then have a baseline to compare it to when you are trying to isolate a problem. The Ethereal sniffer discussed in this chapter creates several nice reports for this. You will also have some data to track your network utilization over time. You can use this data to decide when your network is becoming saturated and what the primary causes are. It might be a busy server, more users, or a change in the type of traffic. If you know what you started with, you can more easily tell what and where your culprit is.

Tcpdump: An Ethernet Traffic Analyzer

Tcpdump

Author/primary contact:	University of California, Lawrence Berkeley Laboratories
Web site:	www.tcpdump.org
Platforms:	Most Unix
License:	BSD
Version Reviewed:	3.8.1

Mailing lists:

tcpdump-announce@tcpdump.org

This list is for announcements only.

tcpdump-workers@tcpdump.org

This list is for discussion of code. It will also receive announcements, so if you subscribe to this list you don't need to subscribe to the other one.

Both lists are archived, so you can search the old postings. The code discussion list is also available in a weekly summary digest format.

There are many sniffers available, both free and commercial, but Tcpdump is the most widely available and inexpensive. It comes with most UNIX distributions, including Linux and BSD. In fact, if you have a fairly current Linux distribution, chances are you already have Tcpdump installed and ready to go.

Installing Tcpdump

Tcpdump does exactly what its name implies: it dumps the contents of the TCP/IP packets passing through an interface to an output device, usually the screen or to a file.

1. In order for Tcpdump to work, it must be able to put your network card into what is called **promiscuous mode**. This means that the network card will intercept all traffic on the Ethernet wire, not just that addressed to it. Each operating system processes traffic from the Ethernet card in a different fashion. To provide a common reference for programmers, a library called **pcap** was created. On UNIX this is known as **libpcap** and on Windows as **WinPcap**. These low-level drivers can modify the way the card would normally handle traffic. They must be installed before you can install Tcpdump.

   If Tcpdump is already on your system, then you already have this driver installed. If not, they are provided on the CD-ROM that accompanies this book in the misc directory, or you can get them from the Tcpdump Web site. Make sure you install them *before* you install Tcpdump.

   Note: Libpcap also requires the Flex and Bison scripting languages, or Lex and Yacc as a substitute. If you don't have these, get them from your OS distribution disks or online and install them so libpcap will install successfully.
2. Install libpcap by unpacking it and issuing the standard compilation commands:

   ```
   ./configure
   make
   make install
   ```

   If you get a warning something like "Cannot determine packet capture interface" during the compilation process, then your network card doesn't support promiscuous mode operation and you will have to get another card to use Tcpdump. Most cards these days should support this mode of operation.
3. Once libpcap is installed, unpack the Tcpdump package and change to that directory.
4. Run the same compilation commands:

   ```
   ./configure
   make
   make install
   ```

   Now you are ready to use Tcpdump.

Running Tcpdump

There are a number of filter operations you can perform on the output to look for a specific type of traffic or lessen the overall amount of output. Indeed, on a busy network, unfiltered Tcpdump output will cause your screen to scroll faster than you can read it! However, for a quick demo of the power of Tcpdump, invoke it from the command line by simply typing:

```
tcpdump
```

You will see all the TCP traffic passing your machine's Ethernet card, unfiltered. It might look something like the example in Listing 6.1.

Listing 6.1 Tcpdump Example

```
12:25:38.504619 12.129.72.142.http > 192.168.1.3.3568: . ack
  1418369642 win 31856 <nop,nop,timestamp 72821542 25475802> (DF)

12:25:38.504758 192.168.1.3.3568 > 12.129.72.142.http: . ack
  1 win 40544 <nop,nop,timestamp 25486047 72811295> (DF)

12:25:38.507753 192.168.1.3.4870 > 65.83.241.167.domain:
  11414+ PTR? 142.72.129.12.in-addr.arpa. (44) (DF)

12:25:38.561481 65.83.241.167.domain > 192.168.1.3.4870:
  11414 NXDomain*- 0/1/0 (113)

12:25:38.562754 192.168.1.3.4870 > 65.83.241.167.domain:
  11415+ PTR? 3.1.168.192.in-addr.arpa. (42) (DF)

12:25:38.609588 65.83.241.167.domain > 192.168.1.3.4870:
  11415 NXDomain 0/1/0 (119)


12:25:38.610428 192.168.1.3.4870 > 65.83.241.167.domain:
 1416+ PTR? 167.241.83.65.in-addr.arpa. (44) (DF)

12:25:38.649808 65.83.241.167.domain > 192.168.1.3.4870:
  11416 1/0/0 (69)

12:25:43.497909 arp who-has 192.168.1.1 tell 192.168.1.3

12:25:43.498153 arp reply 192.168.1.1 is-at 0:6:25:9f:34:ac

12:25:43.498943 192.168.1.3.4870 > 65.83.241.167.domain:
  11417+ PTR? 1.1.168.192.in-addr.arpa. (42) (DF)
```

```
12:25:43.533126 65.83.241.167.domain > 192.168.1.3.4870:
  11417 NXDomain 0/1/0 (119)

12:25:44.578546 192.168.1.1.8783 > 192.168.1.255.snmptrap:
  Trap(35) E:3955.2.2.1 192.168.1.1 enterpriseSpecific[specific-
  trap(1)!=0] 43525500 [|snmp]
```

This might look a little confusing at first, but if you break it down it starts to make more sense. The first number is a timestamp, broken down into fractions of a second, because on a busy network there will be many packets per second on the wire. The next number is the source IP address of the packet followed by > (a greater than sign), and then the destination address. Finally, there may be some comments and other data. You can see several different kinds of traffic in this example, including DNS traffic (domain), ARP, and SNMP.

By default, Tcpdump runs until stopped by you pressing Control+C or another interrupt signal. When Tcpdump stops, it prints a summary of all the traffic it saw. The summary statistics include:

- Packets received by filter. This is the count of packets processed by the Tcpdump filter. It is not a count of all the TCP packets on the wire unless you ran Tcpdump without any filter criteria.
- Packets dropped by kernel. The number of packets that were dropped due to a lack of resources on your system. This feature may not be supported on all systems. Even when it is, it may not be accurate if there is a lot of saturation on the network or your sniffer machine is very slow.

TCP/IP Packet Headers

This section describes the contents of a TCP/IP packet header so you can understand what you see in the Tcpdump display. The layout of the TCP/IP packet is specified in RFC 793 for the TCP portion and RFC 791 for the IP portion. You can find the full text of these RFCs online at www.rfc-editor.org. Figure 6.1 is a graphical representation of TCP and IP headers. Both header types are at least 20 bytes long and are usually shown in 32-bit (4-byte) sections with the addresses, options, and other settings for the session.

Let's look at the IP portion first, since this is the lowest layer of the network model. The IP protocol header contains the delivery address for the packet and its sender. Since each address is 32 bits (4 octets of 8 bits each), the source and destination IP address takes up 8 bytes. The first part of the header contains various switches and options for the packet. The first line contains several switches that identify the IP version. Most networks uses IP version 4 (IPv4), but a newer 128-bit IP system called IP version 6 (IPv6) has been circulating for several years and has been gradually gaining acceptance. IPv6 is supposed to solve the IP address space problem by allowing up to 128 bits for the address portion.

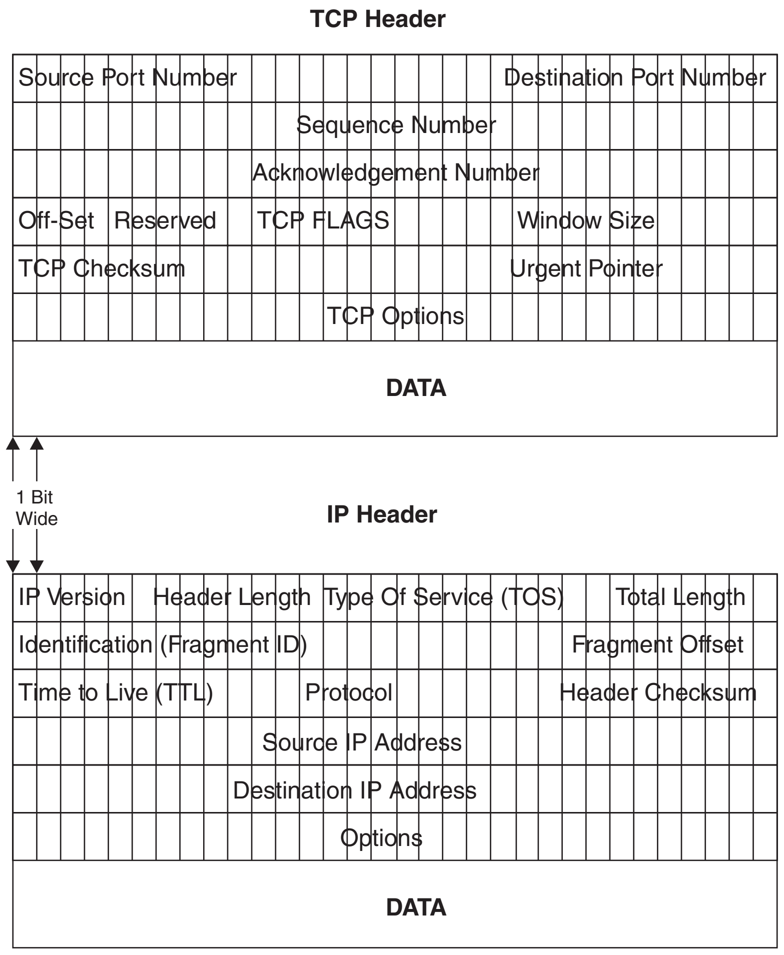


Figure 6.1 TCP/IP Header

This should create enough addresses to solve any foreseeable address space needs. IPv6 also resolves the security and verification issues with IPv4. But for now, you will mostly see IPv4 packets. Then there are the Header Length and the Type Of Service settings (TOS), which allow for differentiating in the priority of packets. The last part of this line is the total length of the header, which is normally the same from packet to packet (20 bytes), but can vary for newer protocols like IPv6.

The next two lines deal with identification of the packet and a checksum to make sure that it is valid. Finally, there are the source and destination IP addresses, and an options field that can be variable length or padded with zeros and any data.

The TCP header takes care of establishing a TCP session and higher-level functions. It is usually 20 bytes long and starts with a source port number of 16 bits and a destination port number of 16 bits. This is why the port numbers can only go up to 65,535—because the port number field in TCP/IP is a 16-bit binary number and 2^{16} power equals 65,536, or 0–65,565. (It is interesting how all these seemingly arbitrary numbers always have a basis in something.)

The port numbers, as mentioned earlier, identify which program the packets need to be directed to on the remote machine and identify the session on the local machine. The next line contains a sequence number. This is used to reassemble the packets in the right order at the other end, even if they arrive in a different order. This is one of the fault-tolerant aspects of TCP sessions. After that, there is an acknowledgment number, also 32 bits long, which allows for verification that it is coming from the right machine. The next line contains a 4-bit section called the data offset, which gives how many 32-bit lines or "words" are in this header (typically 4) and 6 bits that are reserved for future use. After that there is a 6-bit section called the TCP Flags; the last half of that line is used to confer the window size, which tells the recipient how many bits the sender is willing to accept. The Flags are pretty important, as this is where different TCP control bits are set that control how the packet is handled. Each type of TCP communication is designated by one bit, with one being on, or set, and zero being off. Table 6.1 lists the six fields of the TCP Flag section and describes their use. Note: Each "field" is one bit wide, simply a one or zero, on or off.

Table 6.1 TCP Flag Fields

TCP Flags	Full Names	Descriptions
URG	Urgency pointer	Indicates the TCP priority of the packets.
ACK	Acknowledgment	Designates this packet as an acknowledgment of receipt.
PSH	Push	Flushes queued data from buffers.
RST	Reset	Resets a TCP connection on completion or being aborted.
SYN	Synchronization	Synchronizes a connection.
FIN	Finished	Finishes a transmission.

Normally only one or two of these fields are on (the bits set to one), but as you saw in Chapter 4, there is nothing to stop you from sending a packet with all these bits flipped on (XMAS scan) or flipped off (NULL scan) to try to confuse a remote system.

Next are the TCP checksum and an urgent pointer. Then there is a line with any TCP options for the packet. These might include additional checksums, timestamps, or other optional information. This line is padded out to 32 bits with zeros if the options don't fill all the space. Finally the actual payload, the data of the packet, follows. This may seem like a lot of administrative overhead for sending one packet (approximately 48 bytes for every packet), but it does ensure a relatively stable connection on networks that are not always reliable end to end (like the Internet). And indeed, because of the TCP overhead, some protocols that are not connection-sensitive use UDP, which is a connectionless protocol that lowers the amount of overhead.

On a standard Tcpdump session with normal verbosity, you will see a timestamp followed by the TCP sequence number. Then it shows parts of the IP stack, including the source and destination with a > (greater than sign) between them, meaning this packet is going from here to there. At the end is the info field, which tells what the packet is doing. You can use the `-v` or `-vv` option to get more detail from Tcpdump about the header (see the next section).

Usually, you will want to run Tcpdump with some of the options or filters set to narrow down and focus the output. The general form of the Tcpdump statement is:

```
tcpdump options expressions
```

Replace *`options`* or *`expressions`* with one or more of the valid variables. Table 6.2 lists the Tcpdump options.

Table 6.2 Tcpdump Options

Options	Descriptions
-a	Attempts to convert addresses to names. This puts a higher load on the system and may cause packet loss.
-c *count*	Stops Tcpdump after *count* number of packets are processed.
-C *filesize*	Limits the output files to *filesize* number of bytes.
-d	Dumps the packet-matching code in a human-readable form and then stops.
-dd	Dumps the packet-matching code as a C program fragment.

(continues)

Table 6.2 Tcpdump Options (*continued*)

Options	Descriptions
-ddd	Dumps the packet-matching code as decimal numbers.
-e	Prints the link-level header on each dump line. This is the MAC address on an Ethernet network.
-E *algo:secret*	Uses Tcpdump's built-in ability to decrypt packets encrypted with IPsec ESP on the fly. Of course, you must have the shared secret to use this option. The *algo* options include des-cbc, 3des-cdc, blowfish-cbc, r3c-cbc, cast 128-cbc, and none. The default is des-cbc. The value of *secret* should be the ESP secret key in ASCII text form. For more information on IPsec, see Chapter 9.
-F *file*	Uses the filename *file* as input rather than taking input live from the wire. This is useful for analyzing events after the fact.
-i *interface*	Reads from *interface* when there are multiple network interfaces on the sniffer machine. By default, Tcpdump uses the lowest numbered valid interface. On Linux boxes, you can also use the parameter any to capture packets on all network interfaces.
-n	Doesn't convert addresses to names.
-N	Doesn't print the upper-level domain name element of host names. This is useful if you need to provide a sanitized version of the output and don't want to reveal whose network it is on.
-p	Doesn't put the interface into promiscuous mode. Only used when you are troubleshooting traffic to your sniffer box.
-q	Prints quick output. Less protocol information is printed so the lines are shorter.
-T *type*	Forces packets selected by the filter in the expression to be interpreted by *type*.
-t	Doesn't print a timestamp on each line.

Options	Descriptions
-tt	Prints an unformatted timestamp on each line.
-ttt	Prints the delta time between packets.
-tttt	Prints a timestamp in a default format preceded by the date on each line.
-v	Uses slightly more verbose output. Includes the time-to-live, identification, total length, and options fields of each packet.
-vv	Provides more verbose output. NFS and SMB packets are fully decoded.
-vvv	Provides even more verbose output. This may seriously slow down your sniffer.
-w *filename*	Writes the packets to the file *filename* rather than displaying them on the screen. This way, unattended sniffing can be saved and analyzed later. For example, if you had some strange things happening on your network, you could leave Tcpdump running overnight to capture any odd traffic. Just make sure you write a good filter, or you could have a very large file when you come back in the morning.
-x	Displays each packet (minus the link-level header) in hex.
-X	Displays packet contents in both hex and ASCII.

Tcpdump Expressions

The Tcpdump expressions select which packets from the datastream are displayed. This is where the work of Tcpdump is really done. Only items that match the expression are dumped; if no expression is given, then all packets will be displayed. A Tcpdump expression consists of one more directives, called **primitives**. These consist of an ID followed by a qualifier. Table 6.3 lists the three different kinds of qualifiers, and Table 6.4 lists the allowable primitive combinations.

There are also more complex expressions that can be constructed using Boolean arithmetic operators such as and, or, not, greater than, and less than. See the Tcpdump man page for examples and usage.

Table 6.3 Tcpdump Qualifiers

Qualifiers	Descriptions
type	Specifies what the ID name or number refers to. Possible types are host, net, and port. For example, host foo, net 128.3, or port 20.
dir	Specifies the direction of traffic from a particular ID. Possible directions are src; dst; src or dst; and src and dst (**src** stands for source address and **dst** stands for destination address).
proto	Lets you specify the protocol to filter out. Possible protos are ether, fddi, tr, ip, ipv6, arp, rarp, decnet, tcp, and udp. If no proto is specified, then all protocols consistent with the rest of the expression are allowed.You can use this to find out which machine is doing excessive arps or to filter out udp requests, which can be extensive on many networks since DNS requests use udp.

Table 6.4 Allowable Primitive Combinations

Combinations	Descriptions
dst host *host*	Shows only traffic addressed to *host*, which may be either an IP address or hostname.
src host *host*	Shows only traffic coming from *host.*
host *host*	Shows traffic either originating or destined for*host.*
ether dst *ehost*	Shows traffic destined for a specific Ethernet name, *ehost*, which can be either a name or a number (MAC address).
ether src *ehost*	Shows traffic originating from *ehost.*
ether host *ehost*	Shows traffic either originating from or destined for *ehost.*
gateway *host*	Shows any traffic that used *host* as a gateway. In other words, it was forwarded from *host*. This happens when the IP source or destination address doesn't match the Ethernet address of *host*. You can use this when you want to track all traffic going through your Internet gateway or some specific router.

Combinations	Descriptions
dst net *net*	Filters traffic that is destined for a specific network, *net*, specified in 0.0.0.0 notation. Similar to ether dst *ehost*, except it can be much broader than a single host.
src net *net*	Filters for a source network, *net*.
net *net*	Same as the previous two statements except it allows traffic either from or to the *net* network.
net *net* mask *netmask*	Matches traffic from or to *net* network with a netmask of *netmask*. Used for specifying the exact size of a network in increments smaller than a class C. You can also use src or dst with this statement to specify the direction of the traffic.
net *net/len*	Matches traffic with network addresses of *net* and *len* bits in the netmask. Similar to the last statement.
dst port *port*	Filters TCP and UDP traffic with a destination port value of *port*. You can also specify either TCP or UDP here to only catch traffic of that type. Otherwise, both types are shown.
src port *port*	Same as the last statement, except this captures traffic with a source port of *port*.
less *length*	Shows packets with a length of less than *length*. This can also be stated as len <= *length*.
greater *length*	Same as the statement above except it captures only traffic of length greater than the *length* value.
ip proto *protocol*	Captures traffic that is of a specific protocol type. Allowable names are icmp, icmpv6, igmp, igrp, pim, ah, esp, vrrp, udp, and tcp. The names tcp, udp, and icmp must be put between backslashes in order to keep them from being read as keywords. For example: ip proto protocol /tcp/.
ip6 proto *protocol*	Similar to the above statement but for IPv6 packets and types.
ip6 protochain *protocol*	Finds IPv6 packets that have a protocol header of *protocol*.

(continues)

Table 6.4 Allowable Primitive Combinations (*continued*)

Combinations	Descriptions
ip protochain *protocol*	Same as above but for IPv4 packets.
ip broadcast	Identifies only traffic that is broadcast, that is, has all zeros or all ones in the destination fields.
ether multicast	Registers true (displays) if the packet is an Ethernet multicast packet.
ip multicast	Registers true if the packet is an IP multicast packet.
ip6 multicast	Registers true if the packet is an IPv6 multicast packet.
ether proto *protocol*	Displays any traffic that is of Ethernet type *procotol.* Allowable protocol names are ip, ipv6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl, moprc, iso, stp, ipx, or netbeui. These names are also identifiers, so they must be escaped by using backslashes.
decnet src *host*	Captures DECnet traffic with a source address of *host.*
decnet dst *host*	Same as the above statement but filters on destination address of *host.*
decnet *host*	Filters for DECnet addresses with either the source or destination equal to *host.*
ip	A shorter version of the ether proto statement described earlier. Traps traffic matching the Ethernet protocol of IP.
ip6	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of IPv6.
arp	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of arp.
rarp	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of rarp.
atalk	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of AppleTalk.

Combinations	Descriptions
aarp	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of aarp.
decnet	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of DECnet.
iso	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of iso.
stp	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of stp.
ipx	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of ipx.
netbeui	Shorter version of the ether proto statement for trapping traffic matching the Ethernet protocol of netbeui.
vlan *vlan_id*	Captures packets based on the 802.1Q VLAN standard. It can be used by itself or by specifying *vlan_id*.
tcp	An abbreviated form of the statement ip proto tcp.
udp	An abbreviated form of the statement ip proto udp.
icmp	An abbreviated form of the statement ip proto icmp.
iso proto *protocol*	Captures OSI packets with a protocol type of*procotol*. Allowable OSI protocol types are clnp, esis, and isis.
clnp	An abbreviated form of the above statement using clnp for*protocol*.
esis	An abbreviated form of the iso proto *protocol* statement using esis for *protocol*.
isis	An abbreviated form of the iso proto *protocol* statement using isis for *protocol*.

Tcpdump Examples

The following are several practical examples of ways to use Tcpdump.

View All Traffic to and from a Particular Host If you want to monitor only traffic to and from a specific host, you can filter everything else out with the simple "host" expression. For example, to monitor a host with the IP address 192.168.1.1, the statement would look like this:

```
tcpdump –n host 192.168.1.1
```

Watch Only Traffic Coming in or out on a Certain Port If you want to track usage of a certain application, you can use Tcpdump to trap all traffic for a particular TCP/UDP port. If the application you are trying to monitor is Telnet (port 23), you could do this with the following Tcpdump expression:

```
tcpdump –n port 23
```

View All Traffic to and from a Particular Host but Eliminate Some Kinds of Traffic Say you want to monitor a single host as in the first example but want to filter out SSH traffic (if you were ssh'd into that host, unfiltered Tcpdump output would show your own connection traffic). You can do this by adding the `port` expression with a Boolean operator "not" statement. Here is the command:

```
tcpdump –n host 192.168.1.1 and not port 22
```

Find a Rogue Workstation If you are having network problems and suspect a rogue computer is swamping your network, you can use Tcpdump to quickly track down the culprit. Whether it's a bad network card or a trojanized PC causing a denial of service attack, Tcpdump will help shed some light on your problem. First try just running it wide open to see what is generating the most traffic. Use the `-a` and `-e` options to generate names and MAC addresses.

```
tcpdump -ae
```

Notice that you can concatenate the two letters with one dash. If this causes the output to scroll off the screen too fast, use the `-c 1000` option to only count 1,000 packets and then stop.

Monitor a Specific Workstation If you want to log the traffic from a specific workstation to analyze later, you can do this easily with Tcpdump (just make sure that you have the legal right to do so). Use the Tcpdump statement from the first example with a `–w` switch to write to a file. If you use DHCP on your network, you may be better off using SMB (Windows) names. For example:

```
tcpdump –w logfile host 192.168.1.1
```

where *`logfile`* is the file it will log to. You may also want to use the `-c` or `-C` options to limit your output file size.

Look for Suspicious Network Traffic If you are worried about what is happening on your network after hours, you can leave Tcpdump running to flag traffic you might deem questionable. You could run it with the `gateway` *`192.168.0.1`* flag set, where you replace the IP address with that of your own Internet gateway. Assuming your home network was in the IP Range of 192.168.0.0 through 192.168.0.254, this would flag any traffic coming or going from your Internet gateway. If you have an internal mail server and don't want to log that traffic since that would be valid traffic, you could add the statement:

```
and host != 192.168.0.2
```

where the IP address is the address of your mail server. The exclamation point also acts as the Boolean "not" statement. This would flag any incoming traffic not bound for your mail server. The expression would look like this:

```
tcpdump -w logfile gateway 192.168.0.1 and
host!=192.168.1.2
```

If you are looking for users using a particular application, such as a streaming video or an audio program, you can further specify that as long as you know its port number. If you know it uses the TCP port 1000, you can use the proto primitive to trap traffic using that protocol. For example:

```
tcpdump -w logfile gateway 192.168.0.1 and
host!=192.168.1.2
dst port 1000
```

For more complicated intrusion detection scenarios, you will be better off using one of the intrusion detection systems described in Chapter 7, but for a quick and dirty analysis, Tcpdump can be a very handy tool.

WinDump: An Ethernet Traffic Analyzer for Windows

WinDump

Author/primary contact:	Loris Degioanni
Web site:	windump.polito.it/install/default.htm
Platforms	Windows 95, 98, ME, NT4, 2000, XP
License:	BSD
Version reviewed:	3.8 alpha
WinPcap mailing list:	

www.mail-archive.com/winpcap-users@winpcap.polito.it/

Finally, there is a Tcpdump program for Windows. In fact, this is the actual UNIX Tcpdump ported over to the Windows platform, so all the functions and expressions work exactly the same.

Installing WinDump

Loris Degioanni was kind enough to do the porting work and made it a breeze to install—even easier than its UNIX counterpart.

1. Just like the UNIX Tcpdump, you first need to have the packet capture libraries installed before you can run WinDump. There is a special version for Windows called WinPcap. This is included on the CD-ROM in the Misc Folder. The latest version is also available at the program's Web site.
2. Install the WinPcap libraries by clicking on the file.
3. Download the WinDump executable and place it in the directory you want to run it from.

   No additional installation is necessary.

Using WinDump

Using WinDump is exactly the same as using Tcpdump from the command line. Just go to a command prompt in Windows and issue the command from the directory that the WinDump executable is in. All the commands and expressions work the same, but Table 6.5 lists a few commands specific to the Windows version.

The source code is also available on the Web site for those wishing to contribute or to make modifications of their own. A word of warning, though: this kind of Windows coding is only for the hard core and those truly knowledgeable about network protocols.

This is all you need to get going in either Windows or UNIX. If you want more than just a command line interface though, the next tool described offers a graphical interface for your sniffing activities.

Table 6.5 WinDump-Specific Commands

Commands	Descriptions
-B	Sets the driver buffer size in kilobytes for your capture session. If you are experiencing high rates of packet loss, you can try increasing this value a little. The default is 1MB (`-B 1000`)
-D	Prints a list of available network interfaces on your system. It shows the interface name, number, and description, if any. You can use these parameters to specify an interface to capture from using the Tcpdump `-i` switch.

Ethereal: A Network Protocol Analyzer for UNIX and Windows

Ethereal

Author/primary contact:	Gerald Combs
Web site:	www.ethereal.com
Platforms:	Most UNIX, Windows 95, 98, ME, NT4, 2000, XP
License:	GPL
Version reviewed:	0.10.2

Mailing lists:

Ethereal-announce
General announcement list. Doesn't accept posts.
Subscribe at www.ethereal.com/mailman/listinfo/ethereal-announce.

Ethereal-users
General questions about using Ethereal. Post your newbie questions here.
Subscribe at www.ethereal.com/mailman/listinfo/ethereal-users.

Ethereal-dev
Development discussions.
Subscribe at www.ethereal.com/mailman/listinfo/ethereal-dev.

Ethereal-doc
For people writing Ethereal documentation or who want to become involved in writing documentation. Subscribe at www.ethereal.com/mailman/listinfo/ethereal-doc.

Ethereal-cvs
For monitoring changes to the Ethereal CVS tree, which maintains the very latest version of the code for developers. It doesn't accept posts, and any questions should be directed to either Ethereal-users or -dev depending on the question. Subscribe at www.ethereal.com/mailman/listinfo/ethereal-cvs.

Ethereal offers all the benefits of a command line tool like Tcpdump with a number of advantages. It has a user-friendly graphical interface, so you don't have to deal with learning all the command line parameters. It also offers many more analytical and statistical options. Some of the other benefits of Ethereal are:

- Cleaner output format. The output is much easier to read and understand than the raw packet captures of Tcpdump.
- Many more protocol formats are supported. Ethereal can interpret over 300 different network protocols, which covers just about every network type ever invented.

- More physical network formats are supported. This includes newer protocols such as IP over ATM and FDDI.
- Captured network data can be interactively browsed and sorted.
- Output can be saved as plain text or in PostScript format.
- A rich display filter mode. This includes the ability to highlight certain packets in color. There is a filter creation GUI to walk you through the process of creating filters easily.
- The ability to follow a TCP stream and view the content in ASCII. This can be invaluable when you need to read inter-server messages to track down e-mail or Web problems. You can follow the conversation between communicating nodes in order using this feature.
- The ability to work with a number of capture programs and libraries. Ethereal also works with dedicated hardware beyond libpcap. Some of the programs supported include Network Associate's Sniffer and Sniffer Pro; Novell's LANalyser; some Cisco, Lucent, and Toshiba devices; and some wireless sniffing gear such as Net-Stumbler and Kismet Wireless. Ethereal now works as a plug-in module for many of these programs and devices.
- The ability to save sessions in multiple formats. This is useful if you want to do additional analysis with different tools, including libcap (the default), Sun Snoop, Microsoft Network Monitor, and Network Associates' Sniffer.
- A command-line terminal mode. This is for those not graphically inclined, although a huge part of Ethereal's usefulness comes from its GUI tools.

Ethereal is so useful as a networking tool that it has been rated as number two among the most popular network security tools available by the security Web site Insecure.org. Ethereal has many uses beyond just security; in fact, you can also use it as a general network analysis tool.

Installing Ethereal for Linux

1. You need two prerequisites before loading Ethereal: the libpcap libraries and the GTK development libraries. If you have loaded the port scanners or vulnerability scanners from earlier chapters, you should be all set. If not, you will need to download the GTK libraries or install them off of your OS installation disks. You can get libpcap on the CD-ROM or at www.tcpdump.org. GTK is available at www.gtk.org.
2. Now, you have to decide whether to use an RPM or compile from the source code. There are many RPM packages for different versions of Linux. If one exists for your distribution, you can use that and skip the compile process. If there isn't an RPM version for your operating system, you need to compile it.

3. To compile Ethereal, first download and unpack the latest distribution. The default installation should work fine for most uses. Look at the INSTALL file if you want to set additional compile-time parameters.

4. Change to the install directory and type the usual:

```
./configure
make
make install
```

You can now run Ethereal by typing `./ethereal` at the command prompt or by clicking on the executable from X-Windows. You need to be the root user to run Ethereal in the X-windows environment. To run Ethereal in command-line mode, you can type `./tethereal`.

Installing Ethereal for Windows

1. You need to have the WinPcap libraries installed before running Ethereal. If you have already installed the port or vulnerability scanners from the previous chapters on your Windows system, then you already have these loaded and you can go to Step 2. Make sure your version of WinPcap is at least 2.3 or later. If you are running a machine with a multiprocessor or one of the newer Pentium processors with hyper-threading technology, you need to have WinPcap 3.0 or higher, and your results may be unpredictable as Ethereal doesn't work well with multiple processors.

2. The GTK tools for the graphical interface are included in the Ethereal installation package. Go to the Ethereal Web site and download a self-extracting install file. (I recommend you install the binary rather than messing with compilation on a Windows machine. This is much easier and doesn't require a Windows compiler.)

3. After you download the file, double-click on it. The installation program walks you through the install process. When it is done, it will put an icon on your desktop and you are ready to start using Ethereal.

Using Ethereal

Whether you are using the Windows or Linux version, almost all of the operations are the same and the interfaces look the same. When you bring up Ethereal, you will see a screen with three sections in it. These windows display the capture data and other information about your session. Figure 6.2 shows an example of this main window with a session in progress.

The top third of the screen is where the packet stream is displayed in order of receipt, although you can sort this in just about any way by clicking on the headings. Table 6.6 lists the items displayed for each packet or frame.

The next section of the screen goes into more detail on each packet that is highlighted. It is arranged in an order that basically conforms to the OSI model, so the first item listed

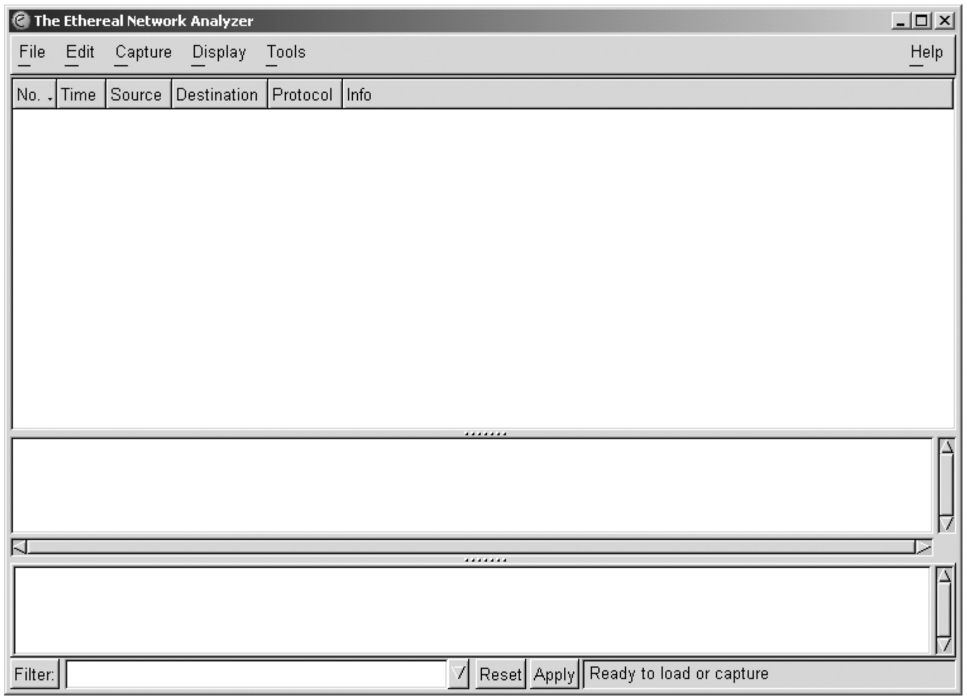


Figure 6.2 Ethereal Main Screen

Table 6.6 Packet Stream Data

Items	**Descriptions**
Packet number	Assigned by Ethereal.
Time	The time the packet was received, set from the elapsed time from the start of the capture session. Alternately, this can be configured to show the clock time, the clock time and date, or even the time between packets (this is helpful for network performance analysis).
Source address	Where the packet came from. This is an IP address on IP networks.
Destination address	Where the packet is going to, also usually an IP address.
Protocol	The level 4 protocol that the packet is using.
Info	Some summary information about the packet, usually a type field.

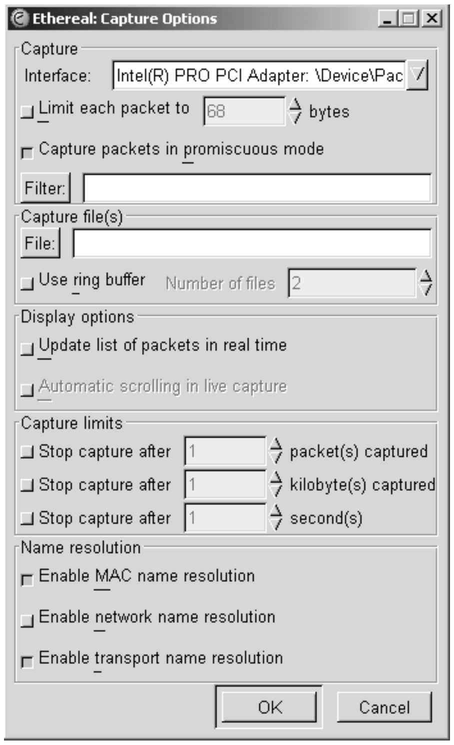
is detail on the data link layer, and so on. The little pluses can be expanded to show even more information on each level. It is amazing how much detail you can see on each packet. Ethereal is like an electron microscope for network packets!

The final section contains the actual packet contents, in both hexadecimal and translated into ASCII where possible. Binary files will still look like garbage, as will encrypted traffic, but anything in clear text will appear. This highlights the power (and danger) of having a sniffer on your network.

Starting a Capture Session

There are a lot of options and filters you can set. Begin by running a wide open capture session. Choose Start from the Capture menu, and the Capture Options window displays (see Figure 6.3).

Table 6.7 describes the options you can set before starting your session.


Ethereal: Capture Options

Capture
Interface: Intel(R) PRO PCI Adapter: \Device\Pac
☐ Limit each packet to 68 bytes
☑ Capture packets in promiscuous mode
Filter:

Capture file(s)
File:
☐ Use ring buffer Number of files 2

Display options
☐ Update list of packets in real time
☐ Automatic scrolling in live capture

Capture limits
☐ Stop capture after 1 packet(s) captured
☐ Stop capture after 1 kilobyte(s) captured
☐ Stop capture after 1 second(s)

Name resolution
☑ Enable MAC name resolution
☐ Enable network name resolution
☑ Enable transport name resolution

OK Cancel

Figure 6.3 Ethereal Capture Options

Howlett_CH06.fm Page 188 Thursday, June 24, 2004 11:47 AM

Table 6.7 Ethereal Capture Options

Options	Describes
Interface	Picks the interface to capture from the pull-down menu. Ethereal automatically senses all the available interfaces and lists them. You can also choose to capture from all interfaces at once, just like Tcpdump.
Limit each packet to *x* bytes	Sets a maximum size for the packets captured. You can use this if you fear some of the packets may be very large and you don't want to overload your machine.
Capture packets in promiscuous mode	This is on by default. Turn this off if you want to capture traffic only to your sniffer machine.
Filter	Click the Filter button to create a filter using Tcpdump-style expressions. It will ask you to name the filter (which you can then use in future sessions) and enter the expression.
Capture file(s)	Click the File button if you want to read from a file rather than capture live data.
Display options	These are disabled by default, but enable them if you want to watch the packets scroll by in real time. If you are capturing on a busy network or your machine is slow, this is not recommended because it will cause the session to bog down and possibly drop packets. However, it is very useful if you want to "eyeball" the traffic to get a general idea of the nature of flow on the network as it goes by.
Capture limits	You have several more options here on when to end your session. Besides manually stopping it, you can have Ethereal stop after *x* number of packets or kilobytes of data have been captured, or after *x* number of seconds have elapsed.
Name resolution	You can specify whether you want Ethereal to resolve names at various levels of the network model. You can selectively resolve MAC address names, network names (SMB or hostnames), and/or transport layer names. Enabling all of these, especially DNS, can slow down your capture significantly.

Once you have set your options, click OK and your session will start. A window will appear that tracks the session statistics in real time (see Figure 6.4). If you set your session to show packets in real time, you will see them as they come across the wire in the window (see Figure 6.2).

You can stop your session at any time by clicking Stop in the statistic window or choosing Stop from the Capture menu. If you set a limit in the options, it will automatically stop when it reaches it. You can now analyze and manipulate your session results.

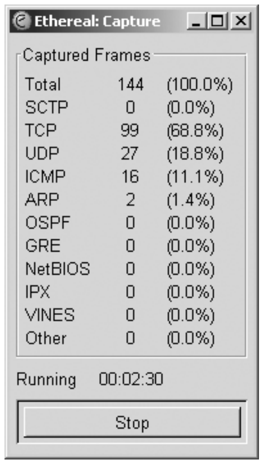
By clicking on the headings at the top of the window, you can resort the results by that heading, so you can sort the output by source address, destination, protocol, or the info fields. This helps to organize things if you are looking for a specific kind of traffic, for example, all the DNS queries or all the mail-related traffic. Of course, you could also write a filter to capture only this kind of traffic in the first place.

Display Options

Table 6.8 lists the commands on the Display menu that you can use to affect how the packets are displayed on the screen.

Ethereal Tools

There are several built-in analytical tools included with Ethereal. It is also built with a plug-in architecture so that other programs can interact with Ethereal or you can write your own. You can access these options under the Tools menu (see Table 6.9).



```
Captured Frames
Total     144  (100.0%)
SCTP      0    (0.0%)
TCP       99   (68.8%)
UDP       27   (18.8%)
ICMP      16   (11.1%)
ARP       2    (1.4%)
OSPF      0    (0.0%)
GRE       0    (0.0%)
NetBIOS   0    (0.0%)
IPX       0    (0.0%)
VINES     0    (0.0%)
Other     0    (0.0%)

Running   00:02:30
```

Figure 6.4 Ethereal Session Statistics Window

Table 6.8 Ethereal Display Menu Options

Menu Options	Descriptions
Options submenu	This where you can set some global settings, such as how the time field is calculated. You can also set automatic scrolling of traffic and name resolution to on since they are turned off by default.
Colorize display	You can select certain kinds of packet to shade different colors. This makes the display easier to read and pick out the items you are looking for.
Collapse/expand all	Shows either full detail on every item or just the top level.

Table 6.9 Ethereal Tools Menu Options

Options	Descriptions
Summary	Shows a listing of the top-level data on your captures session, such as time elapsed, packet count, average packet size, total bytes captured, and average Mps on the wire during the capture.
Protocol hierarchy statistics	Gives a statistical view of the traffic on your network. It shows what percentage of the capture session each type of packet makes up. You can collapse or expand the view to see major levels or minor protocols within a level.
Statistics	Contains a number of reports that are specific to certain kinds of protocols. Refer to the Ethereal documentation for more details on these tests.
Plugins	Shows the protocol analyzer plug-ins that you have loaded. These are decoders for newer protocols that can be added to Ethereal without a major version upgrade. And because it's a plug-in architecture, you can write your own.

Saving Your Ethereal Output

Once you have finished capturing and analyzing your Ethereal data, you may want to save it, either for analysis with additional tools or for presentation to other parties. Using the Save As option from the File menu, you can choose from a number of formats, including libpcap (the default), Sun Snoop, LANalyser, Sniffer, Microsoft Network Monitor, and Visual Networks traffic capture.

Ethereal Applications

Now that you understand the basics of Ethereal, here are some practical applications you can use it for.

Network Optimization By running a wide-open network capture and then using the statistical reports, you can see how saturated your LAN is and what kinds of packets are making up most of the traffic. By looking at this, you may decide that it is time to move to a 100Mps switched network, or to segregate two departments into routed LANs versus one big network. You can also tell if you need to install a WINS server (too many SMB name requests being broadcast across the LAN) or if a particular server should be moved to a DMZ or a separate router port to take that traffic off the network.

Application Server Troubleshooting Do you have a mail server that doesn't seem to be connecting? Having DNS problems? These application-level problems can be fiendishly difficult to troubleshoot. But if you have Ethereal, you can tap into the network and watch the inter-server communications. You can see the actual server messages for protocols like SMTP or HTTP and figure out where the problem is happening by watching the TCP stream.