NAME

snoop – capture and inspect network packets

SYNOPSIS

snoop [**–aPDSvVNC**] [**–d** *device*] [**–s** *snaplen*] [**–c** *maxcount*] [**–i** *filename*]
[**–o** *filename*] [**–n** *filename*] [**–t** [**r** | **a** | **d**]] [**–p** *first* [, *last*]]
[**–x** *offset* [, *length*]] [*expression*]

AVAILABILITY

SUNWcsu

DESCRIPTION

snoop captures packets from the network and displays their contents. **snoop** uses both the network packet filter and streams buffer modules to provide efficient capture of packets from the network. Captured packets can be displayed as they are received, or saved to a file for later inspection.

snoop can display packets in a single-line summary form or in verbose multi-line forms. In summary form, only the data pertaining to the highest level protocol is displayed. For example, an NFS packet will have only NFS information displayed. The underlying RPC, UDP, IP, and ethernet frame information is suppressed but can be displayed if either of the verbose options are chosen.

OPTIONS

–a Listen to packets on **/dev/audio** (warning: can be noisy).

–P Capture packets in non-promiscuous mode. Only broadcast, multicast, or packets addressed to the host machine will be seen.

–d *device* Receive packets from the network using the interface specified by *device*. Usually **le0**
or **ie0**. The program **netstat**(1M), when invoked with the **–i** flag, lists all the interfaces that a machine has. Normally, **snoop** will automatically choose the first non-loopback interface it finds.

–s *snaplen* Truncate each packet after *snaplen* bytes. Usually the whole packet is captured. This option is useful if only certain packet header information is required. The packet truncation is done within the kernel giving better utilization of the streams packet buffer. This means less chance of dropped packets due to buffer overflow during periods of high traffic. It also saves disk space when capturing large traces to a capture file. To capture only IP headers (no options) use a *snaplen* of 34. For UDP use 42, and for TCP use 54. You can capture RPC headers with a *snaplen* of 80 bytes. NFS headers can be captured in 120 bytes.

–c *maxcount* Quit after capturing *maxcount* packets. Otherwise keep capturing until there is no disk left or until interrupted with CTRL-C.

–i *filename* Display packets previously captured in *filename*. Without this option, **snoop** reads packets from the network interface. If a *filename***.names** file is present, it is automatically loaded into **snoop**'s IP address-to-name mapping table (See **–N** flag below).

–o *filename* Save captured packets in *filename* as they are captured. During packet capture, a count of the number of packets saved in the file is displayed. If you wish just to count packets without saving to a file, name the file **/dev/null**.

–n *filename* Use *filename* as an IP address-to-name mapping table. This file must have the same format as the **/etc/hosts** file (IP address followed by the hostname).

–D Display number of packets dropped during capture on the summary line.

–S Display size of the entire ethernet frame in bytes on the summary line.

–t [r | a | d]
Time-stamp presentation. Time-stamps are accurate to within 4 microseconds. The default is for times to be presented in **d** (delta) format (the time since receiving the previous packet).
Option **a** (absolute) gives wall-clock time.
Option **r** (relative) gives time relative to the first packet displayed. This can be used with the **–p** option to display time relative to any selected packet.

–v Verbose mode. Print packet headers in lots of detail. This display consumes many lines per packet and should be used only on selected packets.

–V Verbose summary mode. This is halfway between summary mode and verbose mode in degree of verbosity. Instead of displaying just the summary line for the highest level protocol in a packet, it displays a summary line for each protocol layer in the packet. For instance, for an NFS packet it will display a line each for the ETHER, IP, UDP, RPC and NFS layers. Verbose summary mode output may be easily piped through **grep** to extract packets of interest. For example to view only RPC summary lines:

example# snoop –i rpc.cap –V | grep RPC

–p *first* [, *last*]
Select one or more packets to be displayed from a capture file. The *first* packet in the file is packet #1.

–x *offset* [, *length*]
Display packet data in hexadecimal and ASCII format. The *offset* and *length* values select a portion of the packet to be displayed. To display the whole packet, use an *offset* of 0. If a *length* value is not provided, the rest of the packet is displayed.

–N Create an IP address-to-name file from a capture file. This must be set together with the **–i** option that names a capture file. The address-to-name file has the same name as the capture file with **.names** appended. This file records the IP address to hostname mapping at the capture site and increases the portability of the capture file. Generate a **.names** file if the capture file is to be analyzed elsewhere. Packets are not displayed when this flag is used.

–C List the code generated from the filter expression for either the kernel packet filter, or **snoop**'s own filter.

expression Select packets either from the network or from a capture file. Only packets for which the expression is true will be selected. If no expression is provided it is assumed to be true.

Given a filter expression, **snoop** generates code for either the kernel packet filter or for its own internal filter. If capturing packets with the network interface, code for the kernel packet filter is generated. This filter is implemented as a streams module, upstream of the buffer module. The buffer module accumulates packets until it becomes full and passes the packets on to **snoop**. The kernel packet filter is very efficient, since it rejects unwanted packets in the kernel before they reach the packet buffer or **snoop**. The kernel packet filter has some limitations in its implementation — it is possible to construct filter expressions that it cannot handle. In this event, **snoop** generates code for its own filter. The –**C** flag can be used to view generated code for either the kernel's or **snoop**'s own packet filter. If packets are read from a capture file using the –**i** option, only **snoop**'s packet filter is used.

A filter *expression* consists of a series of one or more boolean primitives that may be combined with boolean operators (AND , OR , and NOT). Normal precedence rules for boolean operators apply. Order of evaluation of these operators may be controlled with parentheses. Since parentheses and other filter expression characters are known to the shell, it is often necessary to enclose the the filter expression in quotes. The primitives are:

host *hostname*
True if the source or destination address is that of *hostname*. The keyword **host** may be omitted if the name does not conflict with the name of another expression primitive e.g. "**pinky**" selects packets transmitted to or received from the host **pinky** whereas "**pinky and dinky**" selects packets exchanged between hosts **pinky** AND **dinky**. Normally the IP address is used. With the **ether** qualifier the ethernet address is used, for instance, "**ether pinky**".

ipaddr or *etheraddr*
Literal addresses, both IP dotted and ethernet colon are recognized. For example, "**129.144.40.13**" matches all packets with that IP address as source or destination, and similarly, "**8:0:20:f:b1:51**" matches all packets with the ethernet address as source or destination. An ethernet address beginning with a letter is interpreted as a hostname. To avoid this, prepend a zero

when specifying the address. For example, if the ethernet address is "aa:0:45:23:52:44", then specify it by add a leading zero to make it "0aa:0:45:23:52:44".

from or **src**
A qualifier that modifies the following **host**, **net**, *ipaddr*, *etheraddr*, **port** or **rpc** primitive to match just the source address, port, or RPC reply.

to or **dst**
A qualifier that modifies the following **host**, **net**, *ipaddr*, *etheraddr*, **port** or **rpc** primitive to match just the destination address, port, or RPC call.

ether A qualifier that modifies the following **host** primitive to resolve a name to an ethernet address. Normally, IP address matching is performed.

ethertype *number*
True if the ethernet type field has value *number*. Equivalent to "**ether[12:2] =** *number*".

ip, **arp**, **rarp**
True if the packet is of the appropriate ethertype.

broadcast
True if the packet is a broadcast packet. Equivalent to "**ether[2:4] = 0xffffffff**".

multicast
True if the packet is a multicast packet. Equivalent to "**ether[0] & 1 = 1**".

apple True if the packet is an Apple Ethertalk packet. Equivalent to "**ethertype 0x809b or ethertype 0x803f**".

decnet True if the packet is a DECNET packet.

greater *length*
True if the packet is longer than *length*.

less *length*
True if the packet is shorter than *length*.

udp, **tcp**, **icmp**
True if the IP protocol is of the appropriate type.

net *net* True if either the IP source or destination address has a network number of *net*. The **from** or **to** qualifier may be used to select packets for which the network number occurs only in the source or destination address.

port *port*
True if either the source or destination port is *port*. The *port* may be either a port number or name from **/etc/services**. The **tcp** or

udp primitives may be used to select TCP or UDP ports only. The **from** or **to** qualifier may be used to select packets for which the *port* occurs only as the source or destination.

rpc *prog* [, *vers* [, *proc*]]
: True if the packet is an RPC call or reply packet for the protocol identified by *prog*. The *prog* may be either the name of an RPC protocol from **/etc/rpc** or a program number. The *vers* and *proc* may be used to further qualify the program *version* and *procedure* number, for example, "**rpc nfs,2,0**" selects all calls and replies for the NFS null procedure. The **to** or **from** qualifier may be used to select either call or reply packets only.

gateway *host*
: True if the packet used *host* as a gateway, that is, the ethernet source or destination address was for *host* but not the IP address.
Equivalent to "**ether host** *host* and not host *host*".

nofrag
: True if the packet is unfragmented or is the first in a series of IP fragments.
Equivalent to "**ip[6:2] & 0x1fff = 0**".

expr relop expr
: True if the relation holds, where *relop* is one of **>**, **<**, **>=**, **<=**, **=**, **!=**, and *expr* is an arithmetic expression composed of numbers, packet field selectors, the **length** primitive, and arithmetic operators **+**, −, ∗, **&**, **|**, **^**, and%. The arithmetic operators within *expr* are evaluated before the relational operator and normal precedence rules apply between the arithmetic operators, such as multiplication before addition. Parentheses may be used to control the order of evaluation. To use the value of a field in the packet use the following syntax:

    base[*expr* [**:** *size*]]

where *expr* evaluates the value of an offset into the packet from a *base* offset which may be **ether**, **ip**, **udp**, **tcp**, or **icmp**. The *size* value specifies the size of the field. If not given, 1 is assumed. Other legal values are 2 and 4.

Examples:

"**ether[0] & 1 = 1**" is equivalent to **multicast**.

"**ether[2:4] = 0xffffffff** " is equivalent to **broadcast**.

"**ip[ip[0] & 0xf ∗ 4 : 2] = 2049**" is equivalent to "**udp[0:2] = 2049**".

"**ip[0] & 0xf > 5**" selects IP packets with options.

"**ip[6:2] & 0x1fff = 0**" eliminates IP fragments.

"**udp and ip[6:2]&0x1fff = 0 and udp[6:2] != 0**" finds all packets with UDP checksums.

The **length** primitive may be used to obtain the length of the packet. For instance "**length > 60**" is equivalent to "**greater 60**", and "**ether[length – 1]**" obtains the value of the last byte in a packet.

and Perform a logical AND operation between two boolean values. The AND operation is implied by the juxtaposition of two boolean expressions, for example "**dinky pinky**" is the same as "**dinky AND pinky**".

or or **,** Perform a logical OR operation between two boolean values. A comma may be used instead, for example, "**dinky,pinky**" is the same as "**dinky OR pinky**".

not or **!** Perform a logical NOT operation on the following boolean value. This operator is evaluated before AND or OR .

EXAMPLES

Capture all packets and display them as they are received:

example#snoop

Capture packets with host **funky** as either the source or destination and display them as they are received:

example#snoop**funky**

Capture packets between **funky** and **pinky** and save them to a file. Then inspect the packets using times (in seconds) relative to the first captured packet:

example# snoop –o cap funky pinky

example$ snoop –i cap –t r | more

Look at selected packets in another capture file:

example$ snoop –i pkts –p99,108

```
 99  0.0027  boutique -> sunroof     NFS C GETATTR FH=8E6C
100  0.0046  sunroof -> boutique     NFS R GETATTR OK
101  0.0080  boutique -> sunroof     NFS C RENAME FH=8E6C MTra00192 to .nfs08
102  0.0102  marmot -> viper         NFS C LOOKUP FH=561E screen.r.13.i386
103  0.0072  viper -> marmot         NFS R LOOKUP No such file or directory
104  0.0085  bugbomb -> sunroof      RLOGIN C PORT=1023 h
105  0.0005  kandinsky -> sparky     RSTAT C Get Statistics
106  0.0004  beeblebrox -> sunroof   NFS C GETATTR FH=0307
107  0.0021  sparky -> kandinsky     RSTAT R
108  0.0073  office -> jeremiah      NFS C READ FH=2584 at 40960 for 8192
```

Packet 101 Looks interesting. Take a look in more detail:

example$ snoop –i pkts –v –p101

```
ETHER:  ----- Ether Header -----
ETHER:
ETHER:  Packet 101 arrived at 16:09:53.59
ETHER:  Packet size = 210 bytes
ETHER:  Destination = 8:0:20:1:3d:94, Sun
ETHER:  Source      = 8:0:69:1:5f:e,  Silicon Graphics
ETHER:  Ethertype = 0800 (IP)
ETHER:
IP:   ----- IP Header -----
IP:
IP:   Version = 4, header length = 20 bytes
IP:   Type of service = 00
IP:         ..0. .... = routine
IP:         ...0 .... = normal delay
IP:         .... 0... = normal throughput
IP:         .... .0.. = normal reliability
IP:   Total length = 196 bytes
IP:   Identification 19846
IP:   Flags = 0X
IP:   .0.. .... = may fragment
IP:   ..0. .... = more fragments
IP:   Fragment offset = 0 bytes
IP:   Time to live = 255 seconds/hops
IP:   Protocol = 17 (UDP)
IP:   Header checksum = 18DC
IP:   Source address = 129.144.40.222, boutique
IP:   Destination address = 129.144.40.200, sunroof
IP:
UDP:  ----- UDP Header -----
UDP:
UDP:  Source port = 1023
UDP:  Destination port = 2049 (Sun RPC)
UDP:  Length = 176
UDP:  Checksum = 0
UDP:
RPC:  ----- SUN RPC Header -----
RPC:
RPC:  Transaction id = 665905
RPC:  Type = 0 (Call)
RPC:  RPC version = 2
RPC:  Program = 100003 (NFS), version = 2, procedure = 1
RPC:  Credentials: Flavor = 1 (Unix), len = 32 bytes
RPC:     Time = 06-Mar-90 07:26:58
RPC:     Hostname = boutique
RPC:     Uid = 0, Gid = 1
RPC:     Groups = 1
RPC:  Verifier   : Flavor = 0 (None), len = 0 bytes
RPC:
NFS:  ----- SUN NFS -----
NFS:
NFS:  Proc = 11 (Rename)
NFS:  File handle = 000016430000000100080000305A1C47
NFS:              597A0000000800002046314AFC450000
NFS:  File name = MTra00192
NFS:  File handle = 000016430000000100080000305A1C47
```

```
NFS:          597A0000000800002046314AFC450000
NFS:  File name = .nfs08
NFS:
```

View just the NFS packets between **sunroof** and **boutique**:

example$ snoop –i pkts rpc nfs and sunroof and boutique

```
 1  0.0000   boutique -> sunroof    NFS C GETATTR FH=8E6C
 2  0.0046    sunroof -> boutique   NFS R GETATTR OK
 3  0.0080   boutique -> sunroof    NFS C RENAME FH=8E6C MTra00192 to .nfs08
```

Save these packets to a new capture file:

$ snoop –i pkts –o pkts.nfs rpc nfs sunroof boutique

EXIT STATUS

Unless **snoop** receives an error signal, its Exit Status is zero. All abnormal exits return **1**.

SEE ALSO

netstat(1M), **bufmod**(7M), **dlpi**(7P), **ie**(7D), **le**(7D), **pfmod**(7M)

WARNINGS

The processing overhead is much higher for realtime packet interpretation. Consequently, the packet drop count may be higher. For more reliable capture, output raw packets to a file using the –**o** option and analyze the packets off-line.

Unfiltered packet capture imposes a heavy processing load on the host computer—particularly if the captured packets are interpreted realtime. This processing load further increases if verbose options are used. Since heavy use of **snoop** may deny computing resources to other processes, it should not be used on production servers. Heavy use of **snoop** should be restricted to a dedicated computer.

snoop does not reassemble IP fragments. Interpretation of higher level protocol halts at the end of the first IP fragment.

snoop may generate extra packets as a side-effect of its use. For example it may use a network name service (NIS or NIS+) to convert IP addresses to host names for display. Capturing into a file for later display can be used to postpone the address-to-name mapping until after the capture session is complete. Capturing into an NFS-mounted file may also generate extra packets.

Setting the **snaplen**(–**s** option) to small values may remove header information required for packet interpretation for higher level protocols. For complete NFS interpretation do not set **snaplen** less than 120 bytes.

snoop requires information from an RPC request to fully interpret an RPC reply. If an RPC reply in a capture file or packet range does not have a request preceding it, then only the RPC reply header will be displayed.

NOTES

snoop requires an interactive interface.