NAME

pcap_loop, pcap_dispatch - process packets from a live capture or savefile

SYNOPSIS

```
#include <pcap/pcap.h>

typedef void (*pcap_handler)(u_char *user, const struct pcap_pkthdr *h,
                             const u_char *bytes);

int pcap_loop(pcap_t *p, int cnt,
pcap_handler callback, u_char *user);
int pcap_dispatch(pcap_t *p, int cnt,
pcap_handler callback, u_char *user);
```

DESCRIPTION

pcap_loop() processes packets from a live capture or ‘‘savefile’’ until *cnt* packets are processed, the end of the ‘‘savefile’’ is reached when reading from a ‘‘savefile’’, **pcap_breakloop**(3) is called, or an error occurs. It does **not** return when live packet buffer timeouts occur. A value of **-1** or **0** for *cnt* is equivalent to infinity, so that packets are processed until another ending condition occurs.

pcap_dispatch() processes packets from a live capture or ‘‘savefile’’ until *cnt* packets are processed, the end of the current bufferful of packets is reached when doing a live capture, the end of the ‘‘savefile’’ is reached when reading from a ‘‘savefile’’, **pcap_breakloop**() is called, or an error occurs. Thus, when doing a live capture, *cnt* is the maximum number of packets to process before returning, but is not a minimum number; when reading a live capture, only one bufferful of packets is read at a time, so fewer than *cnt* packets may be processed. A value of **-1** or **0** for *cnt* causes all the packets received in one buffer to be processed when reading a live capture, and causes all the packets in the file to be processed when reading a ‘‘savefile’’.

Note that, when doing a live capture on some platforms, if the read timeout expires when there are no packets available, **pcap_dispatch**() will return 0, even when not in non-blocking mode, as there are no packets to process. Applications should be prepared for this to happen, but must not rely on it happening.

callback specifies a *pcap_handler* routine to be called with three arguments: a *u_char* pointer which is passed in the *user* argument to **pcap_loop**() or **pcap_dispatch**(), a *const struct pcap_pkthdr* pointer pointing to the packet time stamp and lengths, and a *const u_char* pointer to the first **caplen** (as given in the *struct pcap_pkthdr* a pointer to which is passed to the callback routine) bytes of data from the packet. The *struct pcap_pkthdr* and the packet data are not to be freed by the callback routine, and are not guaranteed to be valid after the callback routine returns; if the code needs them to be valid after the callback, it must make a copy of them.

The bytes of data from the packet begin with a link-layer header. The format of the link-layer header is indicated by the return value of the **pcap_datalink**(3) routine when handed the **pcap_t** value also passed to **pcap_loop**() or **pcap_dispatch**(). *https://www.tcpdump.org/linktypes.html* lists the values **pcap_datalink**() can return and describes the packet formats that correspond to those values. The value it returns will be valid for all packets received unless and until **pcap_set_datalink**(3) is called; after a successful call to **pcap_set_datalink**(), all subsequent packets will have a link-layer header of the type specified by the link-layer header type value passed to **pcap_set_datalink**().

Do **NOT** assume that the packets for a given capture or ‘‘savefile‘‘ will have any given link-layer header type, such as **DLT_EN10MB** for Ethernet. For example, the "any" device on Linux will have a link-layer header type of **DLT_LINUX_SLL** or **DLT_LINUX_SLL2** even if all devices on the system at the time the "any" device is opened have some other data link type, such as **DLT_EN10MB** for Ethernet.

RETURN VALUE

pcap_loop() returns **0** if *cnt* is exhausted or if, when reading from a ‘‘savefile’’, no more packets are available. It returns **PCAP_ERROR_BREAK** if the loop terminated due to a call to **pcap_breakloop**() before any packets were processed, **PCAP_ERROR_NOT_ACTIVATED** if called on a capture handle that has been created but not activated, or **PCAP_ERROR** if another error occurs. It does **not** return when live packet buffer timeouts occur; instead, it attempts to read more packets.

pcap_dispatch() returns the number of packets processed on success; this can be 0 if no packets were read from a live capture (if, for example, they were discarded because they didn’t pass the packet filter, or if, on platforms that support a packet buffer timeout that starts before any packets arrive, the timeout expires before any packets arrive, or if the file descriptor for the capture device is in non-blocking mode and no packets were available to be read) or if no more packets are available in a ‘‘savefile.’’ It returns **PCAP_ERROR_BREAK** if the loop terminated due to a call to **pcap_breakloop**() before any packets were processed, **PCAP_ERROR_NOT_ACTIVATED** if called on a capture handle that has been created but not activated, or **PCAP_ERROR** if another error occurs. **If your application uses pcap_breakloop(), make sure that you explicitly check for PCAP_ERROR and PCAP_ERROR_BREAK, rather than just checking for a return value < 0.**

If **PCAP_ERROR** is returned, **pcap_geterr**(3) or **pcap_perror**(3) may be called with *p* as an argument to fetch or display the error text.

BACKWARD COMPATIBILITY

In libpcap versions before 1.5.0, the behavior when *cnt* was **0** was undefined; different platforms and devices behaved differently, so code that must work with these versions of libpcap should use **-1**, not **0**, as the value of *cnt*.

SEE ALSO

pcap(3)