Raw socket

Il sistema compie molte operazioni sui livelli bassi della rete, ma che non sono immediatamente visibili all'utente tramite l'interfaccia delle socket

- L'intercettazione di un pacchetto IP
  - Anche se non era indirizzato a noi (sniffing)
- La generazione di un pacchetto (IP) partendo da un array di byte
  - Possiamo creare un header arbitrario (spoofing)

Cose certe ...

- Java ? No grazie !
- La programmazione dipende fortemente dal sistema operativo

  Gli header con cui interpretiamo/costruiamo i pacchetti sono molto specifici per ogni implementazione (BSD/Linux/SUN/HP)
- Dobbiamo essere superuser per poter utilizzare queste funzionalità

PCAP

- PCAP (Packet CAPture) è una libreria scritta da Van Jacobson, Craig Leres e Steven McCanne all'università di Berkeley
- Attualmente è subito disponibile dopo l'installazione in molte distribibuzioni di UNIX (BSD e Linux-mandrake), in altre occorre installare software aggiuntivo (Linux-redhat e UNIX commerciali)
- Non è inclusa di default, per farne uso occorre specificare "-lpcal" alla compilazione

```
cc file.o –o file -lpcap
```

Ricezione di pacchetti

- Identificare l'interfaccia da utilizzare
- Inizializzare (aprire) l'interfaccia
- Applicare dei filtri sui pacchetti da intercettare (opzionale)
- Intercettare pacchetti
- Chiuudere l'interfaccia

Identificazione interfaccia

- Abbiamo bisogno di una stringa che identifica la periferica per il sistema operativo (e.g. `"eth0"`)
- Abbiamo due possibilità:
  - L'utente ce la fornisce (con un parametro)
  - La richiediamo al sistema
- I nomi delle interfacce variano da sistema a sistema
  - Linux: `ethX`
  - BSD: `xlX, deX, edX ...`

`pcap_lookupdev()`

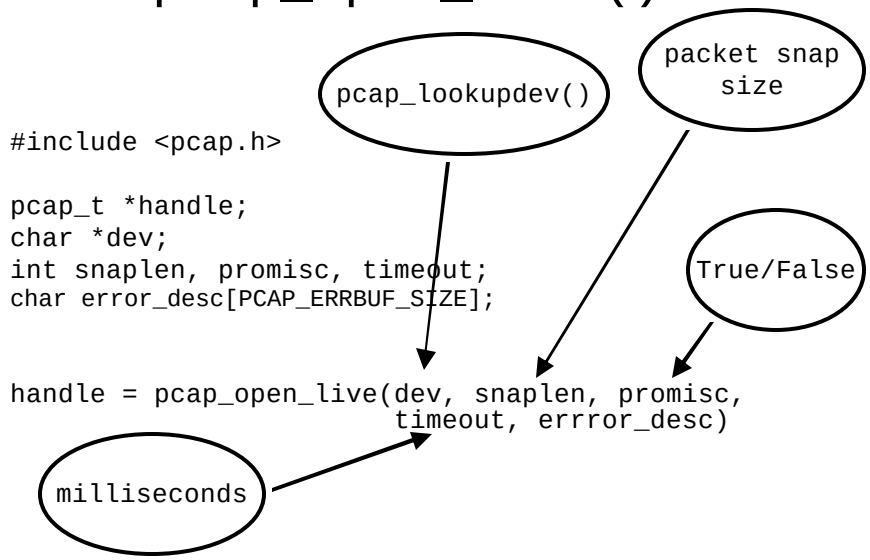
```
#include <pcap.h>

char error_desc[PCAP_ERRBUF_SIZE];
char *dev;

dev = pcap_lookupdev(error_desc)
```

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

int main() {
    char *dev, errbuf[PCAP_ERRBUF_SIZE];
    dev = pcap_lookupdev(errbuf);
    if (dev == null)
        printf(“Error: %s\n”, errbuf);
    else
        printf("Device: %s\n", dev);
}
```

Apertura dell’interfaccia

- Usiamo la funzione `pcap_open_live()` per ottenere un handle dell’interfaccia da cui leggere i pacchetti
- L’interfaccia può essere aperta in due modalità
  - Standard: vedo solo i pacchetti indirizzati alla mia macchina
  - Promisqua: vedo tutti i pacchetti sulla mia sottorete (TUTTI!)

pcap_open_live()




```
#include <pcap.h>

pcap_t *handle;
char *dev;
int snaplen, promisc, timeout;
char error_desc[PCAP_ERRBUF_SIZE];


handle = pcap_open_live(dev, snaplen, promisc,
                        timeout, errror_desc)
```

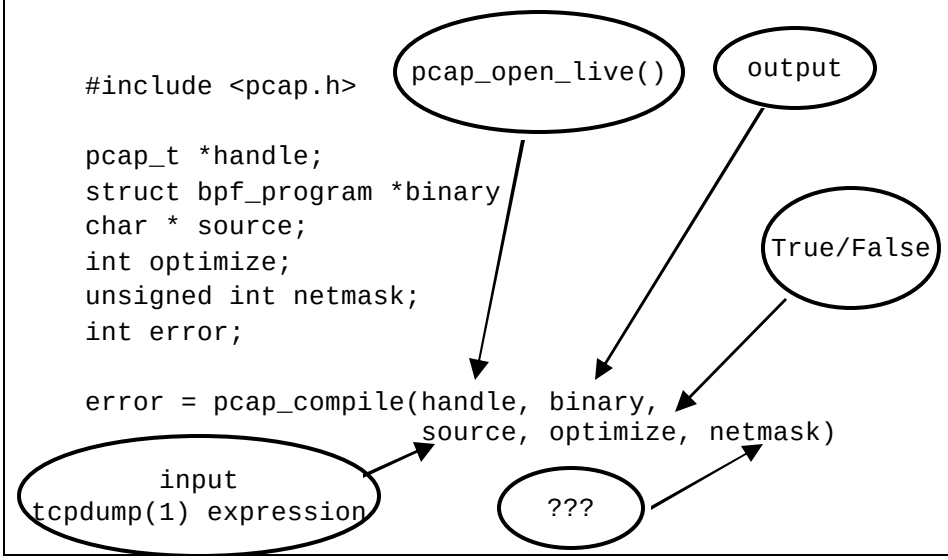
```
#include <pcap.h>

...

char *dev;
char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *handle;
char packet[PACKET_SIZE];

...

dev = pcap_lookupdev(errbuf);
printf("Device: %s\n", dev);

handle = pcap_open_live(dev, PACKET_SIZE, 1, 0, errbuf);
```

Filtri

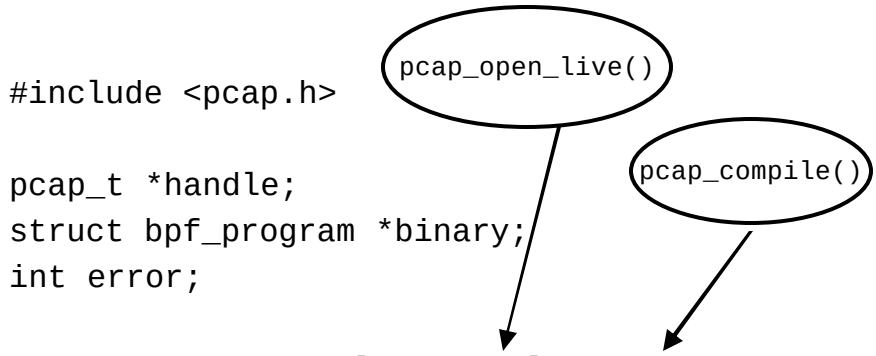
- Se la rete è molto congestionata potremmo non voler vedere tutto il traffico ma solo una parte (e.g. quello verso il server web)
- Risulta utile poter scrivere dei filtri in modo che l'interfaccia selezioni "a monte" quello che il programma vedrà

- La creazione di un filtro avviene in due passaggi:
  - La "compilazione", che permette di passare da una rappresentazione tramite stringa ad una binaria
  - L'associazione della rappresentazione binaria all'interfaccia

pcap_compile()



pcap_setfilter()

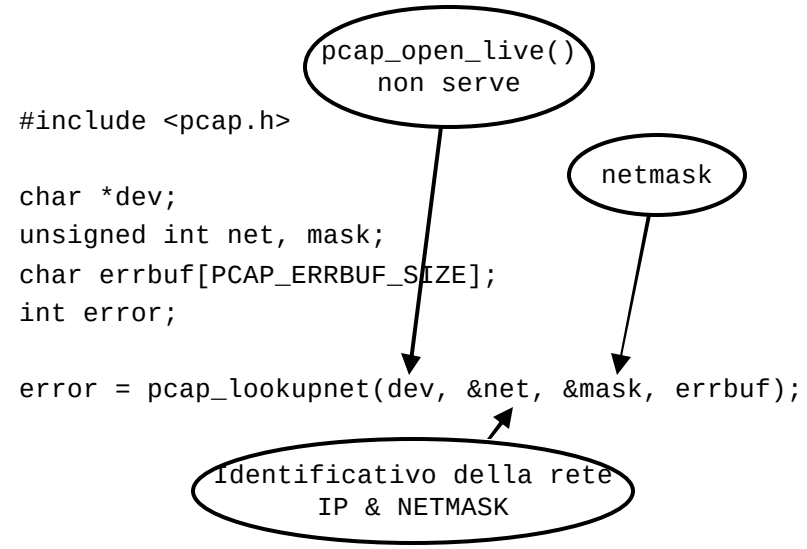



```
#include <pcap.h>

pcap_t *handle;
struct bpf_program *binary;
int error;

error = pcap_setfilter(handle, binary)
```

L’informazione mancante

- A volte è necessario passare parametri relativi alla configurazione della scheda di rete alle funzioni di PCAP
- Potremmo richiederli all’utente o andare a interpretare i file di configurazione
- In realtà PCAP ci mette a disposizione una funzione per chiedere alla scheda stessa come è configurata (`pcap_lookupnet()`)

pcap_lookupnet()



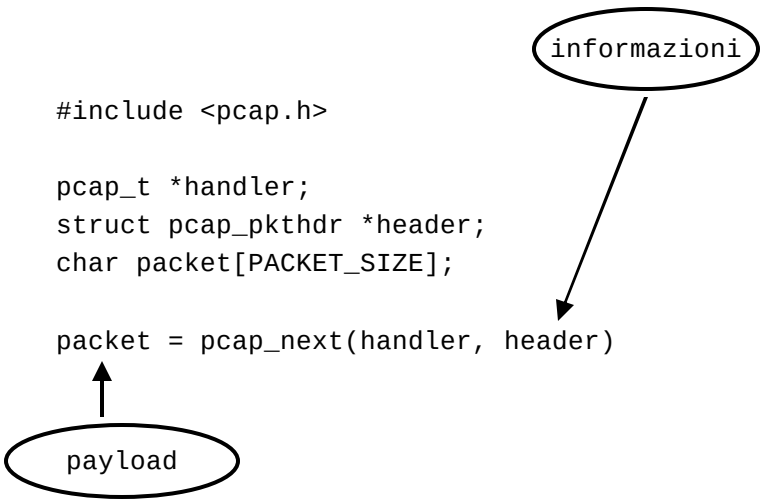

```
#include <pcap.h>

...
char *dev = “eth0”;
unsigned int net, mask;
char errbuf[PCAP_ERRBUF_SIZE];
pcap_t *handle;
char packet[PACKET_SIZE];
struct bpf_program binary;
char * source = “port 80”;
...
pcap_lookupnet(dev, &net, &mask, errbuf);
handle = pcap_open_live(dev, PACKET_SIZE, 1, 0, errbuf);
pcap_compile(handle, &binary, source, 0, net);
pcap_setfilter(handle, &binary);
...
```

Ricezione dei pacchetti

Abbiamo due possibilità:

- Prelevare un pacchetto alla volta
- Predisporre una routine che verrà chiamata ogni volta che arriva un pacchetto fino al raggiungimento di un numero prefissato

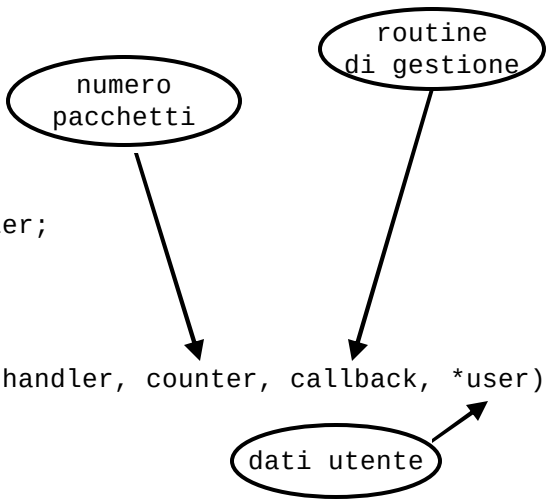
pcap_next()



```
struct pcap_pkthdr {
  struct timeval ts;
     /* time stamp */
  bpf_u_int32 caplen;
     /* length of portion present */
  bpf_u_int32 len;
     /* length this packet (off wire) */
};
```

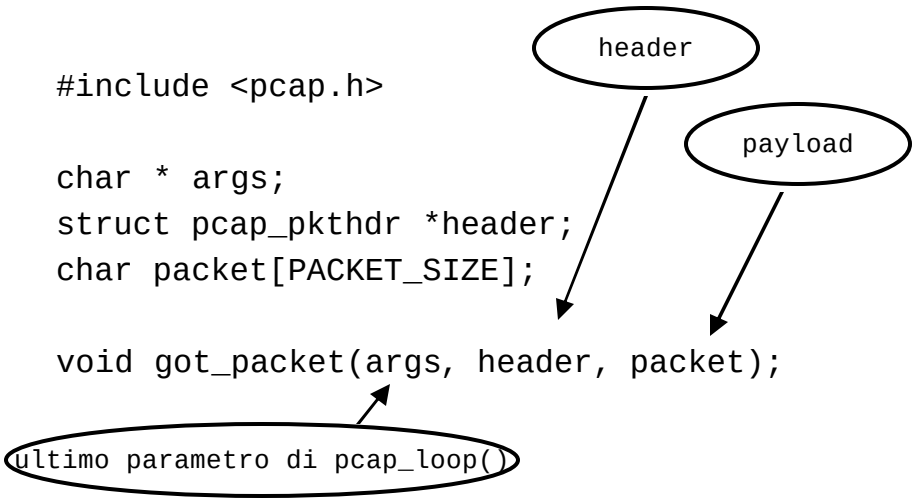
pcap_loop()

```
#include <pcap.h>

pcap_t *handler;
int counter;
pcap_handler handler;
char * user;
int error;

error = pcap_loop(handler, counter, callback, *user)
```



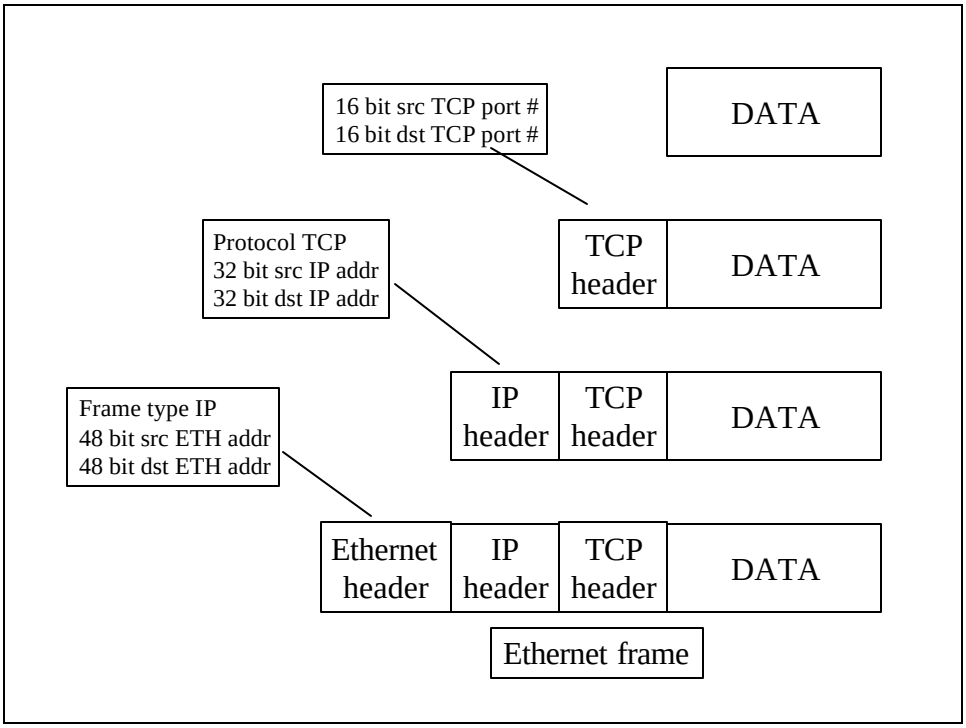
La routine di gestione

```
#include <pcap.h>

char * args;
struct pcap_pkthdr *header;
char packet[PACKET_SIZE];

void got_packet(args, header, packet);
```



Interpretazione di un pacchetto

- Quello che otteniamo tramite le routine di PCAP è un array di byte
- È compito nostro interpretare il contenuto

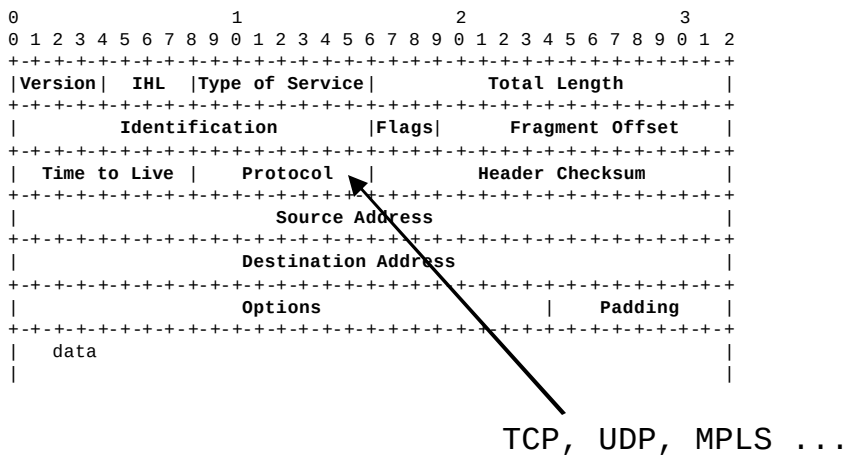


Ethernet header

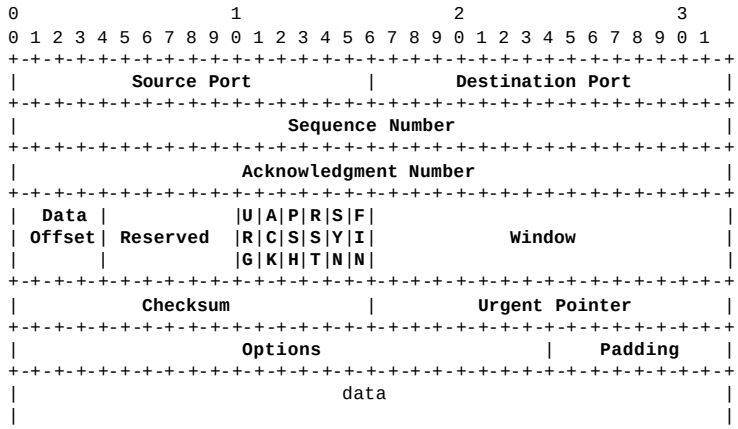
```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Ethernet destination address (first 32 bits)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Ethernet dest (last 16 bits)  |Ethernet source (first 16 bits)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Ethernet source address (last 32 bits)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type code              |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   data                                                        |
|                                                               |
    ...                                                         |
|                                                               |
|   end of data                                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Ethernet Checksum                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IP, ARP, RARP ...

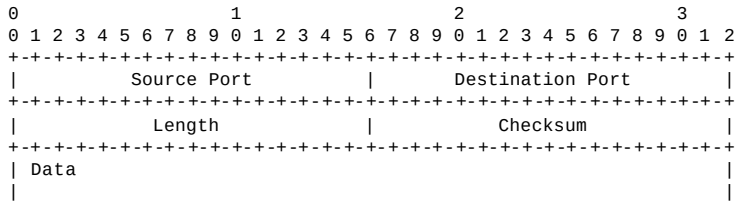
IP header



TCP header



UDP header



Interpretazione di un pacchetto

- L’unica cosa di cui siamo sicuri è che l’inizio dell’array corrisponde ad un header ethernet
- Il sistema operativo mette a disposizione delle strutture che interpretano gli header usando i loro campi

```
char * packet;
struct ether_header * eptr; /* net/ethernet.h */
struct iphdr * iph;         /* netinet/ip.h */
struct tcphdr * tcph;       /* netinet/tcp.h */
struct udphdr * udph;       /* netinet/udp.h */

...

eptr = (struct ether_header *) packet;

if (ntohs (eptr -> ether_type) == ETHERTYPE_IP) {
    printf("IP packet\n“);
    iph = (struct iphdr *) (packet +
                            sizeof(struct ether_header));
    if (ntohs (iph -> ip_p) == ) {
        tcph = (struct tcphdr *) (iph + sizeof(struct iphdr));
```

```
        printf(“TCP destination port is %x\n”,
                                ntohs(tcph -> th_dport);
    }
    else if (ntohs (iph -> ip_p ==) {
        udph = (struct udphdr *) (iph + sizeof(struct iphdr));
        printf(“UDP destination port is %x\n”,
                                ntohs(udph -> uh_dport);
    }
    else {
        printf(“IP type not TCP nor UDP“
    }
}
else if (ntohs (eptr -> ether_type) == ETHERTYPE_ARP) {
    printf("ARP packet\n“);
else {
    printf("Ethernet type %x not IP nor ARP“,
           ntohs(eptr->ether_type));
}
```