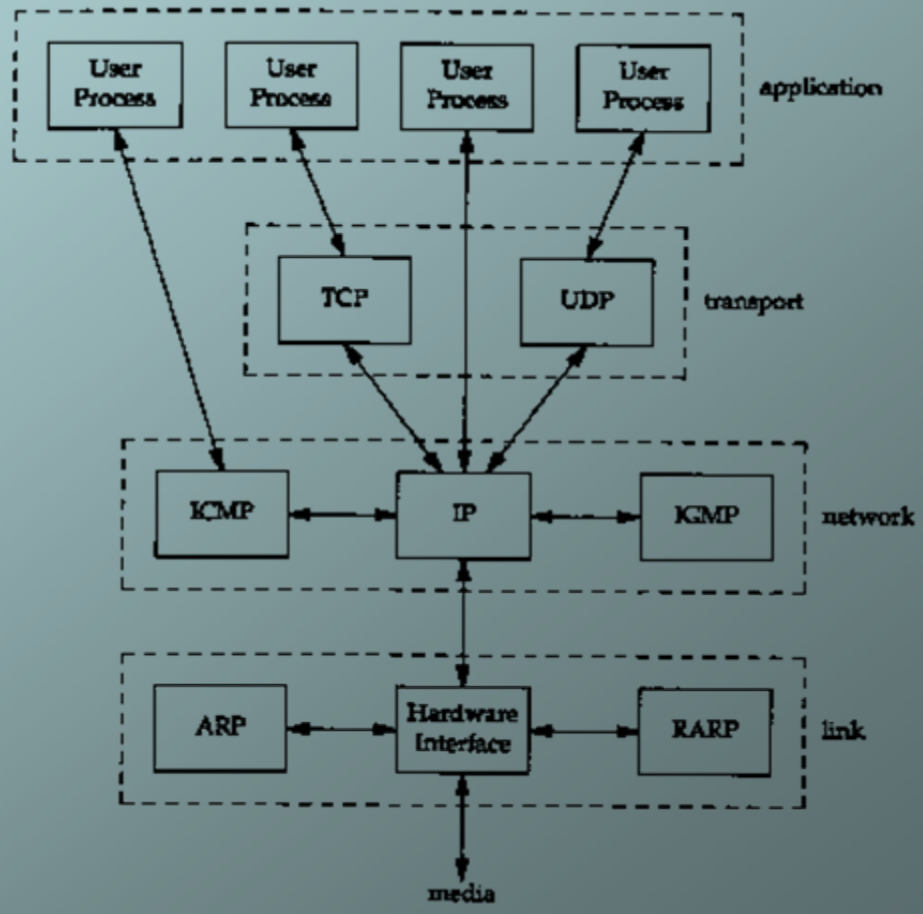
Tecniche Automatiche di Acquisizione Dati

Elementi di TCP/IP

La struttura

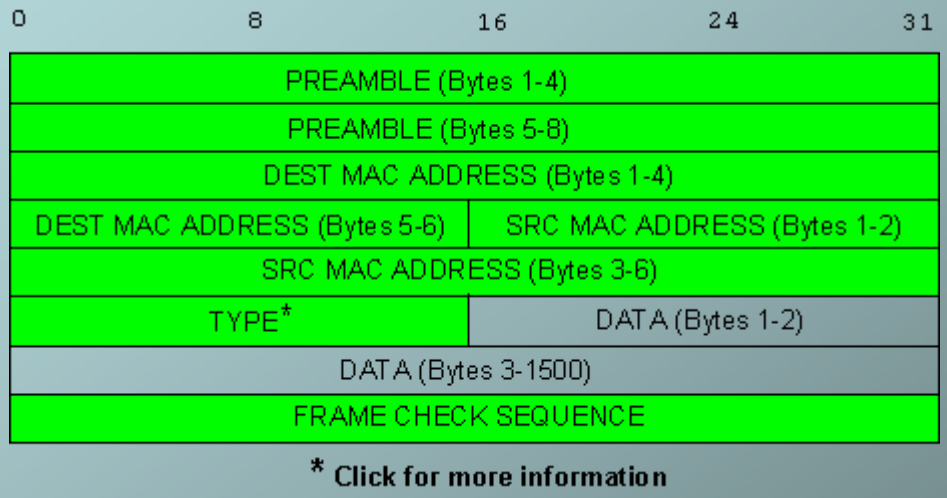
- TCP/IP è, più che un protocollo, quella che si dice una suite di protocolli, cioè la combinazione di più protocolli a vari livelli.
- Normalmente lo si considera come una struttura a 4 livelli:



Il Link Layer

- L'unità fondamentale del network fisico è il Frame:

Sono possibili diversi tipi di Frame, a seconda del protocollo di Link scelto; per esempio Ethernet si presenta come segue:



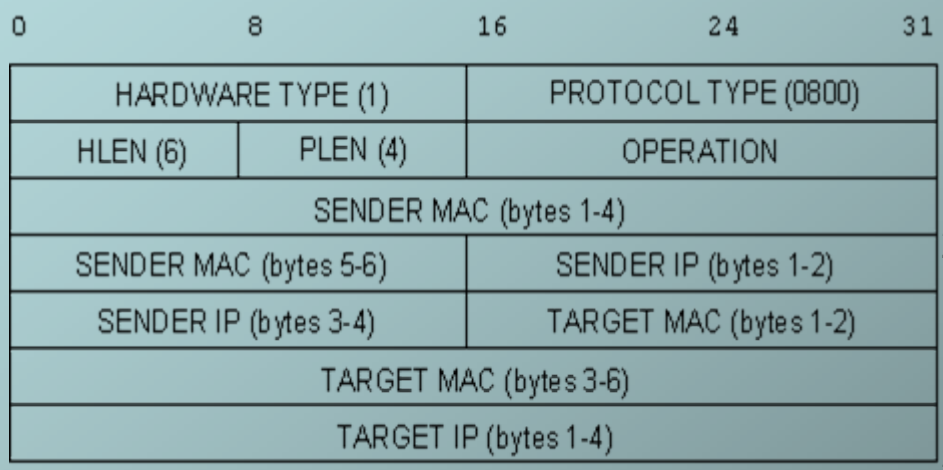
Ogni network fisico ha una lunghezza massima del frame: la maximum transfer unit (MTU):

- Ethernet MTU - 1500 bytes
- Token Ring MTU – 4192 bytes

Alcuni network fisici hanno una lunghezza minima e devonno essere riempiti tipicamente di zeri) a quella lunghezza quando I dati sono meno del minimo (Ethernet ha minimum frame size di 60 bytes)

Il protocollo ARP

- L'Address Resolution Protocol (ARP) serve ad associare l'indirizzo MAC delle schede di rete agli indirizzi del protocollo IP.
- Il pacchetto ARP è incapsulato all'interno del frame del Link layer.



1. Il richiedente manda un pacchetto ARP in Broadcast per richiedere l'indirizzo MAC corrispondente all'indirizzo IP del destinatario (ARP Request)
2. Il destinatario manda un messaggio ARP direttamente al richiedente avendo riempito il campo del suo MAC e memorizza il MAC del mittente
3. Opzionalmente altri host nello stesso dominio broadcast memorizzano il MAC

Esempio con Ethernet

Source IP address = 128.192.6.7 (**80 C0 06 07**)
Source MAC address = **00 00 C0 8D 9C FB**
Destination IP address = 128.192.6.193 (**80 C0 06 C1**)
Destination MAC address = **00 00 1D E5 A3 B9**

Arp Request

PREAMBLE	FF FF FF FF FF FF	00 00 C0 8D 9C FB
08 06	HARDWARE TYPE (01)	PROTOCOL TYPE (0800)
HLEN (06)	PLEN (04)	OPERATION (01)
00 00 C0 8D 9C FB		80 C0 06 07
00 00 00 00 00 00		80 C0 06 C1
FRAME CHECK SEQUENCE		

ARP

Who has?

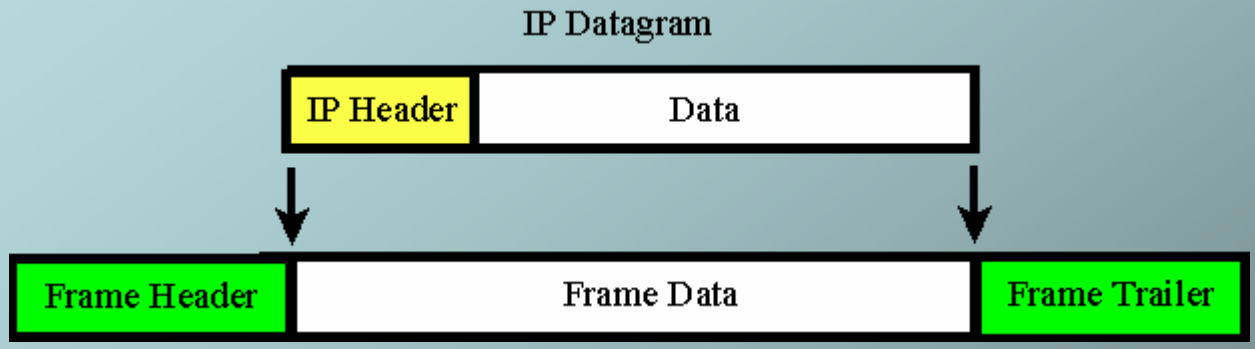
Arp Reply

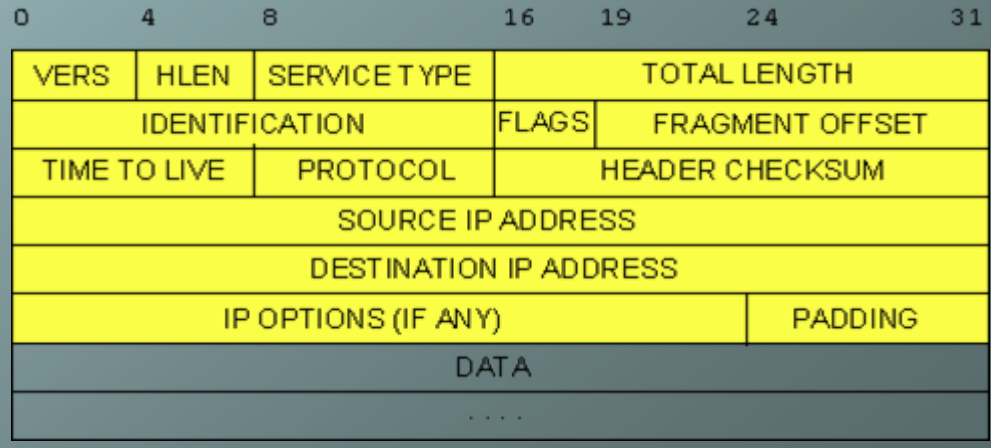
PREAMBLE	00 00 C0 8D 9C FB	00 00 1D E5 A3 B9
08 06	HARDWARE TYPE (01)	PROTOCOL TYPE (0800)
HLEN (06)	PLEN (04)	OPERATION (02)
00 00 1D E5 A3 B9		80 C0 06 C1
00 00 C0 8D 9C FB		80 C0 06 07
FRAME CHECK SEQUENCE		

Per minimizzare il numero di broadcast sula rete, gli host IP e I gateway memorizzano le associazioni IP-MAC in delle Address Resolution Tables in cui timagono per un tempo massimo detto Time to Live (TTL)

Il protocollo IP

- Al livello Network abbiamo il protocollo IP. Esso è:
  - **unreliable** (consegna non garantita)
  - **connectionless** (pacchetti indipendenti gli uni dagli altri)
  - **best-effort** (tenta di consegnare il pacchetto)
- L'unità base è il **datagram** (65535 byte)

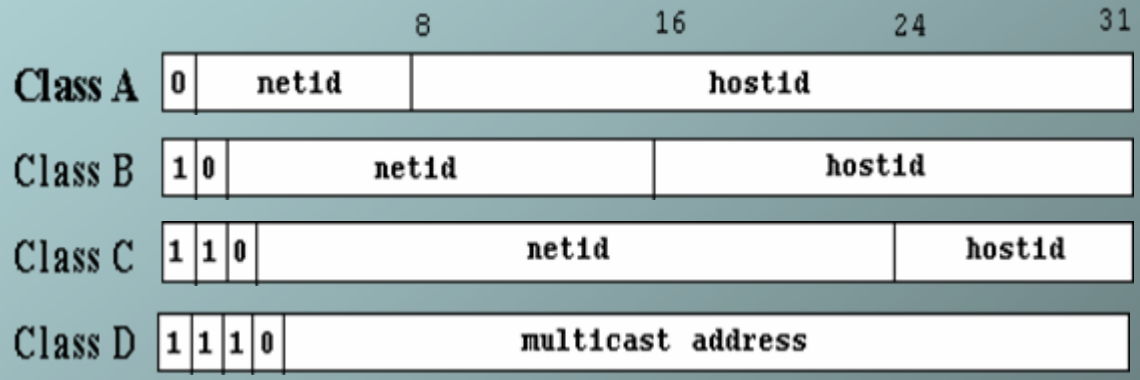




- Gli host IP mandano dati fino alla MTU della network fisica
- I router possono frammentare i datagram più lunghi della MTU a valle.
- Ciascun frammento ha la strutura di un datagram IP.

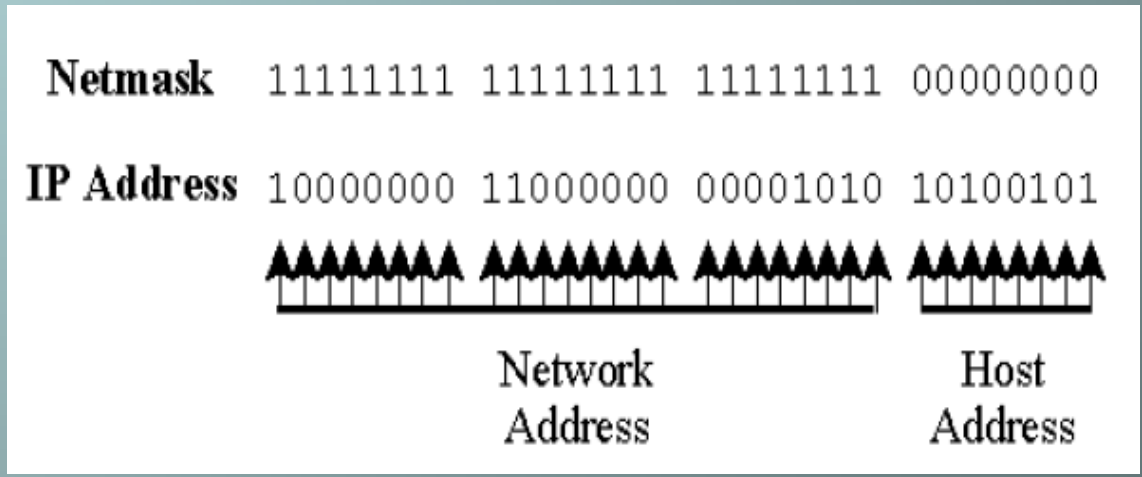
Gli indirizzi IP

- Sono indirizzi univoci a 32 bit, normalmente raggruppati a gruppi di 8.
- Corrispondono a connessioni, non a host: se si muove l'host, cambia l'indirizzo.
- Formano tre classi principali A, B, C e la classe di multicast, classe D, della forma [net_id, host_id]



Sottoreti e netmask

- Una rete IP si può suddividere in sottoreti quasi indipendenti sulla base degli indirizzi
- Per la suddivisione si utilizza una “maschera”, cioè un numero a 32 bit che determina quale porzione dell’indirizzo è la parte di rete e quale quella di host.
  - Un bit di indirizzo IP è di rete se il corrispondente bit della netmask è a 1
  - È di host se il bit corrispondente della netmask è 0.
- Di fatto, I bit 0 della netmask indicano quali bit possono cambiare nell’indirizzo per identificare gli host.
- Esempio: IP-Address: 128.192.10.165; netmask: 255.255.255.0



Indirizzi: convenzioni e classi

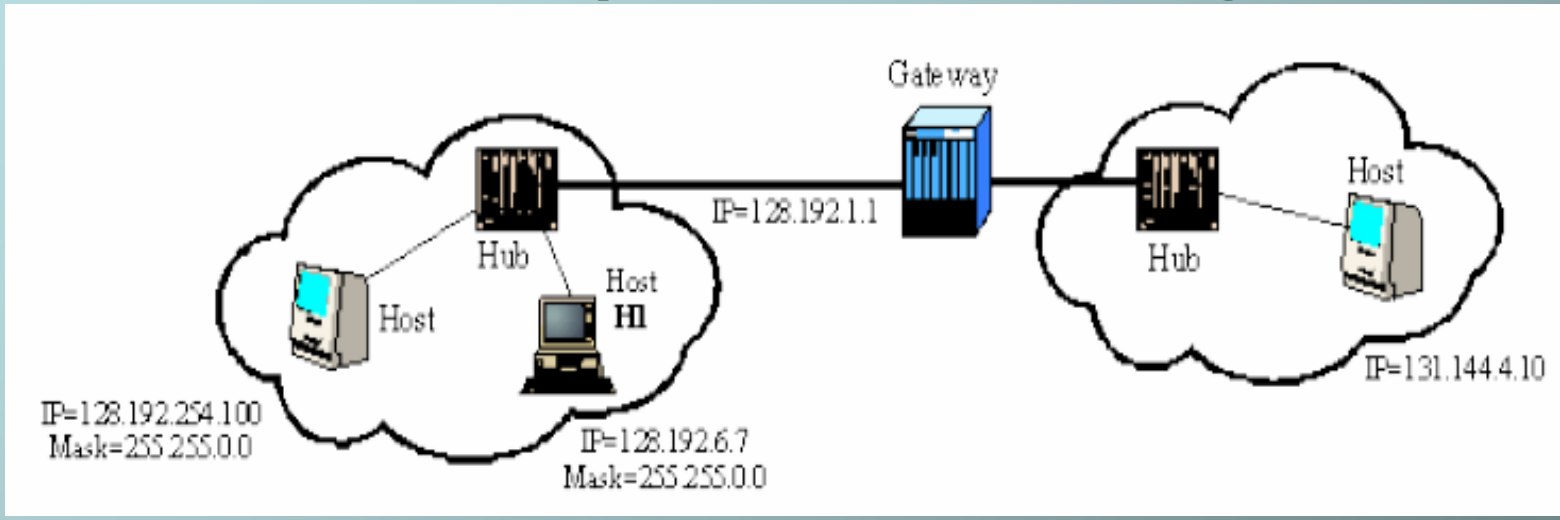
- Per convenzione gli indirizzi delle reti hanno a 0 i bit di host.
- Gli indirizzi di broadcast diretto hanno ad 1 tutti i bit di host
- I broadcast “limitati” hanno tutti i bit a 1
- L’indirizzo 127.0.0.1 (loopback) è solo per uso di test e non genera traffico sulla rete

- Intervalli di indirizzi per classi.
  - Class A => 1.0.0.0 - 126.0.0.0
  - Class B => 128.xxx.0.0 - 191.xxx.0.0
  - Class C => 192.xxx.yyy.0 - 223.xxx.yyy.0
  - Class D => 224.xxx.yyy.zzz - 239.xxx.yyy.zzz (multicast IP)

IP Routing

- Il Routing (instradamento) permette a host di una sottorete di comunicare con host di una sottorete distinta.
  - Routing diretto:trasmettere un datagram da un computer ad uno sulla stessa sottorete fisica
  - Routing indiretto: il destinatario non è sulla stessa sottorete: il datagram deve essere smistato da un router.
  - Routing basato sulle IP routing tables (net address, netmask, next-hop)
    - Gli host hanno delle routing tables minimali

Esempio di routing



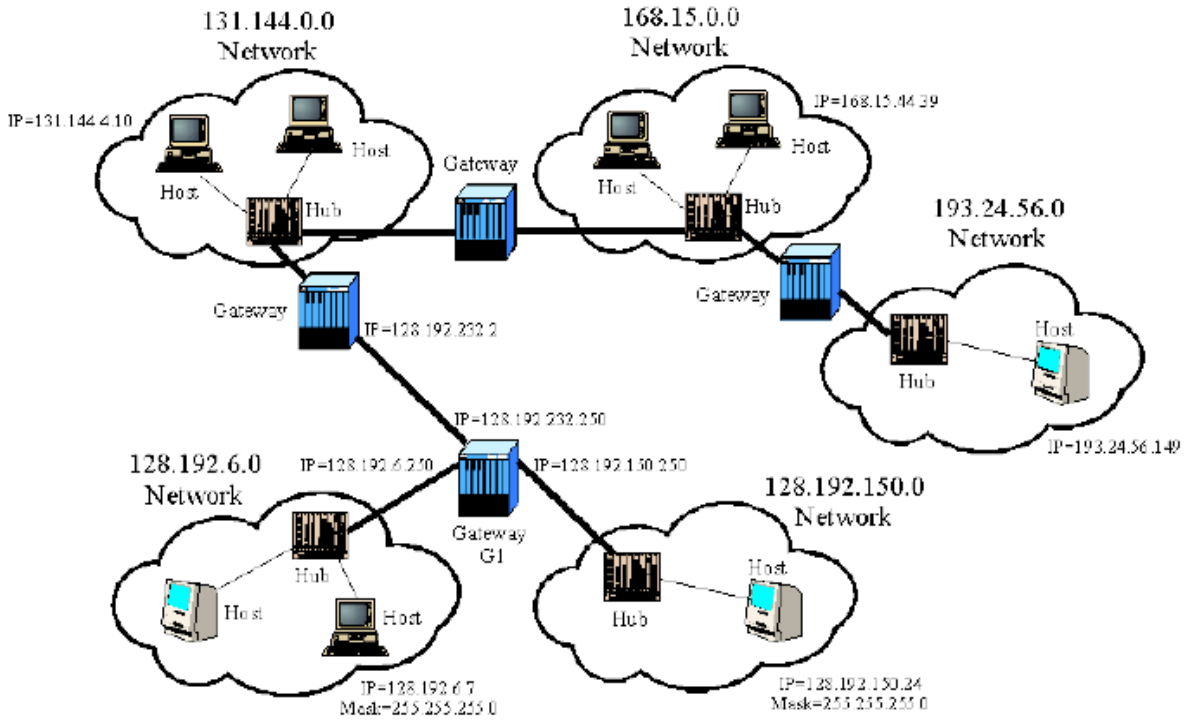
Routing table per H1

entry	netmask	net-address	next-hop	hop-count	(comments)
1	255.255.0.0	128.192.0.0	128.192.6.7	0	DCN
2	0.0.0.0	0.0.0.0	128.192.1.1	1	DR

Direct connected Network

Default route

Altro esempio

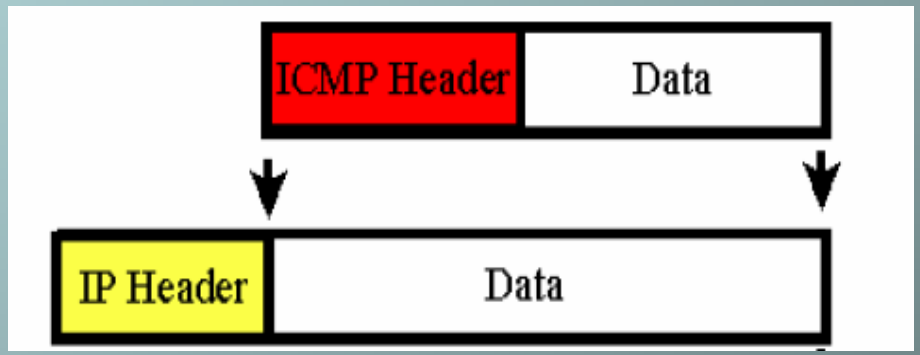


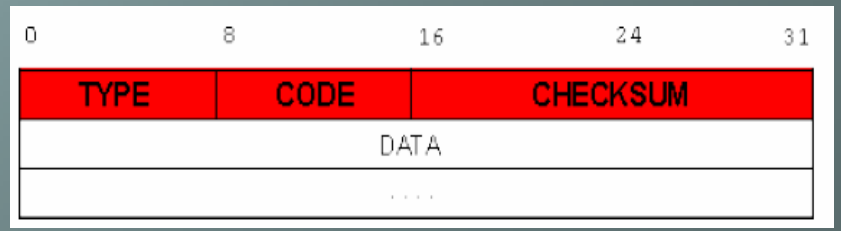
- Routing table for gateway **G1**

entry	netmask	net-address	next-hop	hop-count	(comments)
1	255.255.255.0	128.192.6.0	128.192.6.250	0	DCN
2	255.255.255.0	128.192.7.0	128.192.7.250	0	DCN
3	255.255.255.0	128.192.150.0	128.192.150.250	0	DCN
4	255.255.255.0	128.192.232.0	128.192.232.250	0	DCN
5	255.255.255.255	131.144.4.10	128.192.232.2	1	HSR
6	255.255.0.0	168.15.0.0	128.192.232.2	2	NSR
7	0.0.0.0	0.0.0.0	128.192.232.2	1	DR

ICMP

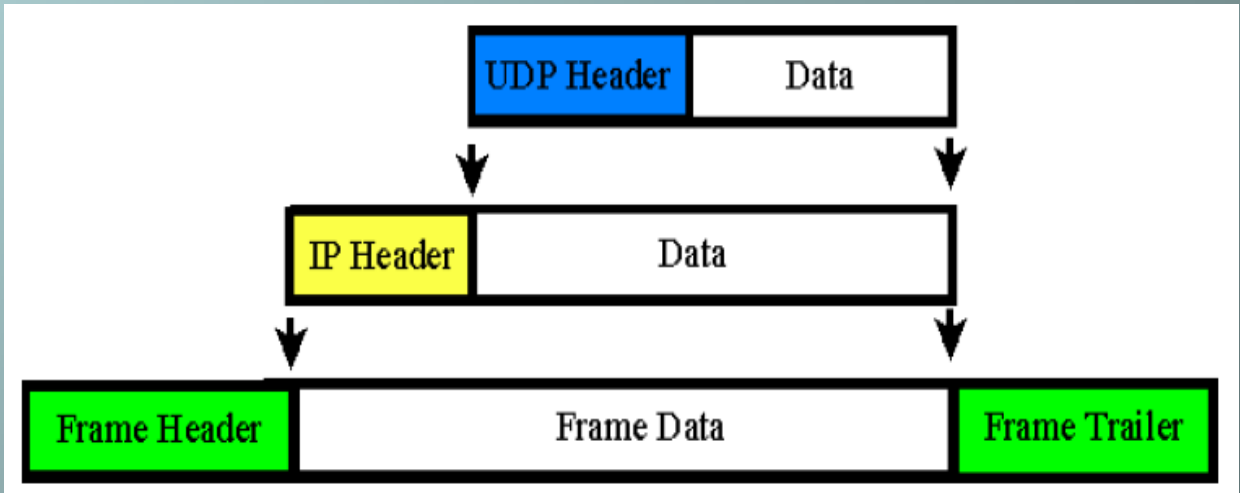
- Siccome le reti sono inerentemente inaffidabili c'è bisogno di un meccanismo per il report di problemi di consegna dei datagram.
- ICMP (**I**nternet **C**ontrol **M**essage **P**rotocol)
  - È richiesta l'implementazione per ogni software TCP/IP
  - I messaggi ICMP vengono rispediti all'host mittente, non ai gateways
  - Normalmente partono dai gateway ma possono partire anche dagli host.





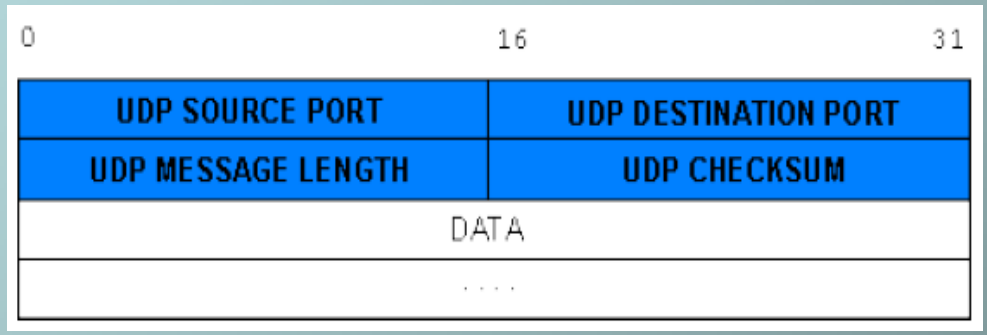
UDP/IP

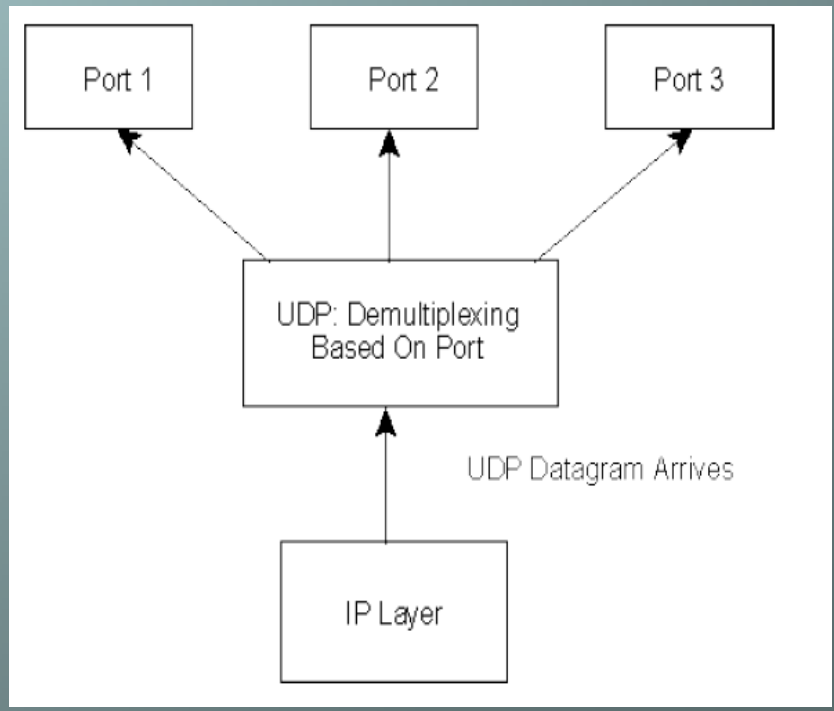
- Lo User Datagram Protocol (UDP) è un protocollo del livello di trasporto orientato ai datagram.
- Ogni operazione produce **un** datagram UDP che fa sì che sia inviato un solo datagram IP
- È un protocollo inaffidabile: manda il datagram al livelli IP ma non ne garantisce la consegna a destinazione.
- UDP non è connection-oriented



UDP Header

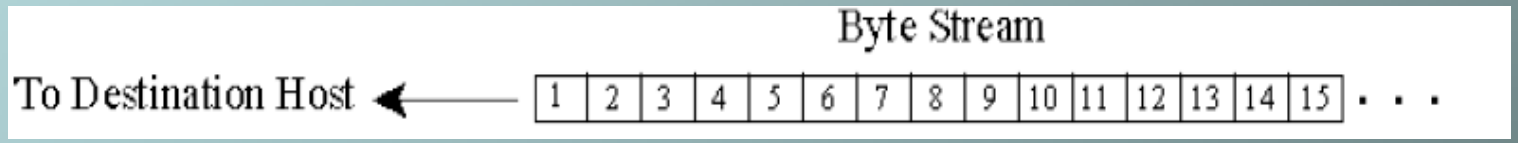
- Tramite il meccanismo delle porte è possibile mandare allo stesso host dati diretti ad applicazioni diverse (demultiplexing)
- La checksum è basata su complementi ad 1 di parole a 16 bit, ma il frame UDP può essere di qualsiasi lunghezza => se serve si aggiiunge un “pad” bit a 0.





TCP/IP

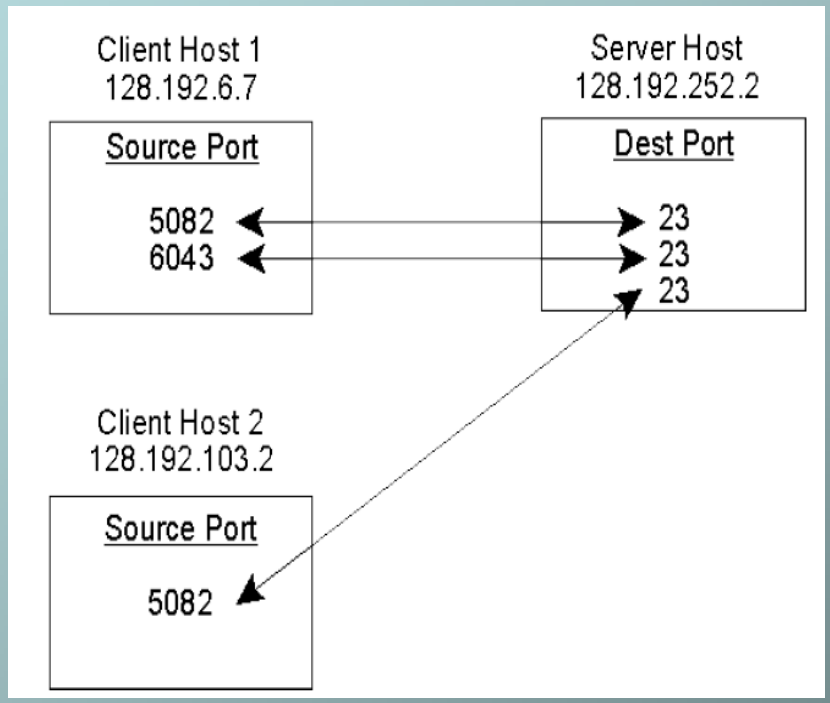
- Il Transmission Control Protocol (TCP) è un protocollo che fornisce una trasmissione reliable, end-to-end, con controllo di flusso.
- A differenza di UDP è orientato alle *stream* di dati. Si tratta di flussi di dati organizzati in bytes.

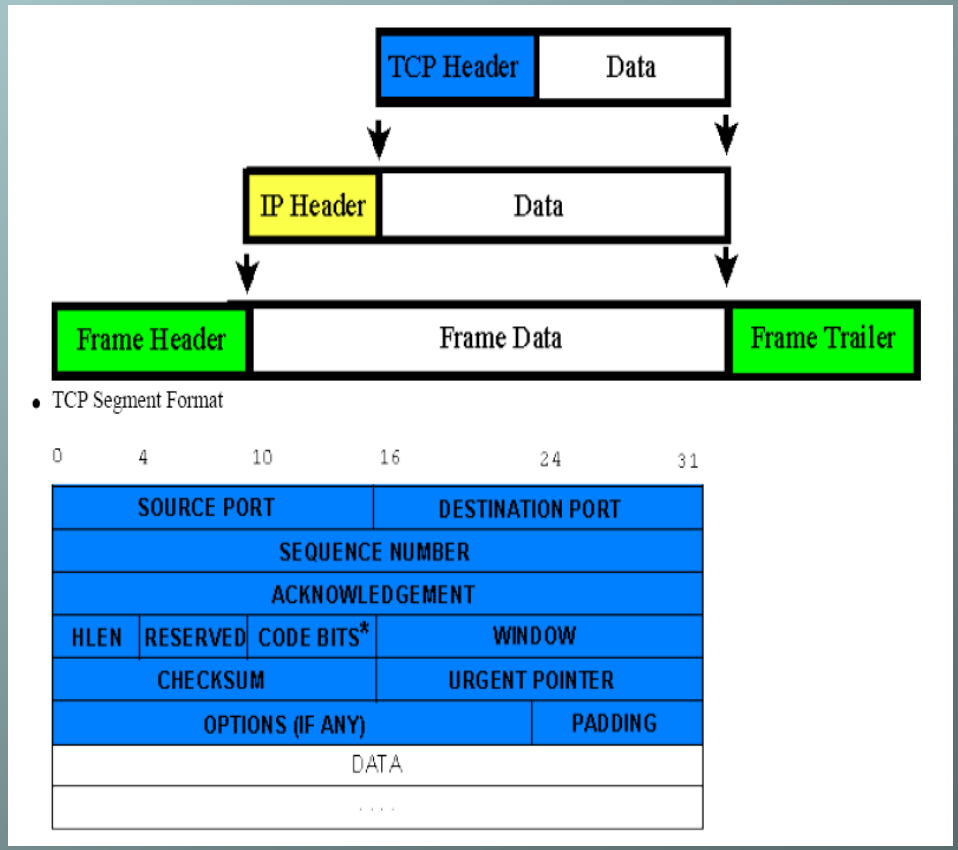


- TCP è orientato alla connessione:
  1. il client chiama il server su una determinata porta;
  2. Il server, se accetta la chiamata, stabilisce una connessione
- Il trasferimento e bufferizzato: le applicazioni consegnano i dati a TCP che li manda esattamente nell'ordine in cui gli arrivano
- La trasmissione è full duplex
- L'unità di trasferimento TCP è il *segmento*.

Segmento TCP

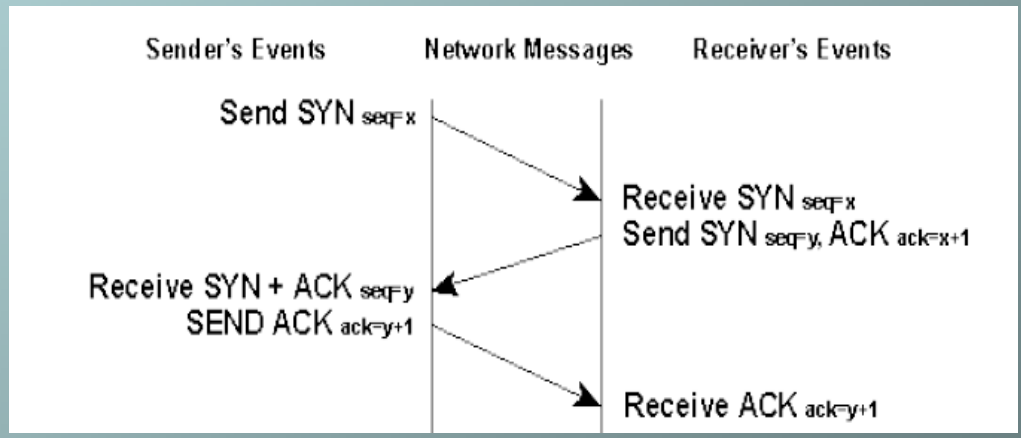
- La connessione è definita dalle coppie (**source IP**, **source port**) e (**dest IP**, **dest port**)
- Differenti client possono chiamare la stessa porta sul server purché siano differenti gli indirizzi o le porte di partenza (source port)

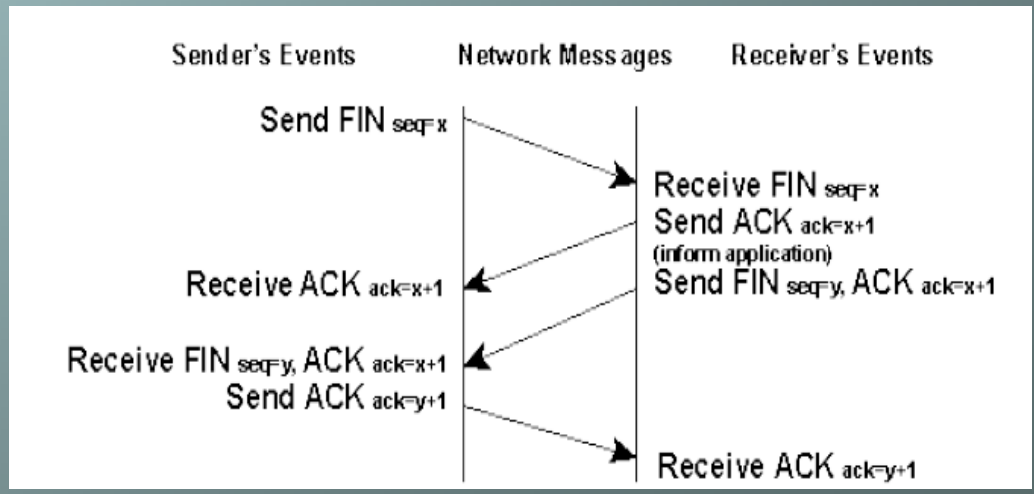




Apertura e chiusura della connessione TCP

- TCP spezza il flusso di dati in segmenti ed a ciascuno assegna un numero di sequenza.
- Il numero si sequenza iniziale, **ISN**, identifica l'inizio della stream, è casuale e negoziato tra client e server all'atto della connessione secondo uno schema detto "three-way handshaking".
- Anche la chiusura della connessione ha uno schema a tre passi, ma leggermente modifocato



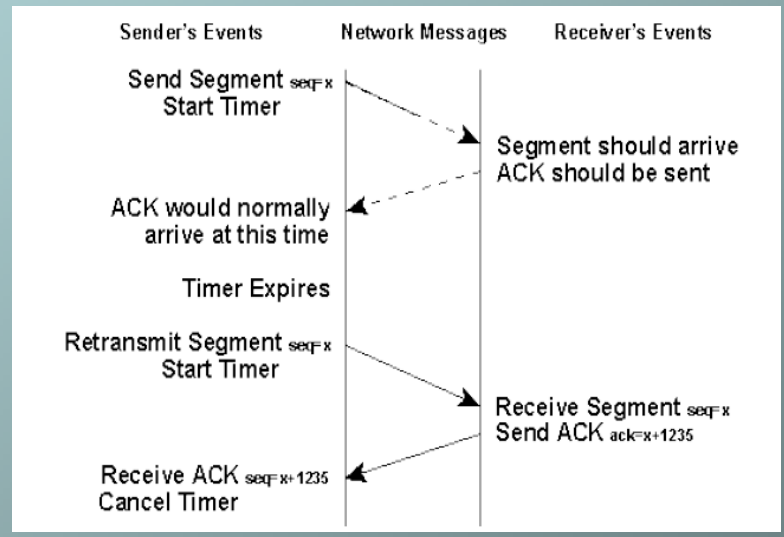


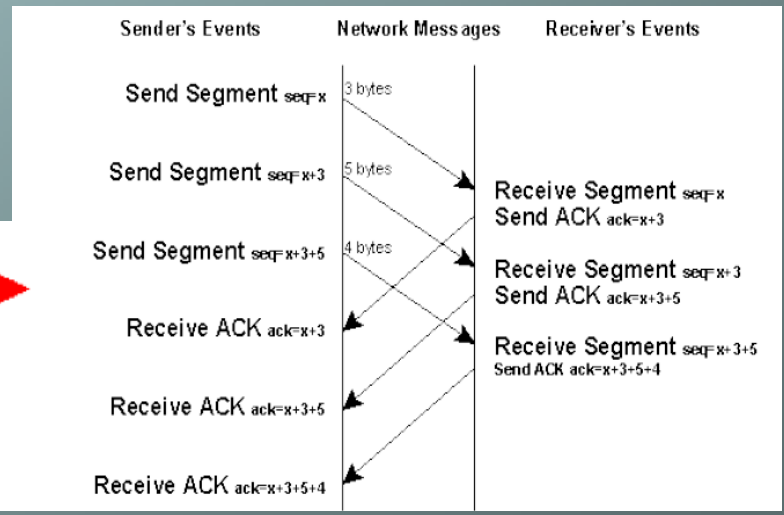
Timeout in apertura

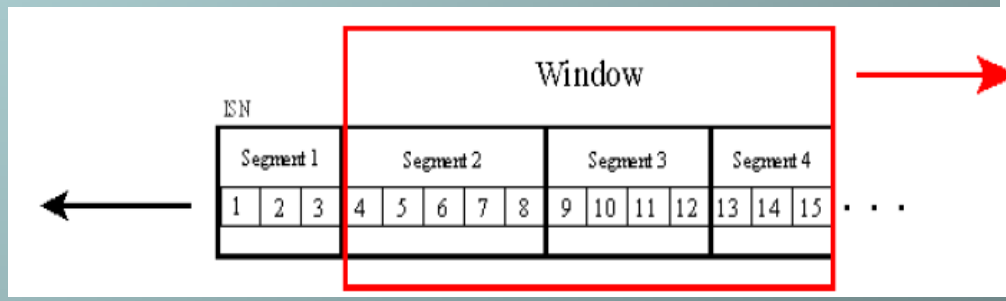
- Ci sono diversi casi nei quali l'apertura della connessione può fallire: per esempio l'host è down.
- Il chiamante invia il segmento SYN e aspetta circa 6 secondi l'ACK.
- Al secondo invio aspetta esattamente 24 secondi
- Infine continua a inviare fino ad una attesa massima di 75 secondi, dopo la quale la connessione viene dichiarata in timeout.

Controllo di flusso

- I numeri di ack dicono al mittente che il ricevente aspetta il prossimo (rispetto al numero di ACK) segmento.
- Quando viene spedito un segmento parte un timer; se ACK non viene ricevuto prima del timeout il segmento viene ri-inviato.
- Per aumentare l'efficienza si usa un controllo a sliding window: tutti i segmenti in una finestra vengono inviati senza aspettare l'ACK, la finestra scorre man mano che vengono ricevuti gli ACK.





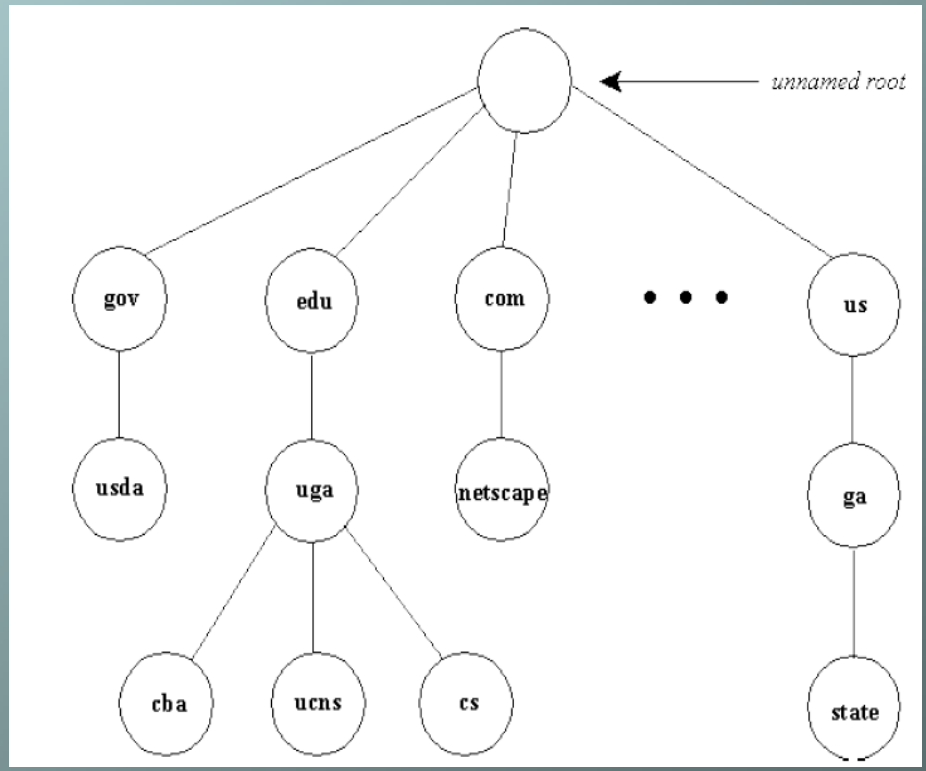


Maximum Segment Size

- La lunghezza massima del segmento inviato, MSS, può venire comunicata al momento della connessione, all'altro capo della connessione. Se un terminale non riceve l'MSS, assume un valore di default di 536.
- In generale più è grande MSS meglio è perchè così il peso degli header è minore.
- MSS è in genere la MTU dell'interfaccia diminuita della lunghezza degli header TCP e IP (quindi per Ethernet vale 1500 – 20 – 20 = 1460).

Nomi degli host

- Per gli esseri umani è molto più facile ricordare I nomi che gli indirizzi numerici, quindi agli host si danno dei nomi.
- Il mapping tra I nomi simbolici e gli indirizzi è eseguito o loalmente con delle tabelle o tramite i Domain Name Servers (DNS).
- La nomenclatura è gerarchica e stabilita da un'autorità internazionale, come la IANA (Internet Assigned Number Authority).



Risoluzione dei nomi

- Ogni implementazione di TCP/IP ha una routine chiamata Name Resolver (NR) per la richiesta ad un DNS.
- Ci sono due tipi di risoluzione:
  - Ricorsiva: NR richiede al Name Server (NS) di risolvere nomi per cui non ha l'autorità per rispondere e questi lo chiede ad altri NS con l'autorità per il dominio richiesto.
  - Iterativa: NR chiede a NS l'indirizzo di un altro NS che può fornire la risposta.

Elementi di programmazione di TCP/IP

- I programmatori hanno a disposizione delle API (Application Programming Interfaces) per avere a disposizione le informazioni sul mapping dei nomi.
- Le funzioni `gethostbyname(const char *name)` e `gethostbyaddr(const char *addr)`, restituiscono un puntatore ad una struttura `hostent`:

```
struct hostent {
char *h_name; /* official name */
char **h_aliases; /* alias list */
int h_addrtype; /* address type */
int h_length; /* address length */
char **h_addr_list; /* address list */
};
#define h_addr h_addr_list[0]
```

- Le porte di alcuni servizi sono fissate dall'autorità e sono "ben note". Sono listate in un file di sistema e sono quelle di valore inferiore a 1024.
- Esiste una funzione `getservbyname(const char *name)` che riempie una struttura `servnt` che riflette la tabella che associa i nomi ai servizi.

I Socket

- Il modello a cui si ispira la programmazione di TCP/IP è quella del cavo con la spina: per avere flusso (di corrente o di dati) bisogna prima infilare la spina nella presa (socket)
- I socket hanno un indirizzo determinato dalla coppia (host.porta) ed esiste una struttura dati `sockaddr_in` che contiene questa informazione:

```
struct sockaddr_in {
short sin_family; /* AF_INET */
u_short sin_port; /* service port */
struct in_addr sin_addr; /* host number */
char sin_zero[8]; /* not used */
};
```

```
struct sockaddr {
u_short sa_family;/* address family*/
char sa_data[14];/*max 14 byte addr*/
};
```

- Il socket si apre e si chiude come si fa con i file: I file si aprono con `open()`, i socket con `socket(int domain, int type, int protocol)` che restituisce un intero che è equivalente al file descriptor.
  - `domain` indica il la famiglia di indirizzi che si usa: per IP è `AF_INET` (o `PF_INET`)
  - `type` specifica la semantica della comunicazione: `SOCK_STREAM` per TCP, `SOCK_DGRAM` per UDP, `SOCK_RAW` per accedere al livello network.
  - `protocol` è sempre 0 per TCP/IP

Connessione del client

- Un'applicazione sul client apre un socket, e si connette tramite la funzione `connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen)`ad un'applicazione server identificata dal suo socket address:

```
#define PORT 12345
int main(int argc, char **argv)
{
  int sockfd;
  int n;
  int size;
  int port=PORT;
  struct sockaddr_in servAddr;

  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  bzero(&servAddr, sizeof(servAddr)); /* ripulisce la struttura */
  servAddr.sin_family = AF_INET;
  servAddr.sin_port=htons(port);
  servAddr.sin_addr.s_addr = inet_addr(argv[1]);

  connect(sockfd, (struct sockaddr *)&servAddr, sizeof(struct sockaddr));
```

Comunicazione del client

- Una volta connesso, il client potrà trasmettere o ricevere dati dal server. Con TCP questo può essere fatto in due modi:
  - Utilizzando le funzioni di scrittura e lettura da file per es. write() e read()
  - Utilizzando le funzioni specifiche send(int s, const void *msg, size_t len, int flags) e recv(int s, void *buf, size_t len, int flags) .
- Esempio misto:

Buffer di ricezione

Numero di byte ricevuti

```
for(;;)
  {
    if( (n = recv (sockfd, &line, 4096, 0)) ==0)
      {
         printf("connection closed by other end\n");
         return 0;
      }
     write(sockfd, &end, 1);
}
```

recv si blocca finché non riceve dati

Applicazioni server

- Il server attenderà le connessioni dai client su un determinato socket.
- Ha bisogno di una funzione che associ al socket la coppia (address, port). Questa funzione è
  - `bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen)`
- Servirà, poi una funzione che ponga il server in attesa di connessioni ed un'altra accetti la connessione dai client:
  - `int listen(int s, int queuelen)`
    - Assegna una coda di lunghezza queuelen al socket s; al massimo accetta queuelen connessioni
  - `int accept(int s, struct sockaddr *addr, socklen_t *addrlen)`
    - Estrae la prima richiesta di connessione dalla coda e crea un socket connesso con il cliente, con le proprietà del socket originario tranne che non è più in stato di attesa.

Esempio Server

```
#define PORT 12345

int main(int argc, char **argv)
{
        int listenfd, connfd;
        int cliLen;
        struct sockaddr_in cliAddr,
                    servAddr;
        pid_t child_pid;
        int port;

       listenfd = socket(AF_INET,
                    SOCK_STREAM, 0);
        bzero(&servAddr,
                sizeof(servAddr));
        servAddr.sin_family = AF_INET;
        servAddr.sin_addr.s_addr =
               htonl(INADDR_ANY);
        port = htons(PORT);
        servAddr.sin_port=port;
        bind(listenfd,
          (struct sockaddr*)&servAddr,
        sizeof(servAddr));
      //5 possible queued connections
        listen(listenfd,5);
```

```
        cliLen=sizeof(cliAddr);
        for(;;)
        {
           connfd=accept(listenfd,
                (struct sockaddr*)&cliAddr,
                 &cliLen);
           if( (child_pid=fork()) == 0)
                {
                    /* Nel processo figlio
                       rimane solo il socket
                       connesso*/
                   close(listenfd);
                    //process the request
                   procreq(connfd);
                   exit(0);
                }
           close(connfd);
        }//for
}//main
```

fork () genera un processo figlio: se ritorna 0 siamo nel figlio, se >0 nel genitore