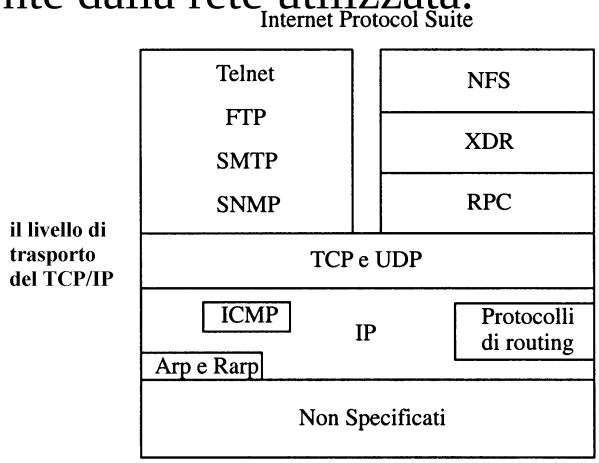
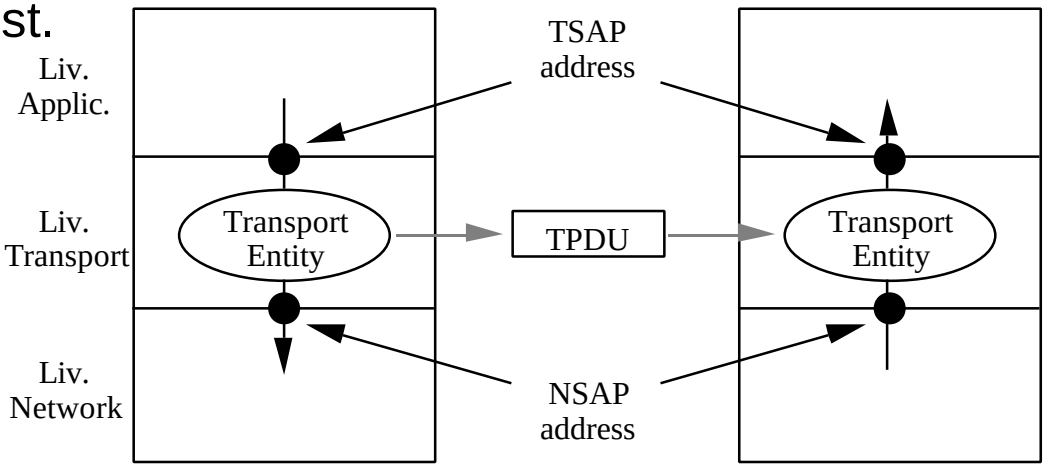
Il livello di Trasporto del TCP/IP

Il compito del livello transport (livello 4) è di fornire un trasporto efficace dall'host di origine a quello di destinazione, indipendentemente dalla rete utilizzata.



Questo è il livello in cui si gestisce per la prima volta (dal basso verso l'alto) una conversazione diretta, cioè senza intermediari, fra una transport entity su un host e la sua peer entity su un altro host.

.



Naturalmente, tali servizi sono realizzati dal livello transport per mezzo dei servizi ad esso offerti dal livello network.

Così come ci sono due tipi di servizi di livello network, ce ne sono due anche a livello transport:

• servizi affidabili orientati alla connessione, detti di tipo stream, offerti dal **TCP** (Transmission Control Protocol);

• servizi senza connessione detti di tipo datagram offerti dall' **UDP** (User Datagram Protocol).

Indirizzi a livello di Trasporto per il TCP/IP

Quando si vuole trasferire una o più **TPDU** (Transport Protocol Data Unit) da una sorgente ad una destinazione di livello 4, occorre specificare mittente e destinatario di livello 4. Il protocollo di livello 4 deve quindi decidere come deve essere fatto l'indirizzo di livello transport, detto ***TSAP address*** (***Transport Service Access Point address***).

Poichè l'indirizzo di livello 4 serve a trasferire informazioni tra applicazioni che lavorano su hosts diversi, deve poter individuare un host e una entità contenuta nell'host. Per questo motivo i protocolli di livello 4 tipicamente definiscono come indirizzo di livello 4 una coppia formata da un indirizzo di livello network che identifica l'host, e da un'altra informazione che identifica un punto di accesso in quell'host **(NSAP address, informazione supplementare)**.

Ad esempio, in TCP/IP un TSAP address (ossia un indirizzo TCP o UDP) ha la forma:

(IP address : port number)

Port number è un intero a 16 bit, che identifica un servizio o un punto di accesso e/o smistamento di livello 4.

Ad es. la coppia (137.204.72.49 : 23) indica la porta 23 dell'host poseidon.csr.unibo.it, cioè l'entry point per il demone telnet, cioè il punto di accesso all'applicazione che permette ad un utente di collegarsi mediante telnet all'host poseidon.

• Anche se l'indirizzo di livello trasporto è formato da questa coppia, **il TCP/IP**, per ridurre l'overhead causato dagli header dei vari livelli, **nella trasmissione effettua una violazione della stratificazione tra i livelli 4 e 3** (Trasporto e Network). Infatti, come vedremo TCP e UDP contengono nei loro header solo i numeri delle porte e riutilizzano gli indirizzi IP di mittente e destinaz. contenuti nel pacchetto IP.

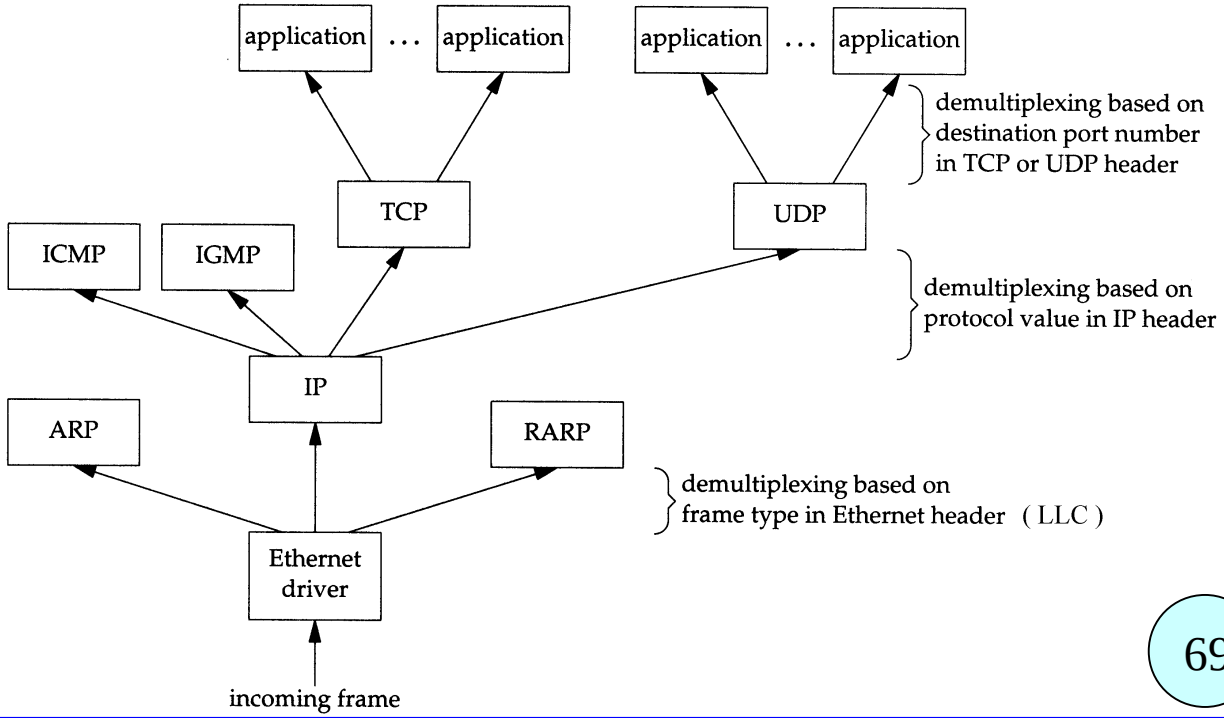
Multiplexing a livello Trasporto

• Gli **identificatori di porta** (port number) permettono di effettuare la **demultiplazione** dei pacchetti di livello 4, ovvero di discriminare l'applicazione destinazione dei pacchetti in funzione del port number contenuto nell'header del pacchetto di livello transport, sia esso di tipo TCP che UDP.

• E' ovvio che mittente e destinatario devono essere d'accordo sul valore della porta del destinatario per poter effettuare la trasmissione.

• Il mittente scrive questo numero di porta come indirizzo del destinatario, ed il destinatario si deve mettere in attesa dei pacchetti che giungono all'host del destinatario e che posseggono come identificatore proprio quel port number.

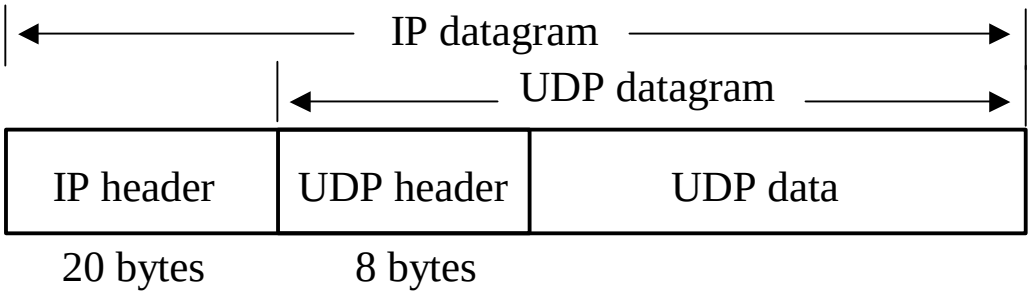
• Alcune primitive fornite dall'interfaccia socket permettono di specificare il numero di porta di cui interessa ricevere i pacchetti (stream=flussi di dati nel caso TCP). E' quindi il sistema operativo che si fa carico di effettuare le operazioni di demultiplexing dei pacchetti ricevuti dal livello network.



Il protocollo di Trasporto UDP (User Datagram Protocol)

• Il livello transport fornisce un protocollo per il trasporto di blocchi di dati non connesso e non affidabile, detto **UDP** (**User Datagram Protocol**), che utilizza l'IP per trasportare messaggi, che è molto simile all'IP in termini di risultato del trasporto, ed offre in più rispetto all'IP la capacità di distinguere tra più destinazioni all'interno di uno stesso host, mediante il meccanismo delle porte.

•Ogni Datagram UDP viene incapsulato in un datagram IP, quindi la dimensione del datagram UDP non può superare la dimensione massima della parte dati del datagram IP. Il datagram IP può essere frammentato se la MTU è piccola.



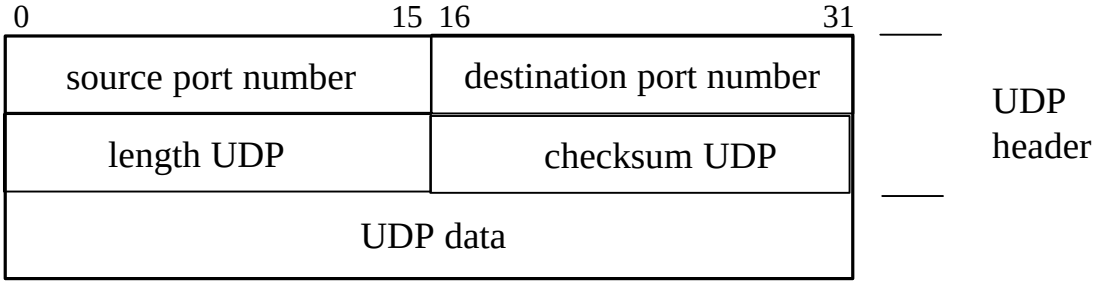
• L'UDP non usa dei riscontri per verificare se un messaggio è arrivato a destinazione, non ordina i messaggi arrivati, e non fornisce nessun tipo di controllo sulla velocità di trasmissione dei dati. Quindi i datagram UDP possono essere persi, duplicati o arrivare fuori ordine.

• Utilizzano UDP alcuni protocolli standard, a cui sono riservati dei numeri di porte predefiniti, in modo da poter essere rintracciati nello stesso punto (punto d'accesso) su tutti gli hosts. Ricordiamo tra gli altri: **nameserver** (server di nomi di dominio, porta 53), **bootps** (server del protocollo di bootstrap, 67), **tftp** (Trivial File Transfer, 69), **ntp** (Network Time protocol, 123).

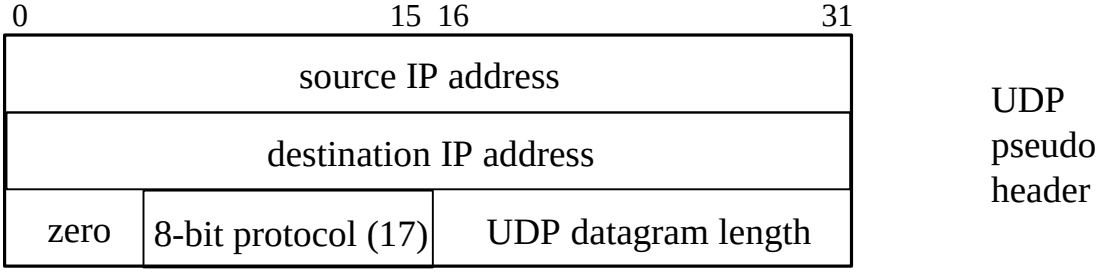
Formato del Datagram UDP

Il pacchetto UDP è costituito da un header e da una parte dati.

L'header UDP è composto da 4 campi:
• i primi due sono i numeri di porta del mittente e del destinatario del datagram, ciascuno di 16 bit.
• il terzo è la lunghezza dei dati del datagram UDP, in byte.
• l'ultimo è un checksum per il controllo d'errore, che però è opzionale. Un valore zero in questo campo indica che la chesksum non è stata calcolata



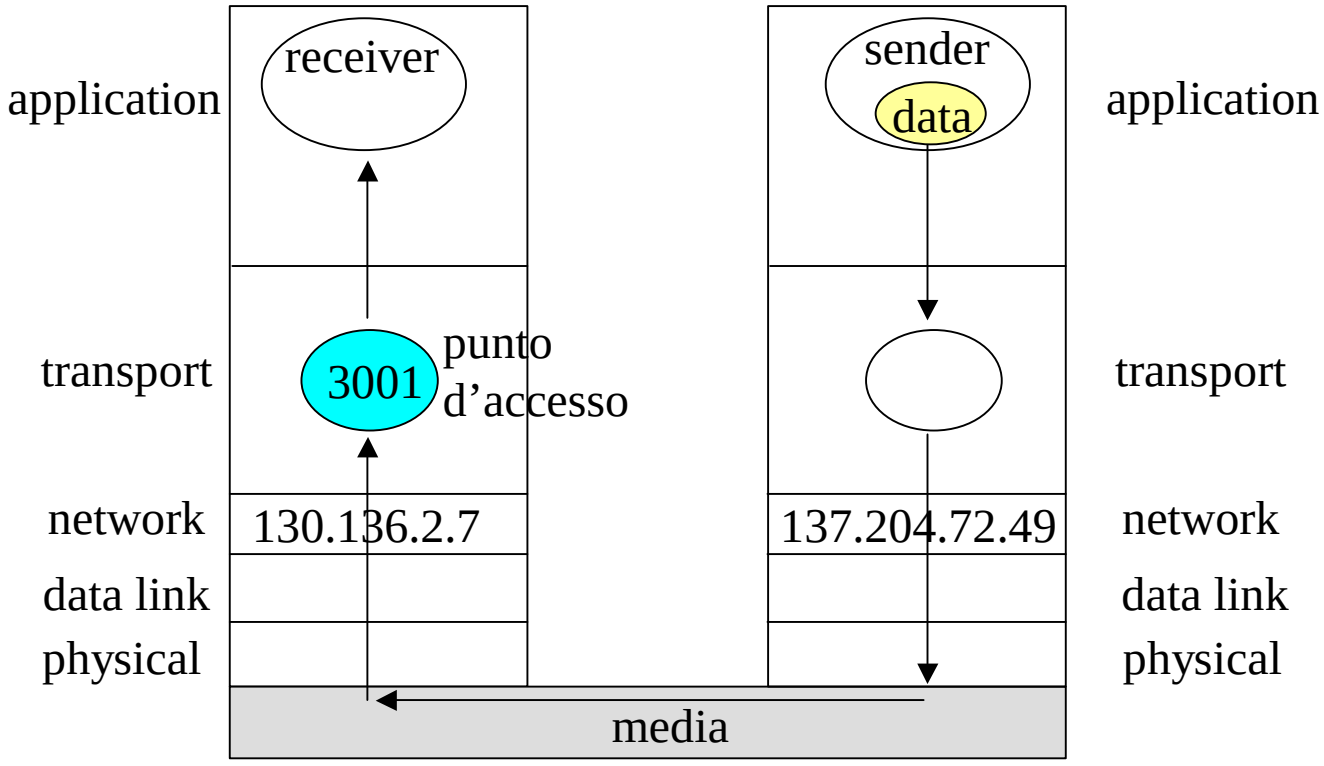
Si noti che non ci sono gli indirizzi IP di mittente e destinatario.
A differenza dell'header IP, il checksum contenuto nell'header UDP considera anche la parte dati UDP.
Inoltre il calcolo della checksum viene effettuato ponendo in testa al datagram UDP una pseudointestazione (che non viene trasmessa), con gli indirizzi IP di provenienza e destinazione ricavata dall'header IP in cui l'UDP viene trasportato, pseudointestazione fatta in questo modo:



Il motivo di questo modo di computare la checksum è verificare a livello UDP, che il datagram UDP abbia raggiunto la corretta destinazione IP, che non compare nell'header UDP.

esempio di trasmissione di datagram UDP

Senza per ora entrare nei dettagli riguardanti i socket, vediamo un semplice esempio di programma che sfrutta i socket per trasmettere un datagram UDP contenente una stringa di testo “pippo” da un host **sender 137.204.72.49** ad un host **receiver 130.136.2.7** .
Il punto di accesso stabilito dal programmatore è nel receiver, nella porta UDP caratterizzata dal numero 3001.
Il receiver si mette in attesa sulla porta **3001** fino a che il sender invia un datagram all’host receiver su quella porta, e stampa il contenuto del datagram ricevuto.
Quindi sender e receiver devono essersi messi d’accordo sulla porta da usare, ed il sender deve conoscere l’indirizzo IP del receiver.



Il codice completo (con la gestione degli errori) dei due programmi che realizzano l’esempio qui mostrato è disponibile allo indirizzo http://www.cs.unibo.it/~ghini/didattica/sistemi3/UDP1/UDP1.html

esempio: receiver di datagram UDP

```
/* eseguito sull'host 130.136.2.7 */
#define SIZEBUF 10000
void main(void)    {
struct sockaddr_in Local, From;   short int local_port_number=3001;
char string_remote_ip_address[100];  short int remote_port_number;
int socketfd, OptVal, msglen, Fromlen;  char msg[SIZEBUF];

/* prende un socket per datagram UDP */
socketfd = socket (AF_INET, SOCK_DGRAM, 0);
/* impedisce l'errore di tipo EADDRINUSE nella bind() */
OptVal = 1;
setsockopt (socketfd, SOL_SOCKET, SO_REUSEADDR,
                (char *)&OptVal, sizeof(OptVal) );
/* assegna l'indirizzo IP locale e una porta UDP locale al socket */
Local.sin_family            =  AF_INET;
Local.sin_addr.s_addr    =  htonl(INADDR_ANY);
Local.sin_port              =  htons(local_port_number);
bind ( socketfd, (struct sockaddr*) &Local, sizeof(Local));

/* wait for datagram */
Fromlen=sizeof(struct sockaddr);
msglen = recvfrom ( socketfd, msg, (int)SIZEBUF, 0,
                              (struct sockaddr*)&From, &Fromlen);
sprintf((char*)string_remote_ip_address,"%s",inet_ntoa(From.sin_addr);

remote_port_number = ntohs(From.sin_port);
printf("ricevuto msg: \"%s\" len %d, from host %s, port %d\n",
        msg, msglen, string_remote_ip_address, remote_port_number);
}
```

esempio: sender di datagram UDP

```
/* eseguito sull'host 137.204.72.49 */
int main(void)   {
struct sockaddr_in Local, To;   char msg[]="pippo";
char string_remote_ip_address[]="130.136.2.7";
short int remote_port_number = 3001;
int socketfd, OptVal, addr_size;

/* prende un socket per datagram UDP */
socketfd = socket(AF_INET, SOCK_DGRAM, 0);
/* impedisce l'errore di tipo EADDRINUSE nella bind() */
OptVal = 1;
setsockopt(socketfd, SOL_SOCKET, SO_REUSEADDR,
                (char *)&OptVal, sizeof(OptVal));
/* assegna l'indirizzo IP locale e una porta UDP locale al socket */
Local.sin_family         =  AF_INET;
/* il socket verra' legato all'indirizzo IP dell'interfaccia che verrà
   usata per inoltrare il datagram IP, e ad una porta a scelta del s.o.  */
Local.sin_addr.s_addr  =  htonl(INADDR_ANY);
Local.sin_port           =  htons(0); /* il s.o. decide la porta locale */
bind( socketfd, (struct sockaddr*) &Local, sizeof(Local));
/* assegna la destinazione */
To.sin_family           =  AF_INET;
To.sin_addr.s_addr  =  inet_addr(string_remote_ip_address);
To.sin_port             =  htons(remote_port_number);
addr_size = sizeof(struct sockaddr_in);
/* send to the address */
sendto(socketfd, msg, strlen(msg) , 0,
          (struct sockaddr*)&To, addr_size);
}
```

Il protocollo di Trasporto TCP (Transmission Control Protocol)

Il protocollo TCP è stato progettato per fornire **un flusso di byte** da sorgente a destinazione **full-duplex, affidabile, su una rete non affidabile**.
Dunque, offre un servizio **reliable** e **connection oriented**, e si occupa di:

- **stabilire la connessione** full duplex tra due punti di accesso a livello trasporto;
- **accettare dati dal livello application** eventualmente **bufferizzando** in input;
- a richiesta, **forzare l'invio interrompendo la bufferizzazione**;
- **spezzare o accorpare** i dati in ***segment***, il nome usato per i TPDU (Transport Protocol Data Unit) aventi dimensione massima 64 Kbyte, ma tipicamente di circa 1.500 byte;
- consegnarli al livello network, per effettuare la trasmissione all'interno di singoli datagram IP, **eventualmente ritrasmettendoli**;
- ricevere segmenti dal livello network;
- **rimetterli in ordine**, eliminando buchi e doppioni;
- **consegnare i dati, in ordine, al livello application**.
- **chiudere la connessione**.

Il servizio effettua internamente la gestione di ack, il controllo del flusso e il **controllo della congestione**.
Il servizio del TCP è di tipo **orientato allo stream**, ovvero **trasporta un flusso di byte**, il che significa che se anche la sorgente spedisce (scrive sul device) i dati a blocchi (es 1 KB poi 3 KB poi ancora 2 KB) la connessione non informa la destinazione su come sono state effettuate individualmente le scritture; la destinazione potrebbe ad es. leggere i dati a blocchi di 20 byte per volta.

Indirizzamento nel TCP (1)

• Come l'UDP, il TCP impiega **numeri di porta di protocollo** per identificare la destinazione finale all'interno di un host.

• La situazione è però molto diversa rispetto all'UDP, in cui possiamo immaginare ogni porta come una sorta di coda a cui arrivano dei datagram delimitati, provenienti ciascuno da un mittente eventualmente diverso.

• **Per il TCP si vuole invece che una connessione sia dedicata in esclusiva ad una coppia di applicazioni risiedenti su macchine diverse.** Il TCP impiega la connessione (virtuale), non la porta di protocollo, come sua astrazione fondamentale;

• le connessioni sono identificate da una coppia di punti d'accesso (endpoint) detti socket, uno su ciascuna macchina.

• Ogni socket è caratterizzato da una coppia (**IP address: Port number**) che può essere utilizzata da più processi simultaneamente. Questa è ad esempio la situazione prodotta dal processo demone del telnet (telnetd) che consente agli utenti di una macchina unix A di collegarsi ad A da un'altro host accedendo tutti alla stessa porta TCP numero 23 di quell'host A. Tutte le connessioni condividono quindi la stessa coppia (IP_A, 23).

• Però **ciascuna connessione viene univocamente individuata dalla coppia di socket dei due host A e B implicati nella connessione, ovvero dalla terna:**
(**IP address A: Port number A** , **IP address B: Port number B**)

Più utenti che effettuano tutti il telnet da una stessa macchina B verso una stessa macchina A instaurano connessioni identificate da terne del tipo (IP_A : 23 , IP_B : **tcp_port_B**) con **tcp_port_B** tutti diversi, e quindi con connessioni univocamente determinate.

Well-Know-Port nel TCP

Come per il caso UDP, anche per il TCP esistono dei port number che sono riservati ad uso di protocolli standard, e vengono detti well-know-port. Queste porte sono quelle con valore inferiore a 256.

Port Number	Service
20	Ftp (control)
21	Ftp (data)
23	Telnet
25	Smtp
80	Http

I segmenti TCP

• L’unità di trasferimento del TCP è detta **segmento**, e viene usato per stabilire connessioni, per trasferire dati, per inviare riscontri (una sorta di ricevuta di ritorno), per dimensionare le finestre scorrevoli e per chiudere le connessioni.

• TCP usa un meccanismo di sliding window (finestre scorrevoli) di tipo go-back-n con timeout. Se questo scade, il segmento si ritrasmette. Si noti che le dimensioni della finestra scorrevole e i valori degli ack sono espressi in numero di byte, non in numero di segmenti.

• ogni byte del flusso TCP è numerato con un numero d'ordine a 32 bit, usato sia per il controllo di flusso che per la gestione degli ack;

• ogni segmento TCP non può superare i 65.535 byte, e **viene incluso in un singolo datagram IP**;

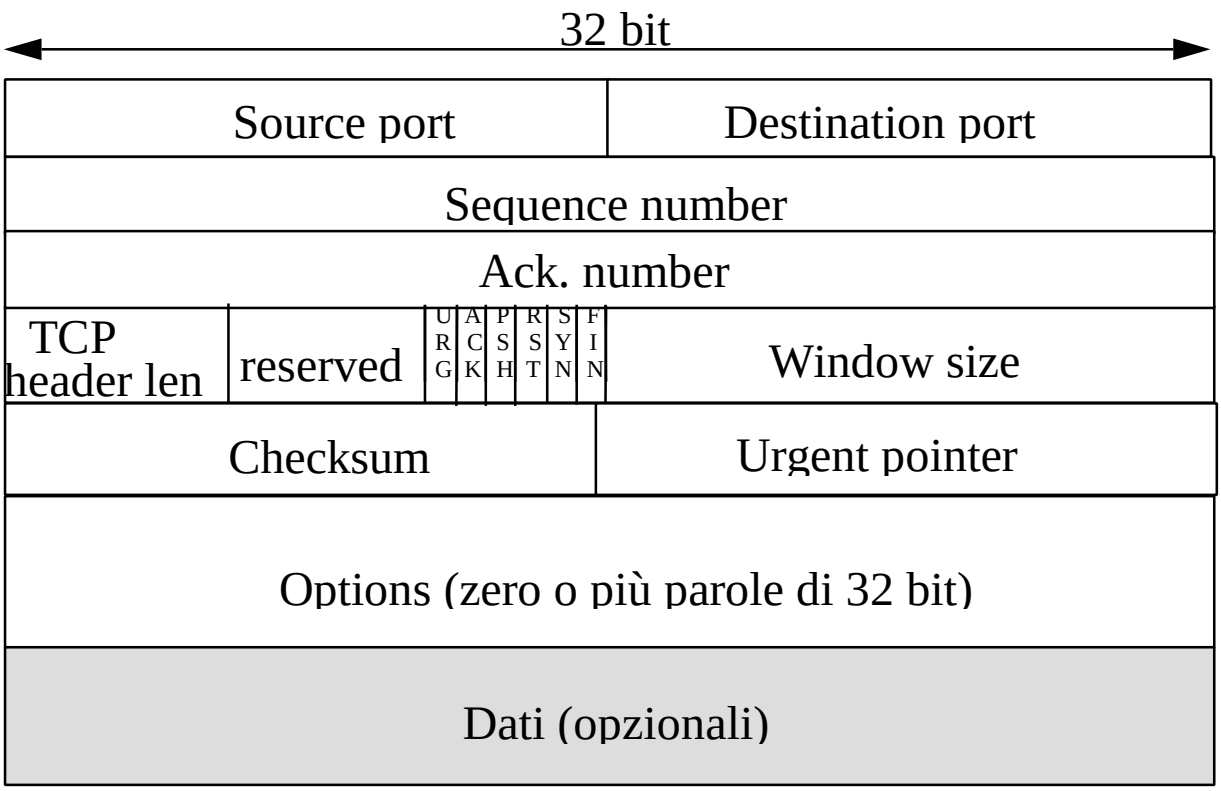
• un segmento TCP è formato da:

uno **header**, a sua volta costituito da:

- una parte fissa di 20 byte;
- una parte opzionale;

i **dati** da trasportare;

Formato del segmento TCP



Source port, destination port: identificano gli end point (locali ai due host) della connessione. Essi, assieme ai corrispondenti numeri IP, identificano la connessione a cui appartiene il segmento;

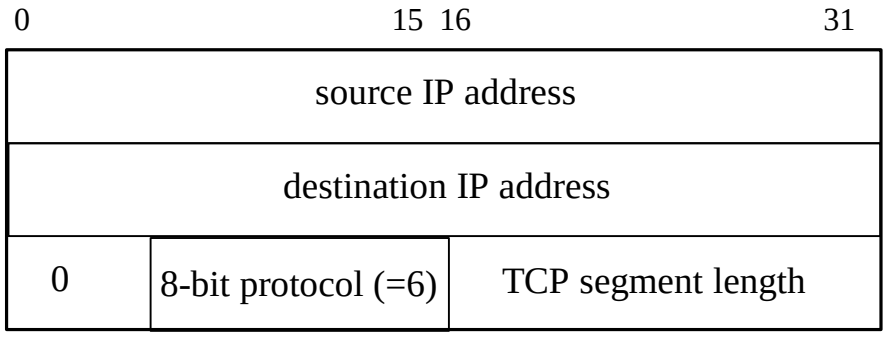
Sequence number: la posizione del primo byte contenuto nel campo dati **all'interno dello stream di byte che il trasmettitore del segmento invia** (si possono inviare quindi al max 4 miliardi di byte circa in uno stesso stream).

Ack. number: la posizione del prossimo byte aspettato **all'interno del segmento inviato dal ricevitore del presente segmento**.

TCP header length: lunghezza del segmento misurata in parole di 32 bit (necessario perché il campo options ha dimensione variabile).

Formato del segmento TCP (2)

Checksum, simile a quello di UDP, verifica che sia l'header che i dati del segmento siano arrivati a destinazione senza errori. Come per UDP il calcolo del checksum prevede che sia aggiunto in testa al segmento uno pseudoheader contenente gli indirizzi IP di sorgente e destinazione, la lunghezza del segmento, estratti dall'header del datagram IP che contiene il segmento.



TCP pseudo header

Nell'header del segmento TCP sono presenti inoltre 6 **flags** di un bit ciascuno che servono per assegnare validità ad alcuni campi dell'header o per segnalare richieste o conferme:

URG 1 se il campo urgent pointer è usato, 0 altrimenti.
ACK 1 se l'ack number è valido (cioè se si trasporta un ack), 0 altrimenti.
PSH indica che questo segmento contiene dati urgenti (**pushed data**), da consegnare senza aspettare che il buffer si riempia.
RST richiesta di reset della connessione (ci sono problemi!).
SYN usato nella fase di setup della connessione:
SYN=1 ACK=0 richiesta connessione;
SYN=1 ACK=1 accettata connessione.
FIN usato per rilasciare una connessione.

Formato del segmento TCP (3)

altri campi presenti nel segmento TCP sono:

Window size: il controllo di flusso è di tipo sliding window di dimensione variabile. **Window size dice quanti byte possono essere spediti a partire da quello (compreso) che viene confermato con l'ack number**. Un valore zero significa: fermati per un pò, riprenderai quando ti arriverà un uguale ack number con un valore di window size diverso da zero. Questo valore serve a ridurre la velocità di trasmissione dei dati nel flusso di byte che va dal ricevente al trasmettitore del presente segmento. Viene usato quando ci si accorge di una congestione della rete, per non incrementare ancora la congestione, oppure quando i buffer di ricezione sono ormai pieni e non si vuole rischiare di perdere dei dati .

Urgent pointer puntatore ai dati urgenti, indica la posizione in cui terminano i dati urgenti nello stream inviati dal trasmettitore.

Options contiene alcune opzioni appliccabili al flusso. Le più importante sono negoziabili durante il setup della connessione, e sono:

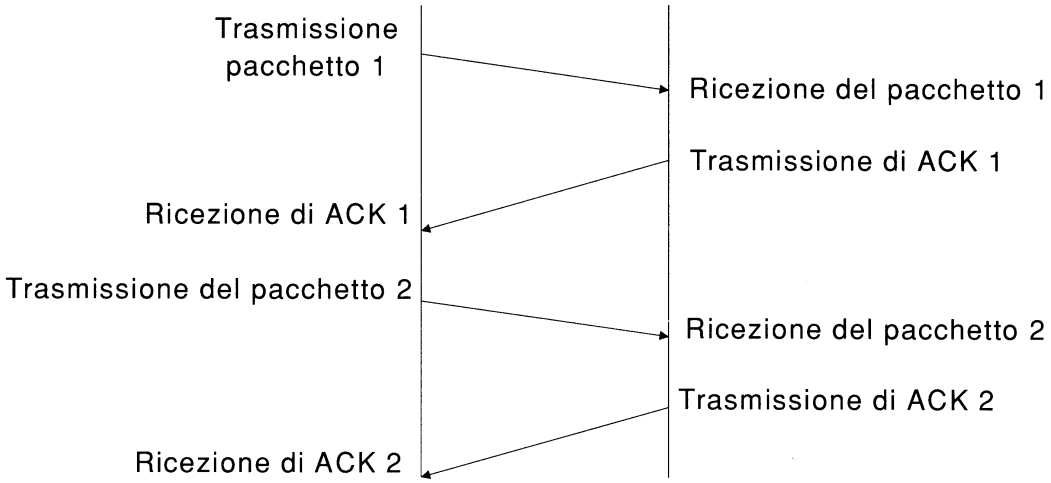
• dimensione massima dei segmenti da spedire (**MSS: Maximum Segment Size**), serve se uno degli host ha dei buffer molto limitati ma soprattutto per adattare il segmento alla MTU della rete che collega i due host, in modo che **ogni segmento venga incluso in un datagram IP che non debba essere frammentato**;

• uso di selective repeat invece che go-back-n (un diverso algoritmo di controllo del flusso);

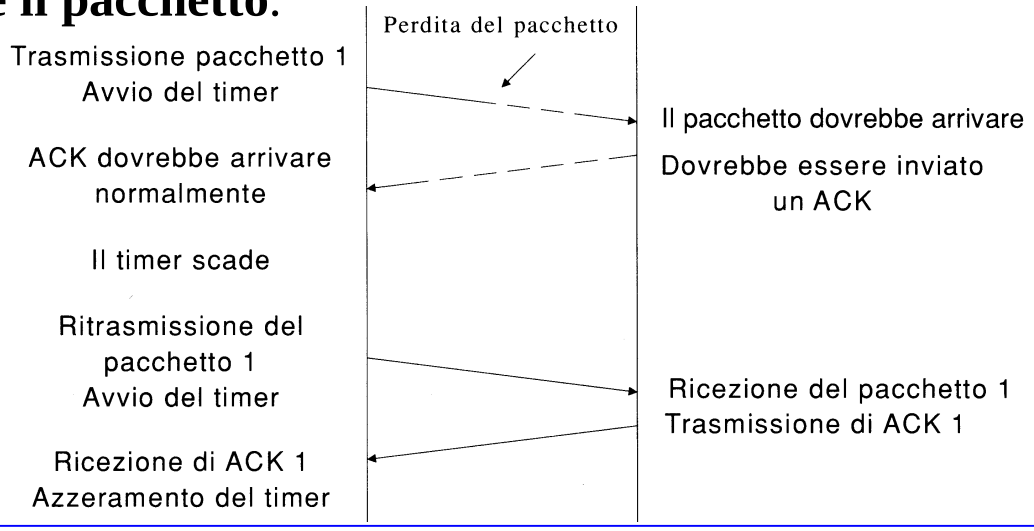
• uso di NAK.

Trasmissione affidabile: riscontro positivo con ritrasmissione (1)

L'affidabilità del TCP si basa sulla combinazione di alcune tecniche che adesso analizzeremo separatamente. La tecnica più semplice per garantire che tutti i byte trasmessi in un flusso giungano a destinazione è nota come **riscontro positivo con ritrasmissione**. **Quando un ricevitore riceve un pacchetto risponde al pacchetto inviando un riscontro (acknowledgment, ACK) al trasmettitore per confermare di averlo ricevuto.**



Quando il trasmettitore deve inviare un pacchetto, nel momento in cui lo spedisce ne mantiene una copia e fa partire un timer. Se allo scadere del timer non ha ancora ricevuto il relativo ACK ritrasmette il pacchetto.



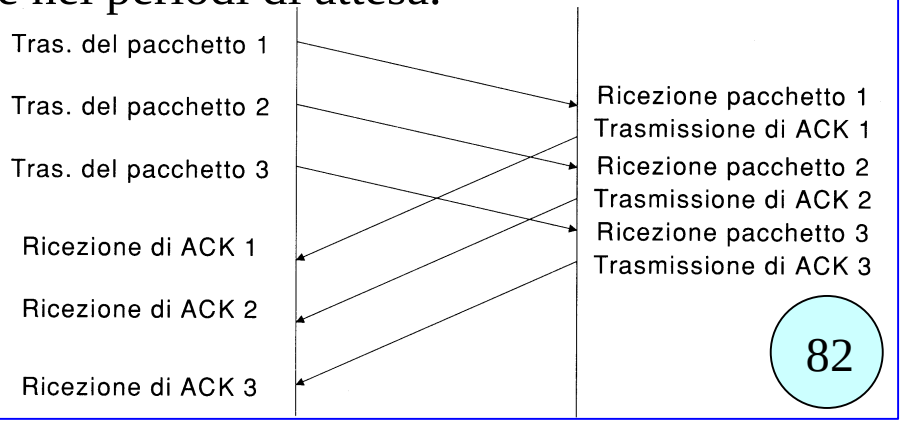
Trasmissione affidabile: riscontro positivo con ritrasmissione (2)

Poichè può capitare che la rete duplichi un pacchetto, può capitare di ricevere un ACK per un pacchetto che era già stato riscontrato in precedenza, e di scambiarlo per un riscontro di un successivo pacchetto per cui attendevamo un ACK.
Per ovviare al problema dei riscontri duplicati, i pacchetti vengono numerati sequenzialmente, e l'ACK contiene il numero sequenziale del pacchetto che vuole riscontrare.

Le finestre Scorrevoli (1)

• Il meccanismo di riscontro positivo rallenta la trasmissione, perchè **il trasmettitore deve attendere il riscontro di ciascun pacchetto prima di inviare il successivo**, quindi i pacchetti viaggiano in rete in una sola direzione per volta, e la rete è sottoutilizzata mentre gli end system attendono le risposte, che possono ritardare.
• La tecnica delle finestre scorrevoli (**Sliding Windows**) invece permette al **trasmettitore di poter continuare ad inviare un certo numero N (dimensione della finestra) di pacchetti successivi all'ultimo per cui ha ricevuto il riscontro**, ovvero permette di trasmettere fino ad altri N pacchetti mentre si è in attesa di ricevere il riscontro di un pacchetto precedentemente inviato. In tal modo la rete viene utilizzata anche nei periodi di attesa.

In figura vengono trasmessi 3 pacchetti prima di ricevere un riscontro.

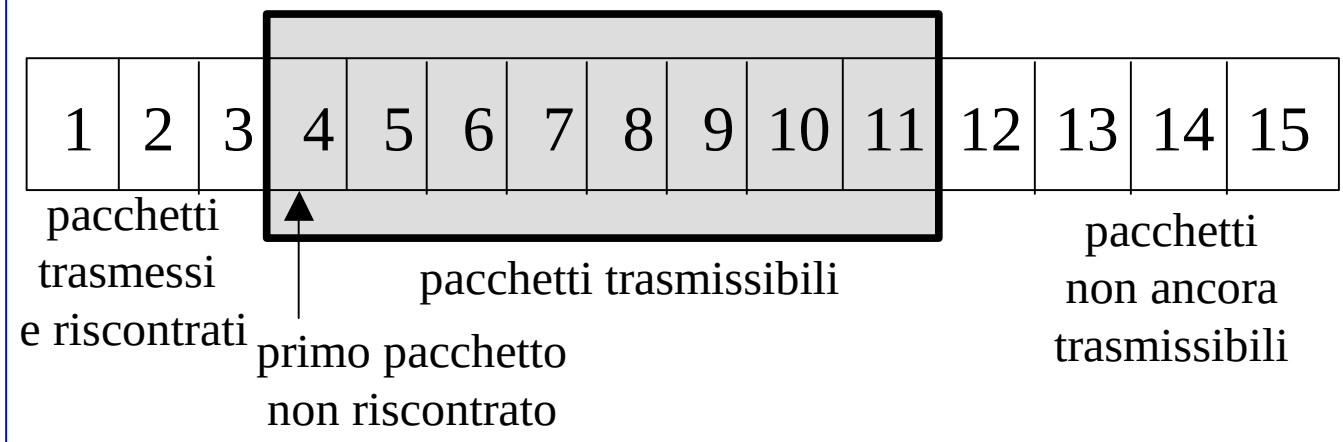


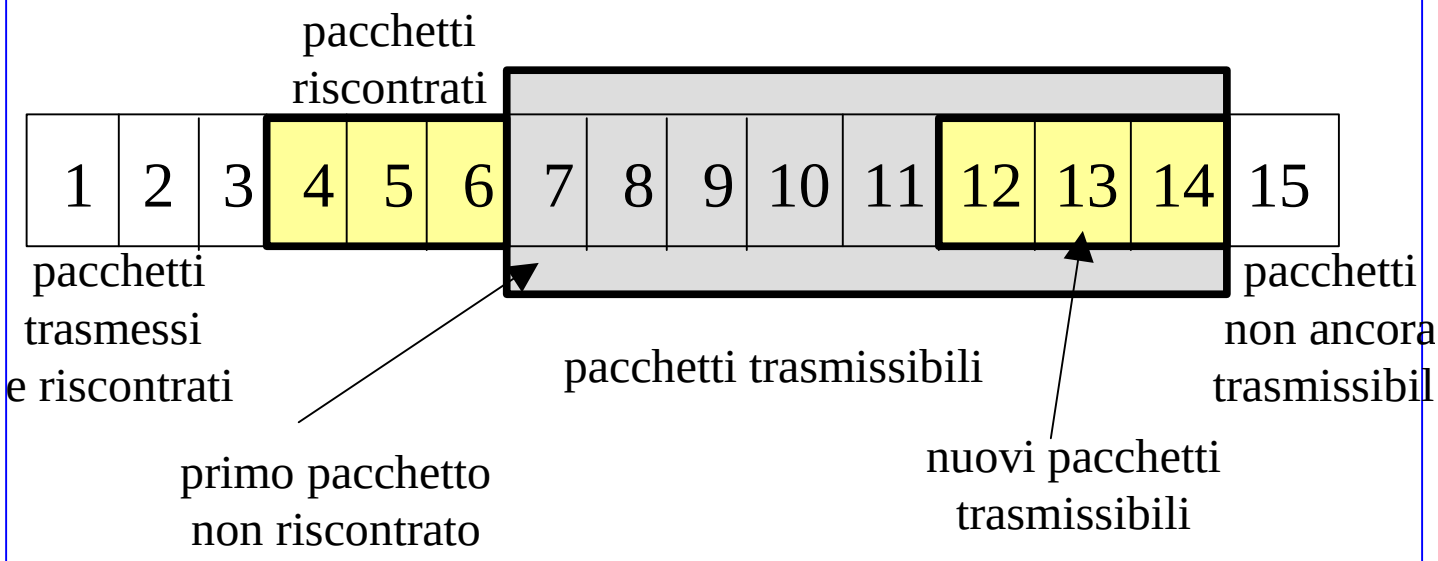
Le finestre Scorrevoli (2)

• definiamo **non riscontrato** un pacchetto trasmesso per il quale non è ancora stato ricevuto il riscontro.

• La finestra **del trasmettitore** inizia col primo pacchetto non riscontrato. I pacchetti successivi, che stanno dentro alla finestra, possono essere trasmessi, mentre i pacchetti che seguono la finestra non possono essere trasmessi.

• Quando un pacchetto P (trasmesso) nella finestra viene riscontrato (il pacchetto 6 in figura) (e sono riscontrati anche i suoi precedenti) allora la finestra avanza fino al pacchetto successivo a P, consentendo di trasmettere i pacchetti che sono entrati nella finestra.





(in figura la dimensione della finestra è 8)

Le finestre Scorrevoli (3)

• le finestre **continuano a scorrere finchè si ricevono riscontri**, permettendo di **trasmettere i nuovi pacchetti che entrano nella finestra**.

• Le prestazioni dei protocolli a finestra scorrevole dipendono dalla dimensione della finestra e dalla velocità della rete.

• se la dimensione della finestra è 1, si ritorna all'algoritmo di riscontro positivo.

• all'aumentare della dimensione della finestra diminuisce il periodo di non utilizzo della rete.

• con un opportuna scelta della dimensione della finestra è possibile mantenere la rete satura di pacchetti senza congestionarla.

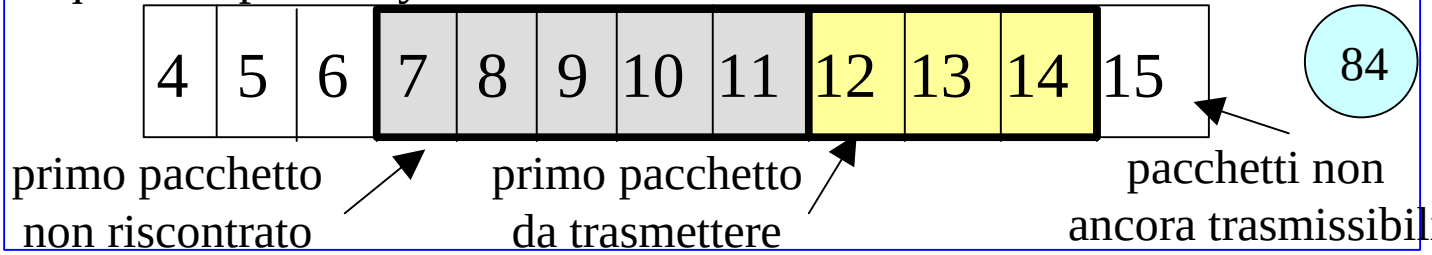
• Per ogni pacchetto trasmesso viene comunque ancora fatto partire **un timer**, allo scadere del quale, se non è ancora stato ricevuto il riscontro, **il pacchetto deve essere ritrasmesso**.

• Il meccanismo delle finestre scorrevoli è stato spiegato riferendosi alla trasmissione di pacchetti, ma vale ugualmente se:

1) consideriamo lo **stesso meccanismo** (la finestra) replicato **nei due end system**, perchè essendo la connessione TCP **bidirezionale**, ogni host funge da trasmettitore;

2) al posto dei pacchetti **consideriamo i byte del flusso** realizzato dalla connessione TCP, e utilizziamo la posizione dei byte nel flusso come indicatore per sapere nel trasmettitore:

- qual'è il primo byte della finestra (da riscontrare)
- qual'è il primo byte che segue la finestra (da non trasmettere)
- qual'è il primo byte ancora da trasmettere dentro la finestra.



I Riscontri (1)

• Poichè il TCP invia i dati in segmenti di lunghezza variabile, e poichè i segmenti ritrasmessi possono includere più dati dell'originale, **i riscontri** non possono fare riferimento facilmente ai datagrammi o ai segmenti, ma **fanno riferimento alla posizione dei byte nello stream**.

• Il ricevitore raccoglie i byte di dati dei segmenti in arrivo e li riordina, ma poichè i segmenti viaggiano in datagram IP possono essere persi o consegnati disordinatamente, generando dei buchi nella ricostruzione.

• **Il ricevitore riscontra solo i byte che precedono il primo buco**, o meglio, **specifica il valore sequenziale (la posizione) del primo byte che ancora gli manca e che si aspetta di ricevere**, il primo byte del primo buco, anche se ha già ricevuto qualcosa di successivo.

• Il vantaggio è che questi riscontri cumulativi sono facili da generare, e che la perdita di un riscontro non costringe alla ritrasmissione se arriva un riscontro per un byte successivo.

• Uno svantaggio è che il trasmettitore, se un riscontro per un byte non arriva, non è in grado di capire se manca solo un pezzetto (un buco anche di quel solo byte) o manca tutto da quel byte in avanti, e ricomincia a trasmettere tutto da quel byte.