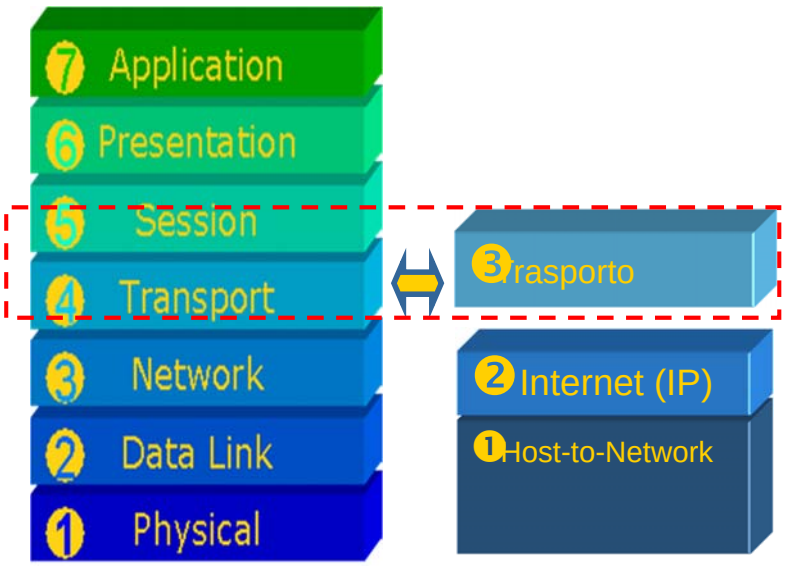
RETI DI CALCOLATORI

A.A. 2013-2014

I Protocolli TCP e UDP

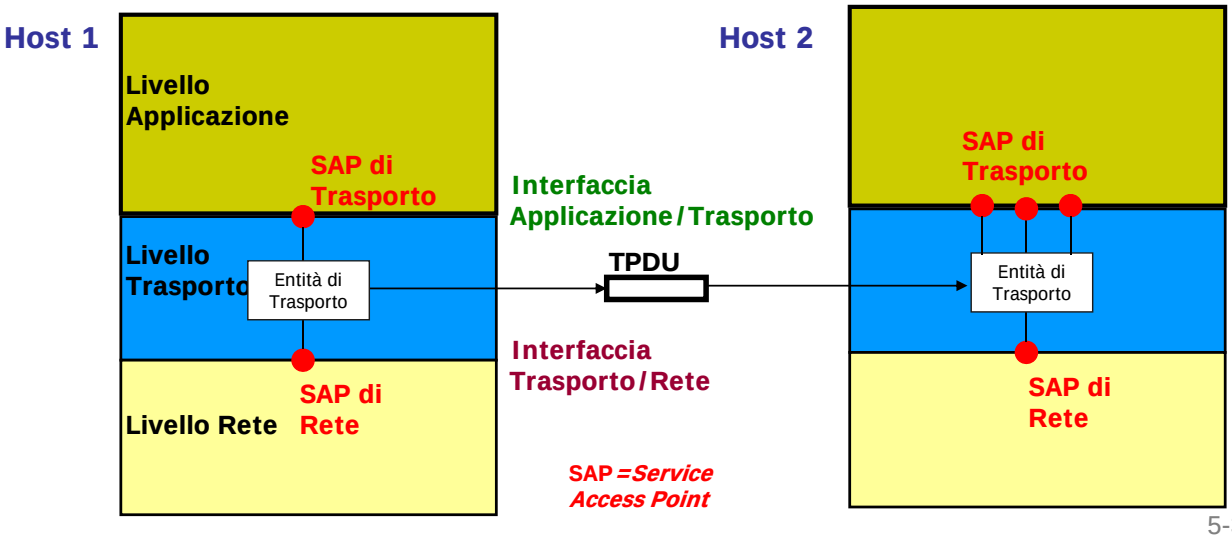
Carlo Mastroianni

Livello TRASPORTO



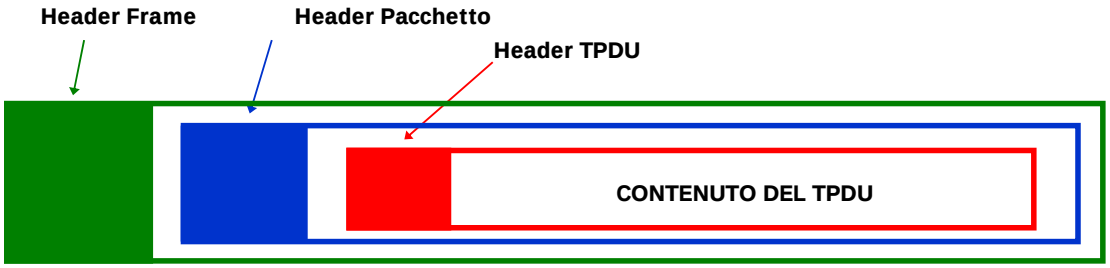
Servizi di Trasporto

- Il software che fornisce i servizi di trasporto è detto entità di trasporto.
- Le entità di trasporto sono eseguite sui computer degli utenti, ed usano i servizi, per lo più forniti dai router, del livello di rete.



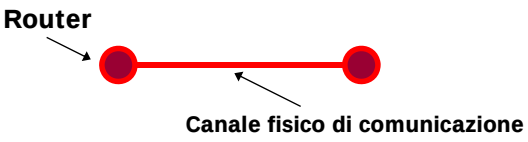
TPDU

- La **TPDU** (**Transport Protocol Data Unit**), spesso denominata anche **segmento**, è l'unità di dati scambiata dal protocollo di trasporto.
- La TPDU viene inserita nella parte dati dei pacchetti dei livelli inferiori (rete e data link).

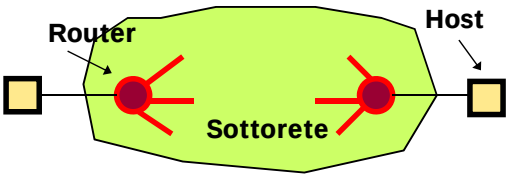


Protocolli di Trasporto

- Sono protocolli **end-to-end**
- Gestiscono l'**apertura** e **chiusura** delle connessioni, **l'ordinamento dei dati**, il **multiplexing/demultiplexing**, il **controllo degli errori**, il **controllo di flusso**, il **controllo della congestione**.
- Molte di queste problematiche sono affrontate anche al livello data link, ma nel livello di trasporto la situazione da gestire è più complessa, perché gli host non sono connessi non direttamente connessi, ma attraverso la rete



Livello Data Link



Livello Trasporto

Protocolli Trasporto di Internet: TCP e UDP

Protocolli di trasporto definiti su rete Internet (IP)

- **Trasmission Control Protocol** (**TCP**) definisce un protocollo di trasporto orientato alla connessione
  - progettato per fornire un flusso affidabile end-to-end su una rete best effort (IP)
- **User Data Protocol** (**UDP**) definisce un protocollo senza connessione
  - permette di inviare datagrammi IP senza stabilire una connessione
  - si usa per connessioni real-time, multicast, e a volte per comunicazioni che prevedono solo una richiesta ed una risposta (es. il DNS).

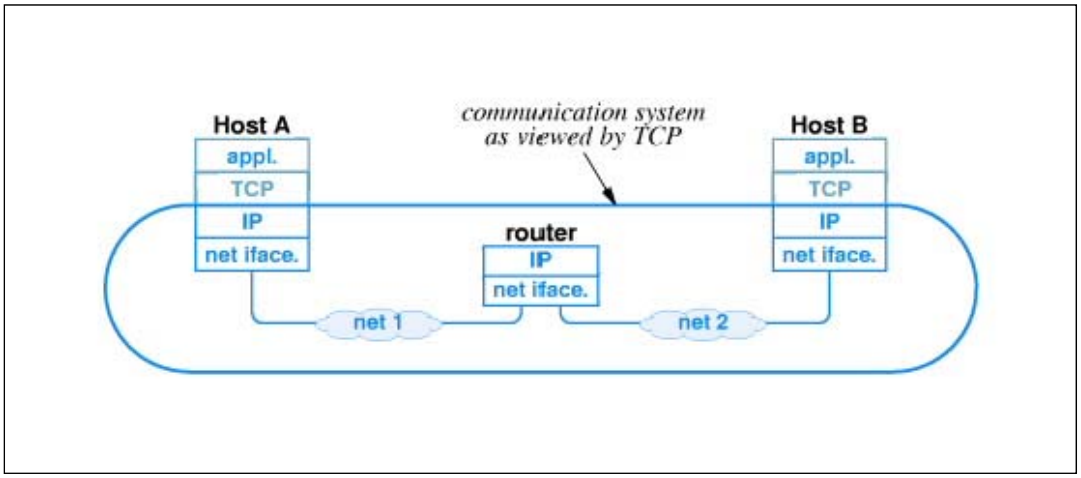
Uso di TCP e di UDP

Applicazioni di internet e relativi protocolli.

Applicazione	Protocollo dello strato dell'applicazione	Protocollo di trasporto adottato
Posta elettronica	SMTP	TCP
Accesso a terminale remoto	Telnet	TCP
Web	HTTP	TCP
Trasferimento di file	FTP	TCP
Server di file remoto	NFS	tipicamente UDP
Streaming multimediale	proprietario	tipicamente UDP
Telefonia Internet	proprietario	tipicamente UDP
Gestione della rete	SNMP	tipicamente UDP
Protocollo di routing	RIP	tipicamente UDP
Traduzione del nome	DNS	tipicamente UDP

TCP su IP

L'entità **TCP** di un computer usa **IP** per comunicare con l'entità **TCP** di un altro computer.



Caratteristiche di TCP

- **Orientamento alla connessione** (quindi consegna ordinata dei dati)
- Comunicazione punto-punto (non gestisce il multicast)
- Comunicazione full-duplex
- Multiplexing/Demultiplexing
- Totale affidabilità (senza perdite, duplicazioni, inversioni di dati)
- Trasmissione di flussi di byte (informazioni non strutturate)
- Instaurazione affidabile della connessione (sulla rete potrebbero viaggiare richieste di connessione duplicate, che non devono essere considerate)
- Rimozione non traumatica della connessione (consegna di tutti i dati inviati prima della chiusura della connessione)
- Controllo di flusso (il trasmettitore non deve sommergere il ricevitore)
- Controllo della congestione

Funzionamento di base di TCP

Entità TCP in trasmissione

- Riceve un flusso di dati dall'applicazione
- Li organizza in **segmenti** e li consegna ad IP per immetterli sulla rete
- Tuttavia è il flusso ordinato di byte che conta, non quello dei segmenti
- I segmenti sono lunghi al massimo 64 KB, ma in genere sono lunghi non più di 1460 byte, per farli entrare in un frame Ethernet del livello di collegamento dati
- Ritrasmette i dati per i quali non riceve l'ack dalla controparte

Entità TCP in ricezione

- Estrae i segmenti dai pacchetti IP consegnati dal livello di rete
- Se i dati sono ricevuti correttamente, invia un ack alla controparte
- Ricostruisce il flusso di byte originale nella sequenza corretta
- Consegna i dati ordinati all'applicazione

TCP: numeri di sequenza

- TCP deve garantire la consegna dei dati ed il loro corretto ordinamento
- A tal scopo, i **byte** inviati sono ordinati tramite **numeri di sequenza**
- Attenzione: i numeri di sequenza non riguardano i segmenti, ma i singoli byte: il numero di sequenza inserito in un segmento è quello del primo byte contenuto nel segmento, gli altri byte dello stesso segmento sono numerati implicitamente in maniera consecutiva
- Dopo aver inviato un segmento di dati, il trasmettitore attende un **ack** da parte del ricevitore.

TCP: ack e ritrasmissioni

- La ricezione di un ack avente un dato numero di sequenza vale come *ricevuta* per tutti i byte, fino a quel numero di sequenza, trasmessi in precedenza
- Viene utilizzato un **timer** per limitare l'attesa dell'ack: se il timer scade, il trasmettitore provvede alla ritrasmissione dei byte per cui non è giunto l'ack
- Il trasmettitore deve mantenere in un buffer i byte che potrebbe dover ritrasmettere, cioè i byte per cui non è ancora giunto l'ack
- Il ricevitore ha il compito di riordinare i dati; per poter accettare dati non in ordine (e riordinarli in seguito) anch'esso mantiene un apposito buffer

I Service Access Point di TCP: le Porte

- Per connettersi ad un servizio specifico su un server si deve conoscere il **numero di porta** su cui il processo server accetta le connessioni
- Le porte da 0 a 1023 sono dette **porte ben note** (*well-known ports*) e corrispondono a servizi standard
- Ad esempio:
  - la **porta 21** di TCP corrisponde ad **FTP** (File Transfer Protocol).
  - la **porta 80** di TCP corrisponde ad **HTTP** (HyperText Transfer Protocol) utilizzato per il Web
- Per la lista completa **http://www.iana.org/assignments/port-numbers**
  - In Unix/Linux la lista dei servizi e delle porte è nel file **/etc/services**
  - In Windows è nel file **C:\windows\system32\drivers\etc\services**

Porte del Server e del Client

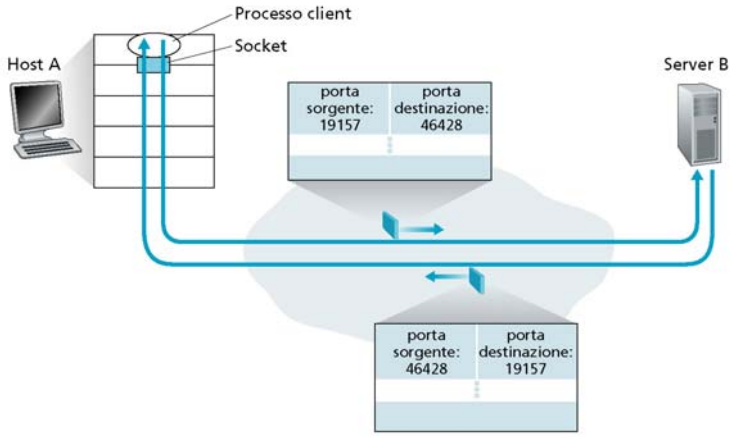
- La **porta del server** è definita dal tipo di applicazione che si vuole utilizzare (Web, FTP, e-mail ecc.)
- Il **client** invece utilizza in genere numeri di porta elevati e scelti in modo da essere unici sull'host.
- Ad esempio nella richiesta di connessione ad un server Web si può avere:

client port = 18426
server port = 80

- Le connessioni TCP sono punto-a-punto e full duplex: il server può rispondere inviando dati sulla porta comunicatagli dal client.

Connessione full duplex: inversione delle porte

- Il processo server utilizza la porta del client per inviare a sua volta dei dati al client
- I dati di ritorno possono essere utili alle applicazioni, o allo stesso livello di trasporto (ack, dim. della finestra di ricezione ecc.)



Socket

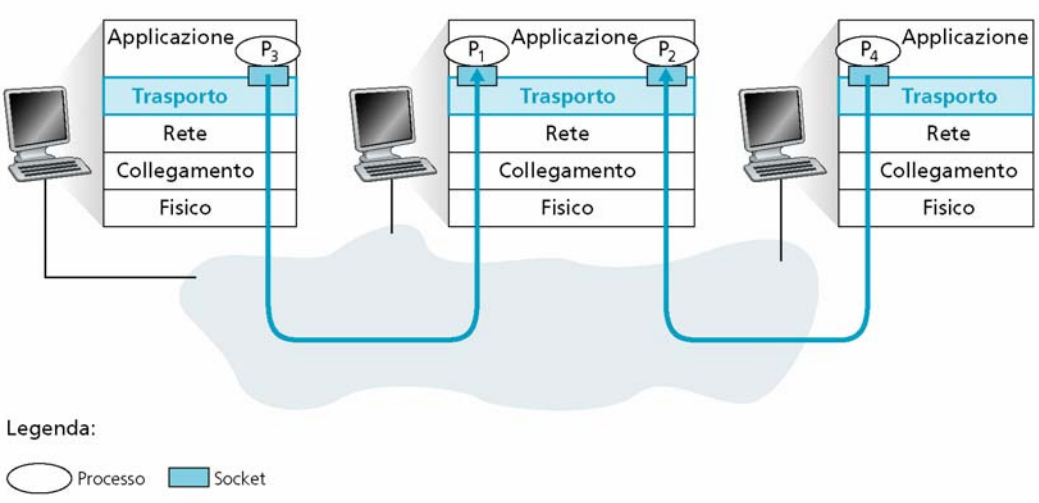
- Il concetto di **socket** è stato introdotto su UNIX BSD (Berkeley).
- Il socket definisce **l'end-point** di una connessione TCP
- Ogni socket è caratterizzato da un indirizzo composto dall'**indirizzo IP** dell'host e da un numero locale a 16 bit (**porta**)
- Per ottenere un servizio TCP si deve creare esplicitamente una connessione fra un socket della macchina richiedente (**client**) ed un socket della macchina ricevente (**server**)
- Una connessione è identificata tramite gli identificatori dei socket sui due lati
- Una volta attivato, un socket è utilizzato come un file: la **send** corrisponde ad una scrittura su uno stream in uscita, la **receive** ad una lettura da uno stream in ingresso

Multiplexing/Demultiplexing

- I **socket** sono utilizzati come intermediari tra gli host ed i processi del livello applicativo
- Il livello di trasporto, tramite l'identificativo del socket, è in grado di smistare un segmento in arrivo verso il corretto processo del livello applicativo
- **Multiplexing**. E' l'operazione, in fase di trasmissione, tramite cui l'entità di trasporto riceve i dati da un processo del livello applicativo, costruisce un segmento, e passa il segmento ad IP per l'invio sulla rete
- **Demultiplexing**. E' l'operazione, in fase di ricezione, tramite cui l'entità di trasporto recapita i dati al corretto processo applicativo

Multiplexing/Demultiplexing

- Esempio: l'host al centro della figura effettua il demultiplexing di due connessioni su due diversi processi, tramite i rispettivi socket.



Primitive Socket

Sono utilizzate dai programmi per aprire e chiudere connessioni, e per inviare i dati. Sotto: **Primitive Socket di Berkeley UNIX**

Primitiva	Significato	Uso sul
SOCKET	Crea un socket e restituisce il suo descrittore	C / S
BIND	Assegna una porta locale al socket	Server
LISTEN	Il socket si mette in attesa di richieste di connessione; a tal scopo gestisce una coda	Server
ACCEPT	Se la coda è vuota, attende una richiesta di connessione; altrimenti gestisce la prima richiesta in coda	Server
CONNECT	Richiede una connessione alla controparte	Client
SEND	Invia dati alla controparte	C / S
RECEIVE	Riceve dati dalla controparte	C / S
CLOSE	Rilascia la connessione	C / S

Sequenze di primitive socket

Client
- Socket
- Connect
- (Send / Receive)*
- Close

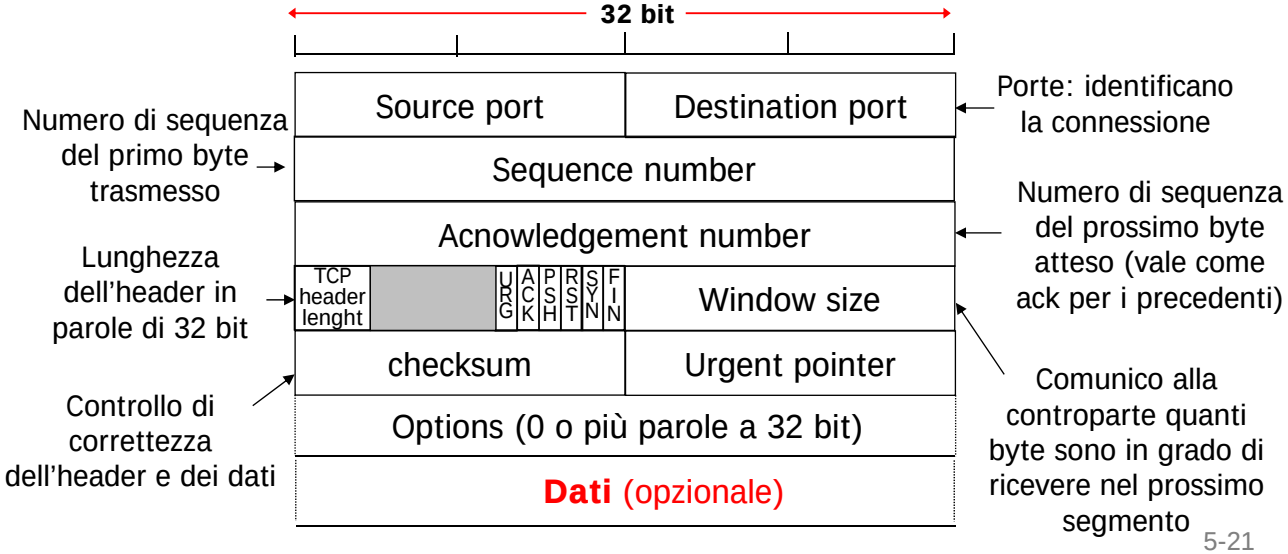
Server semplice
- Socket
- Bind
- Listen
- Accept
- (Send / Receive)*
- Close

Server "multiplo"
- Socket
- Bind
- Listen
- Accept
- Fork
  - (Send / Receive)*
  - Close
  - Exit

Attenzione: i metodi java delle classi Socket e ServerSocket sono definiti ad un livello di astrazione leggermente superiore

Il Segmento TCP

- Ogni **segmento TCP** ha un header fisso di 20 byte (più eventuali dati opzionali) seguito da 0 o più byte di dati.
- dim max della parte dati = 65535 - 20 (header TCP) – 20 (header IP) bytes



I Flag TCP

- Nel segmento TCP sono presenti 6 bit di flag

URG

Se il bit URG è 1, il campo **Urgent Pointer** indica la posizione, a partire dal numero di sequenza attuale, dei **dati urgenti** (es. in una interfaccia grafica pressione di CTRL-C per interrompere il programma remoto)

ACK

Indica se il campo **Acknowledgement number** è valido. E' sempre settato ad 1, eccetto il primo segmento di una connessione

PSH

Indica dati di tipo **PUSH**, ovvero si richiede al ricevitore di consegnare subito i dati senza bufferizzarli

RST

Richiesta di re-inizializzazione di una connessione diventata instabile. Viene anche usato per rifiutare un segmento non valido o la richiesta di apertura di una connessione

Flag SYN e FIN

SYN

Viene utilizzato per creare connessioni. La richiesta di connessione contiene SYN=1 e ACK=0. La risposta ha SYN=1 e ACK=1.

Tutti gli altri segmenti hanno SYN=0, e ACK=1.

FIN

Viene utilizzato per chiudere una connessione (il mittente non ha altri dati da spedire).

La richiesta di chiusura connessione ha FIN=1.

Le opzioni dell'intestazione

Ce ne sono due importanti:

1. Maximum TCP payload

Specifica la grandezza massima del segmento (in realtà della sola parte dati, esclusa l'intestazione) che un host può accettare.

Entrambi gli host possono inviare questo dato all'apertura della connessione: la dimensione massima del segmento per la connessione è settata al minimo dei due valori.

Ogni host è comunque obbligato ad accettare segmenti con payload di almeno 536 byte (almeno 556 compresa l'intestazione).

2. Window scale option

Se si utilizza questa opzione, il campo *window size*, di 16 bit, viene fatto scorrere logicamente di 14 bit verso sinistra, permettendo così una finestra di 2^{30} byte.

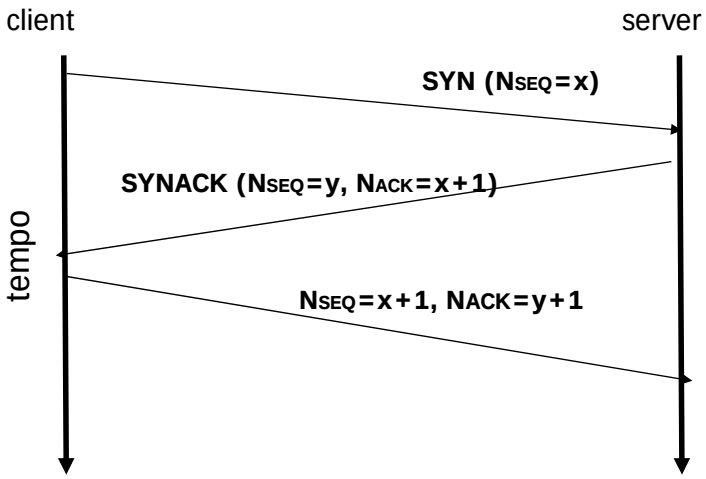
In questo modo il ricevitore può comunicare al trasmettitore una dimensione del buffer fino a 1 GB, invece che fino a 64 KB.

Apertura della Connessione

- Si utilizza un protocollo **3-way handshake**. La connessione è stabilita quando entrambe la parti sono certe che l'altra parte è d'accordo.
- La richiesta di connessione del client (segmento **SYN**) contiene:
  - Bit SYN=1 e bit ACK=0
  - Numero di sequenza = x
- Se il server autorizza la connessione, la risposta (segmento **SYNACK**), contiene:
  - Bit SYN=1 e bit ACK=1
  - Numero di sequenza = y; numero di ack = x+1.
- Il client deve rispondere con un ulteriore segmento con:
  - Bit SYN=0 e bit ACK=1
  - Numero di sequenza = x+1; numero di ack = y+1.

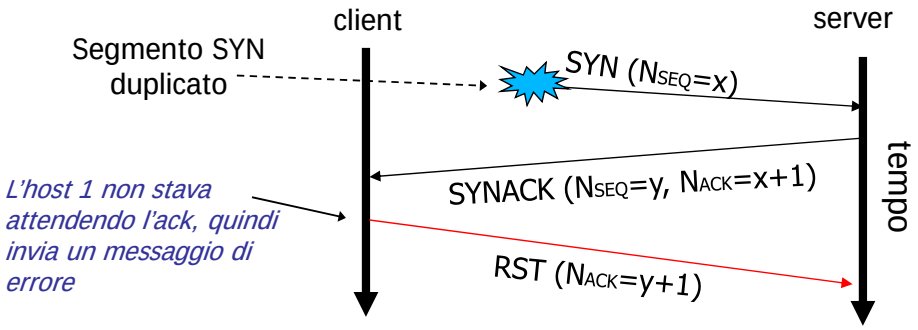
Protocollo di handshake a 3 vie

- I numeri di sequenza iniziali (**x** ed **y**) sono determinati dai due host indipendentemente, utilizzando un orologio locale con clock=4 μs.



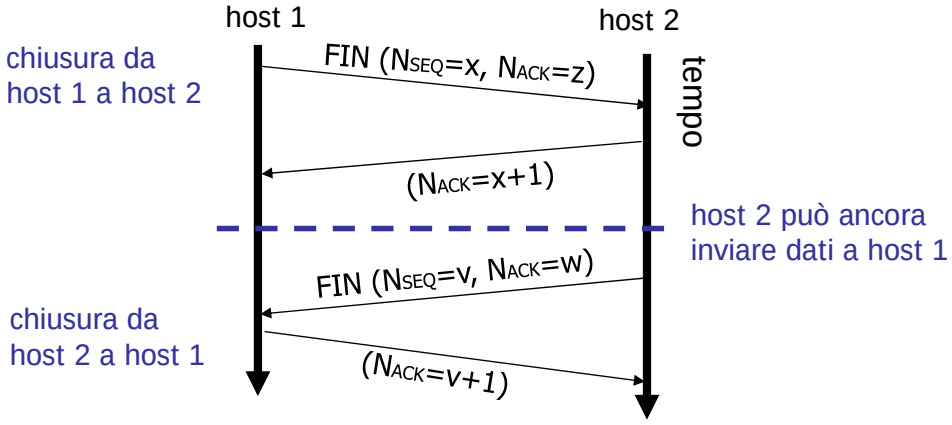
Duplicati di richieste di connessione

- Una richiesta di connessione può essere “memorizzata” e ricomparire nella rete, quando in realtà la connessione è stata già stabilita tramite una richiesta successiva. Bisogna quindi evitare di stabilire erroneamente una seconda connessione.
- Supponiamo che il server riceva un duplicato di una richiesta di connessione. Poiché non può sapere che è un duplicato, risponde con l’ack della richiesta e con un suo numero di sequenza.
- Il client però non è in attesa di una risposta del server con quel numero di ack, quindi rifiuta la connessione.



Chiusura della Connessione

- La connessione è full-duplex e le due direzioni devono essere chiuse indipendentemente, con relativo ACK di riscontro.
- I segmenti di chiusura connessione (segmenti cosiddetti “FIN”) hanno il bit FIN=1.



TCP: timer di ritrasmissione

- Il problema è quello di determinare la durata ottima del timer. I ritardi di trasmissione possono variare nel tempo, e TCP ne deve tenere conto nel fissare la durata del timer.
  - Se il timer è troppo piccolo si faranno ritrasmissioni inutili
  - Se il timer è troppo elevato si avranno ritardi di ritrasmissione eccessivi
- Si utilizza un algoritmo di stima del **Round-Trip Time** (**RTT**), cioè del tempo di round-trip (andata e ritorno) verso il ricevitore.

Algoritmo per settare il timer di ritrasmissione

- Per ogni connessione si mantiene una stima di **RTT**, aggiornata ad ogni pacchetto, con la seguente formula
  - tipicamente α=7/8; **M** è il ritardo calcolato alla ricezione dell'i-esimo ack

$$\mathbf{RTT_i = \alpha\, RTT_{i-1} + (1-\alpha)\, M}$$

- Si stima allo stesso modo anche la deviazione standard del ritardo:

$$\mathbf{D_i = \alpha\, D_{i-1} + (1-\alpha)\, |RTT_i - M|}$$

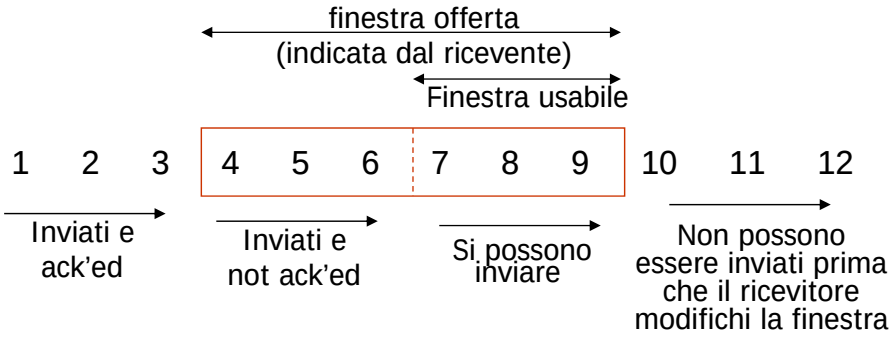
- Si imposta quindi

$$\mathbf{durata\ timer = RTT + 4*D}$$

- Ci sono altre soluzioni (es. algoritmo di Karn: raddoppio del timer ad ogni fallimento).

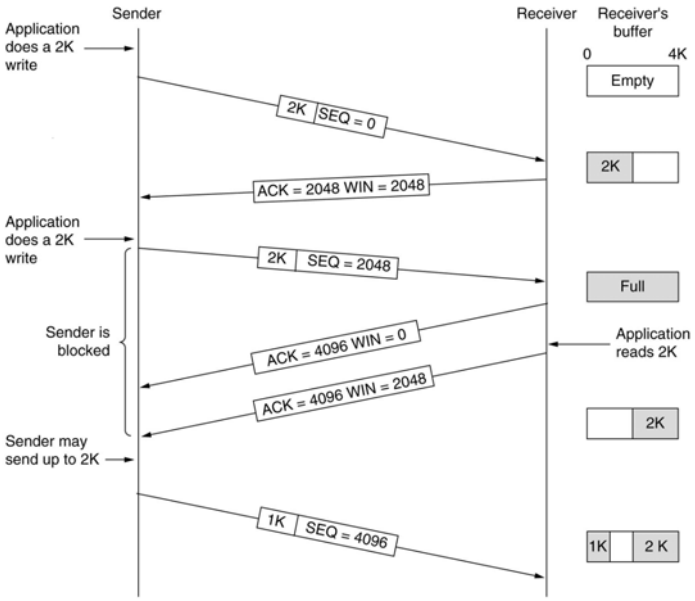
Controllo di flusso: finestra del ricevitore

- Il trasmettitore deve evitare di inviare dati che il ricevitore non può gestire; viene utilizzato un protocollo a finestra scorrevole (**sliding window**)
- Il ricevente indica, tramite il campo **windows size**, la dimensione della finestra che può gestire in un dato momento
- La finestra del ricevitore si restringe quando il ricevitore riceve dei byte dal trasmettitore, e si allarga quando i processi applicativi del ricevitore prelevano i dati dal buffer di TCP del ricevitore
- Sotto: esempio di invio dei dati da parte del trasmettitore; vengono inviati i byte che fanno parte della finestra indicata dal ricevitore



Gestione del flusso dei dati

- Il trasmittente invia il numero di sequenza del primo byte del segmento.
- Il ricevente invia l'ack e l'ampiezza della finestra ancora disponibile.
- Il ricevente può indicare una finestra 0 (in tal modo blocca il mittente).



Esercizio

Si supponga di avere instaurato una connessione TCP tra l'host A e l'host B. L'host A deve inviare all'host B in tutto 8.000 byte di traffico utile (cioè di dati del livello applicativo). Il Maximum TCP payload della connessione è di 5.000 byte. Si supponga inoltre che la finestra di congestione del trasmettitore non ponga alcun limite all'invio di dati.

In ogni segmento inviato a B, l'host A inserisce la massima quantità di dati possibile. Nel primo segmento contenente traffico utile (*segmento A.1*), A invia a B 5.000 byte di dati dell'applicazione, e nell'intestazione pone il valore del campo *Sequence Number* uguale a 12.001 (valore in decimale).

Si supponga che l'host B risponda con un segmento (*segmento B.1*) avente 1.000 byte di dati utili, tutti ricevuti correttamente da A.

Si considerino 4 scenari differenti, in cui il segmento *B.1* contiene rispettivamente i seguenti valori:

a) Sequence Number = 36.001; Acknowledgement Number = 13.501; Window Size = 4.000.
b) Sequence Number = 36.001; Acknowledgement Number = 17.001; Window Size = 16.000.
c) Sequence Number = 24.001; Acknowledgement Number = 17.001; Window Size = 2.000.
d) Sequence Number = 24.001; Acknowledgement Number = 13.501; Window Size = 16.000.

Per ognuno dei 4 casi, specificare, per il successivo segmento (*segmento A.2*) inviato da A a B, i valori dei seguenti campi:

1. Sequence Number
2. Ack Number
3. numero di bytes "utili**"** inviati nel segmento (cioè intestazione a parte)

Gestione del flusso dei dati (2)

- Il mittente non è obbligato ad inviare i dati appena li riceve dall'applicazione: può bufferizzarli.
- Il ricevente non è obbligato ad inviare subito gli ack alla controparte: anche lui può usare un buffer.
- Queste caratteristiche possono essere sfruttate per migliorare l'efficienza, cioè per diminuire il traffico sulla rete.
- Si consideri il caso di una **connessione interattiva** (es. Telnet).
  - Il mittente invia un segmento per ogni carattere battuto dall'utente.
  - Il server ricevente genera un ack ed un echo per ogni carattere battuto.

Flusso di dati interattivi (es. Telnet)

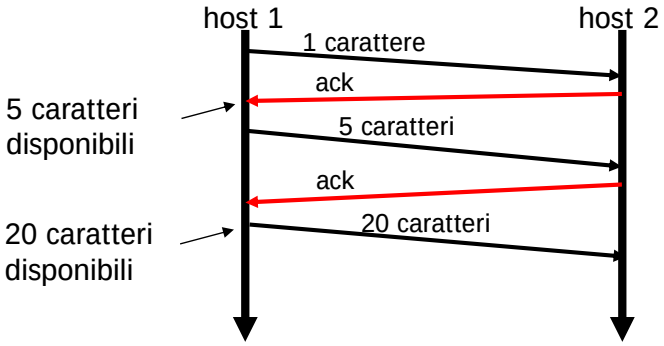
- Traffico dati per gestire un singolo carattere in una connessione interattiva Telnet.
  - Segmento inviato dal client col carattere battuto: 20 Header IP + 20 Header TCP + 1byte dati TCP = 41 byte
  - Segmento di ack TCP dal server al client = 40 byte
  - Segmento di echo dell'applicativo Telnet dal server = 41 byte
  - Segmento di ack TCP dal client per l'echo di Telnet = 40 byte
- **In totale si usano 162 byte in 4 segmenti TCP per 1 solo carattere della connessione Telnet!**

Ack ritardati (piggybacking)

- Normalmente il ricevente non invia un ack istantaneamente ma aspetta di avere dati da spedire insieme con l'ack (nel caso precedente il messaggio di echo di Telnet)
- L'attesa è in genere di 500 ms
- Questa tecnica è detta **Ack piggybacking**
- Il problema di Telnet è risolto solo in parte:
  - il ricevente riduce il carico che impone alla rete, perché invia l'ack del byte ricevuto (risposta del livello di trasporto) insieme all'echo di Telnet (risposta del livello applicativo)
  - tuttavia il mittente continua ad inviare segmenti di 41 byte con 1 solo byte utile

Algoritmo di Nagle

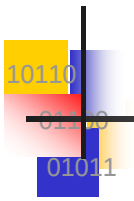
- Ha effetto per connessioni lente (es. WAN)
- Inizialmente il mittente invia anche 1 solo byte
- Poi accumula i dati fino a che non riceve l'ack per il segmento inviato in precedenza; a questo punto invia i dati accumulati
- In questo modo ogni segmento può contenere una quantità significativa di dati



In alcuni casi (alta interattività) l'algoritmo di Nagle deve essere disabilitato (es. movimenti del mouse in connessioni remote).

Algoritmo di Clark

- **Problema: sindrome della finestra stupida** (silly window syndhrome)
  - Accade quando il mittente ha molti dati da inviare ma l'applicazione del ricevente legge 1 byte alla volta
  - All'inizio il buffer del ricevente è pieno
  - Il ricevente avverte il mittente appena l'applicazione legge 1 byte, costringendo il mittente ad inviare 1 byte alla volta!
- **Soluzione: algoritmo di Clark**
  - Il ricevente non invia subito gli aggiornamenti riguardo il proprio buffer
  - Attende invece che nel suo buffer ci sia uno spazio libero pari almeno alla dimensione del segmento massimo della connessione o alla metà del buffer stesso



Algoritmi di Nagle e Clark

Sono algoritmi complementari ed in genere utilizzati insieme

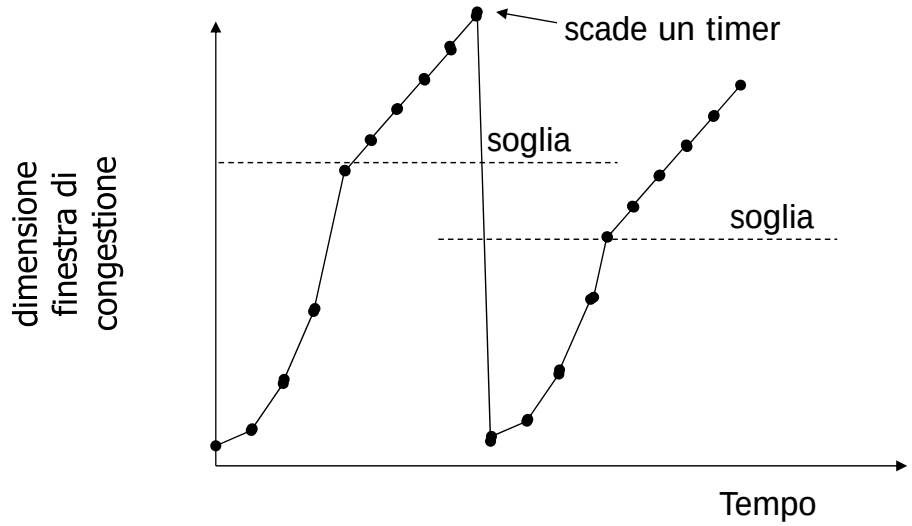
- **Nagle** evita che il mittente invii i dati 1 byte alla volta
- **Clark** evita che il ricevente *chieda* al mittente di inviare i dati 1 byte alla volta

Controllo di congestione: finestra di congestione

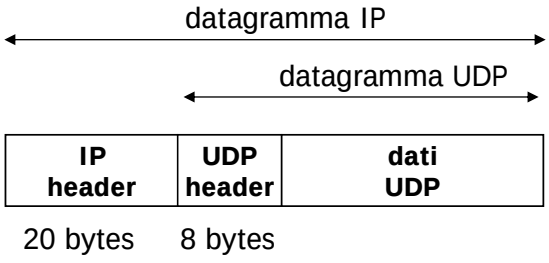
- Il TCP deve adattare la velocità di trasmissione alla capacità della rete, per evitare la congestione
- Oltre alla finestra del ricevitore, si utilizza la **finestra di congestione** del trasmettitore
- Il trasmettitore può inviare una quantità di dati pari alla *finestra più piccola* tra quella di congestione e quella comunicata dal ricevitore
- La dimensione della finestra di congestione è ridotta se si ha sentore di una possibile congestione (es. se scade un timer di ritrasmissione); è invece aumentata se i pacchetti vengono consegnati prima della scadenza del timer

Finestra di congestione

- Esempio di funzionamento dell'algoritmo di controllo della congestione di TCP (con uso di una **soglia** inizialmente uguale a 32 KB).



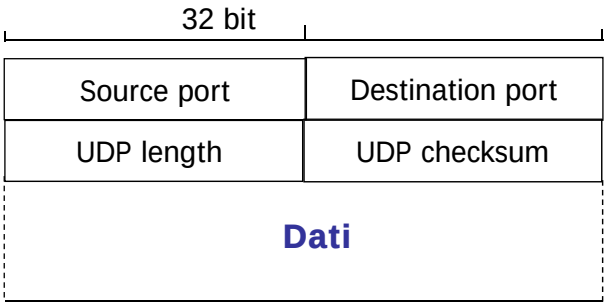
UDP: Trasporto senza Connessione



- UDP non garantisce: affidabilità di consegna (non gestisce ack e ritrasmissioni), ordinamento dei dati, controllo di flusso ecc.
- Viene utilizzato soprattutto per:
  - Applicazioni real time
  - Applicazioni request-response (es. DNS o interrogazione di un database)
  - Chiamate a procedura remota (**RPC = Remote Procedure Call**)
  - Applicazioni multicast

Header UDP

- Header di un datagramma UDP: soli 8 byte, dimensione fissa.



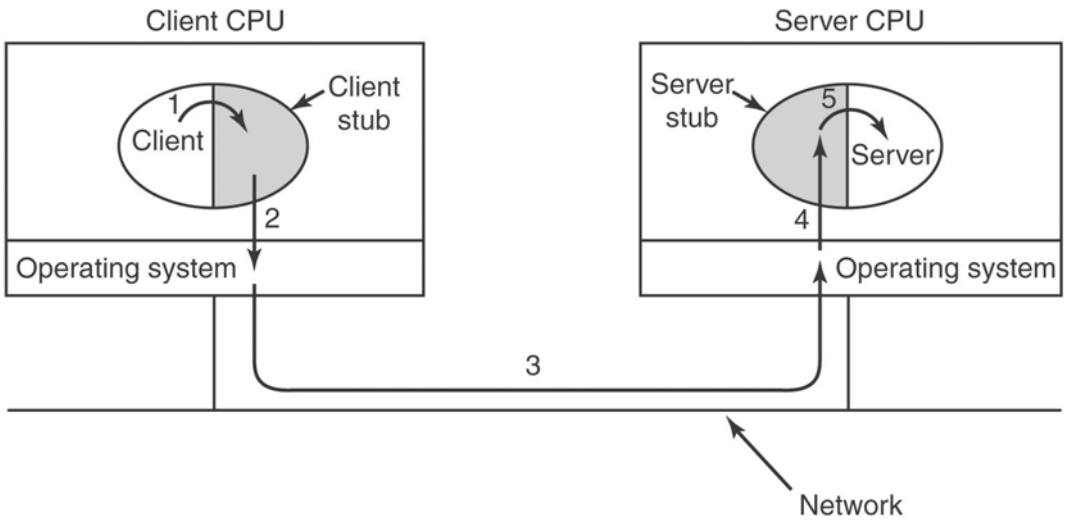
- Le **porte** UDP sono indipendenti da quelle TCP
- La **lunghezza** in byte comprende sia i dati che l'header
- Il **checksum** controlla la correttezza dell'intestazione e dei dati: non sempre conviene usarlo, perché rallenta le comunicazioni

Chiamata a procedura remota (RPC)

- L'interazione client/server (e l'invio dei relativi messaggi) è modellata come una chiamata a procedura.
- È un'astrazione che consente di facilitare la programmazione e la gestione delle applicazioni di rete (es. **RMI** di Java, **CORBA**, **DCOM** di Microsoft, **Web services**).
- Virtualmente il programma chiamante A (client) invoca una procedura sul programma B (server) in esecuzione su un host remoto.
- In realtà il client invoca, in maniera trasparente per l'utente, una procedura di libreria (**stub**) che risiede sull'host del client; lo stub del client invia un **messaggio** (UDP o TCP) allo stub che risiede sull'host del server, che a sua volta invoca la procedura remota.
- I parametri della chiamata sono inseriti nel messaggio tramite una tecnica nota come **marshalling**.

Chiamata a procedura remota (RPC)

- Passaggi per effettuare una chiamata a procedura remota. Nella figura gli stub sono in grigio.



Trasmissioni real-time

- Applicazioni: radio su Internet, voice over IP, videoconferenza, video e musica on demand ecc.
- TCP garantisce un servizio affidabile, ma può comportare ritardi non ammissibili per queste applicazioni, a causa delle ritrasmissioni.
- Si usa spesso il protocollo **RTP** (**Real-Time Transport Protocol**) che è in effetti un protocollo di trasporto implementato nel livello di applicazione.
- RTP utilizza il servizio di base di **UDP**.
- RTP gestisce funzionalità utili per le trasmissioni real-time: **contrassegni temporali**, **multiplexing** di più flussi di dati (es. audio e video) su un unico flusso di pacchetti UDP, **buffering** sul ricevitore.