UNIVERSITÀ DEGLI STUDI DI UDINE

hic sunt futura

TECNOLOGIE WEB E LABORATORIO (a.a. 2022-2023)

INTERNET - PARTE 2

Daniele Salvati

Dipartimento di Scienze Matematiche, Informatiche e Fisiche
Università degli Studi di Udine

Informazioni slide

- Il materiale contenuto in queste slide è riservato esclusivamente agli studenti del corso di Tecnologie Web e Laboratorio del Corso di Studio in **Internet of Things, Big Data, Machine Learning** dell'Università degli Studi di Udine.

- Non è consentita la diffusione del materiale contenuto in queste slide, ma solo l'utilizzo inerente la preparazione dell'esame del suddetto corso.

Livello
di applicazione

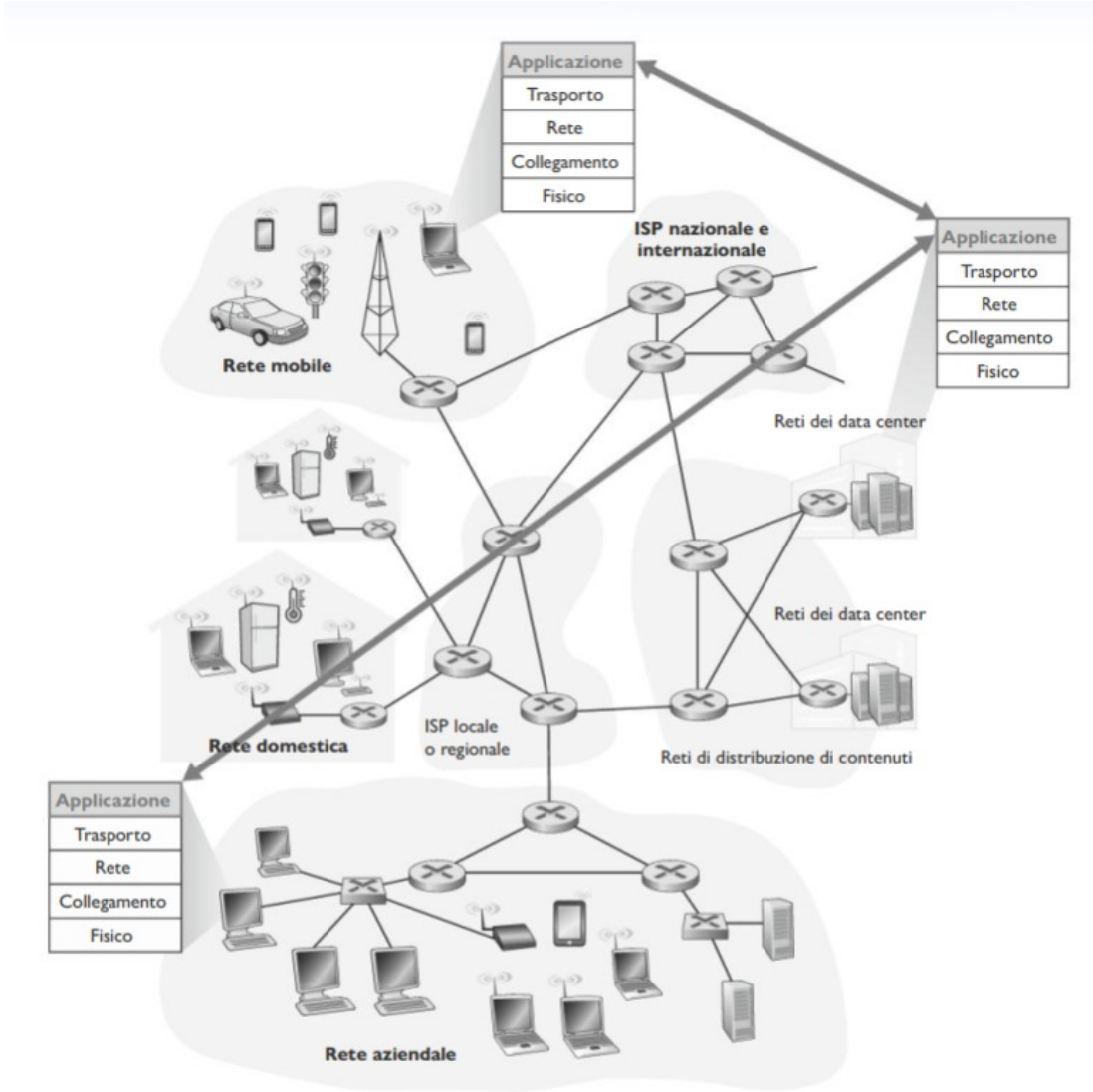
Applicazioni di rete

- Il cuore dello sviluppo delle applicazioni di rete è costituito dalla creazione dei **programmi** che sono **eseguiti dai sistemi periferici** e che comunicano tra loro via rete.

- I programmi hanno l'obiettivo di definire **l'architettura dell'applicazione** (*application architecture*) e stabilire la sua organizzazione sui vari sistemi periferici, basandosi su una delle due principali architetture di rete attualmente utilizzate:
  - client-server
  - peer-to-peer (P2P)

Applicazioni di rete (2)

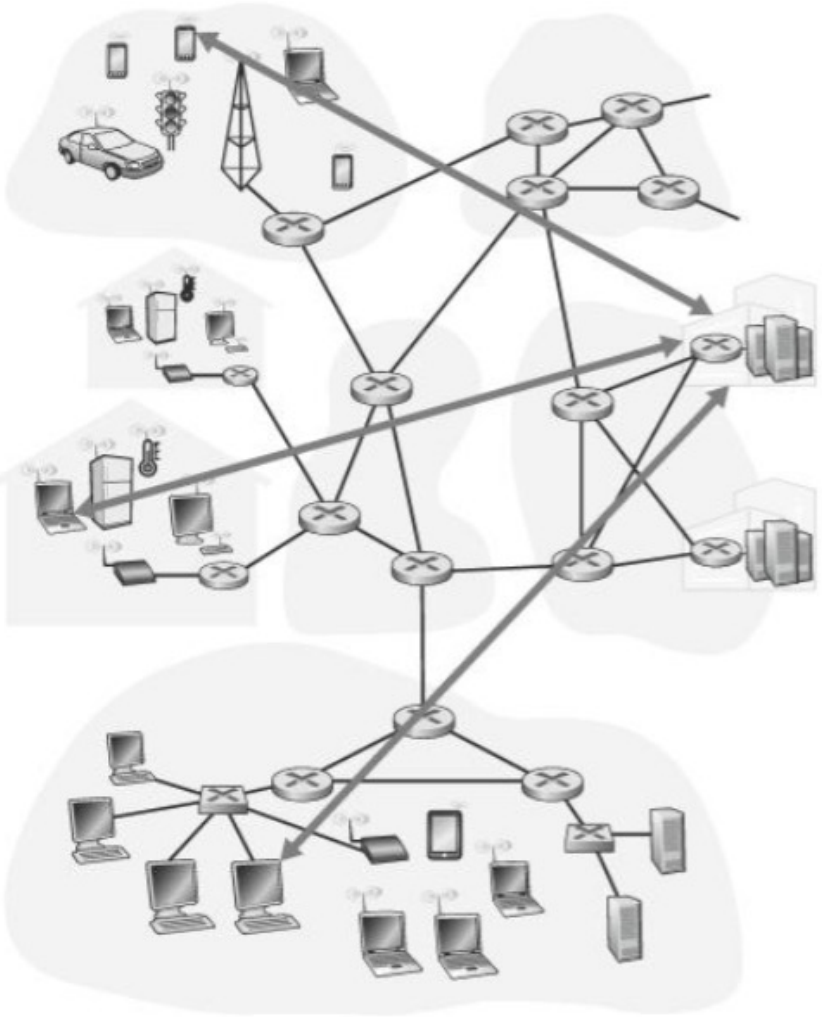
La comunicazione in un'applicazione di rete ha luogo tra sistemi periferici a livello di applicazione.



Architettura client-server

- Nell'architettura **client-server** (*client-server architecture*) vi è un host sempre attivo, chiamato server, che risponde alle richieste di servizio di molti altri host, detti client.

- **CLIENT**: è l'entità che chiede il servizio
  - Es. un PC che attraverso il browser (Firefox, Safari, Chrome) richiede una pagina web.
- **SERVER**: è l'entità che fornisce il servizio
  - Es. un web server Apache (software) installato in una macchina server.



Architettura client-server (2)

- Un singolo host che esegue un server spesso non è in grado di rispondere a tutte le richieste dei suoi client.

- Per questo motivo nelle architetture client-server si usano spesso **data center** che, ospitando molti host (creano un potente server virtuale).

- Esempio: Google dispone di 19 data center sparsi in tutto il mondo che collettivamente gestiscono ricerche, YouTube, Gmail e altri servizi.

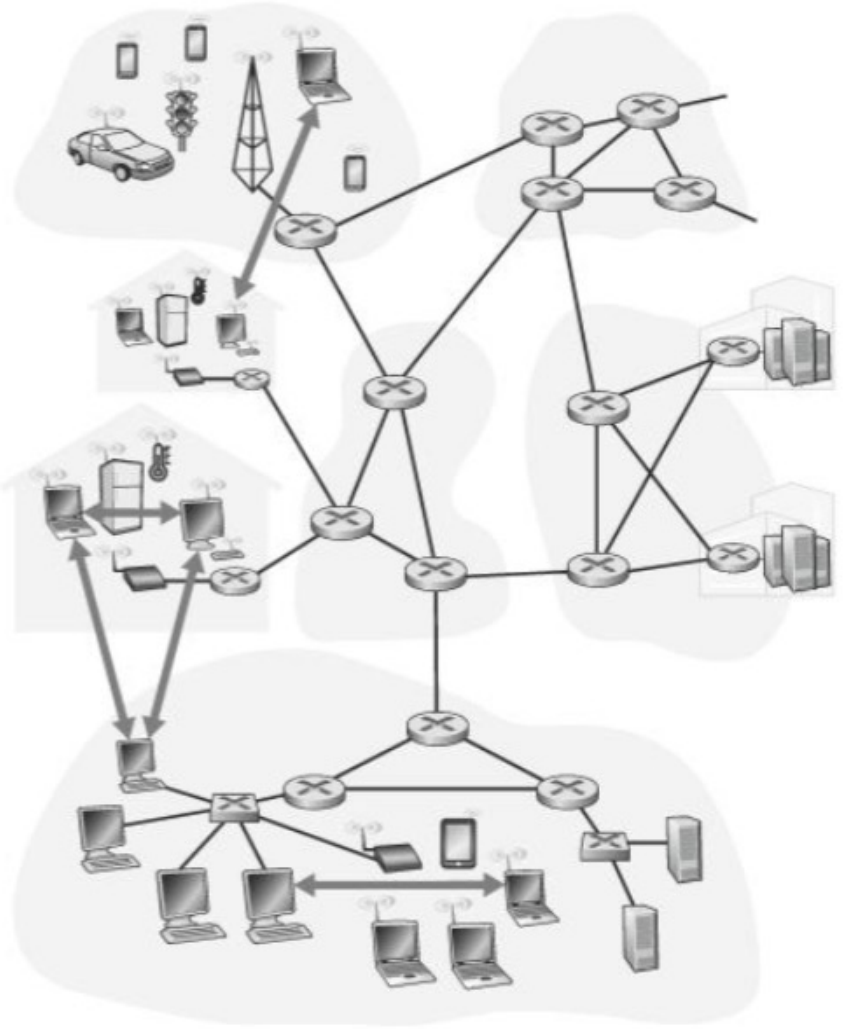
Architettura client-server (3)

- **Server**:
  - Host sempre attivo
  - Indirizzo statico
  - Può ricevere richieste dai client in qualunque momento

- **Client**:
  - Comunica con il server
  - Può contattare il server in qualunque momento
  - Può avere indirizzo dinamico
  - Non comunica direttamente con gli altri client

Architettura peer-to-peer

- **Peer-to-peer** (**P2P**): architettura di reti a livello applicativo dove non vi è un server sempre attivo, ma coppie arbitrarie di *host*, chiamati ***peer*** che comunicano direttamente tra di loro.
- Nessuno degli *host* che prende parte all'architettura P2P deve necessariamente essere sempre attivo.
- Ciascun ***peer*** può ricevere ed inviare richieste e può ricevere ed inviare risposte .



Architettura peer-to-peer (2)

- In un'architettura P2P l'infrastruttura di server in data center è minima o del tutto assente; si sfrutta, invece, la comunicazione diretta tra coppie di host.
- Uno dei punti di forza dell'architettura P2P è la sua intrinseca **scalabilità**.
- In un'applicazione di condivisione dei file P2P, ogni peer, sebbene generi carico di lavoro richiedendo dei file, aggiunge anche capacità di servizio al sistema, rispondendo alle richieste di altri peer.

Architettura peer-to-peer (3)

- Le architetture P2P sono anche economicamente convenienti, perché normalmente non richiedono per i server né una significativa infrastruttura né una disponibilità di banda elevata (al contrario dell'architettura client-server con data center).

- **Problemi**:
  - Sicurezza
  - Prestazioni
  - Affidabilità

Protocollo a livello di applicazione

- Un protocollo a livello di applicazione definisce come i processi di un'applicazione, in esecuzione su sistemi periferici diversi, si scambiano i messaggi.
- Un protocollo a livello di applicazione definisce:
  - I tipi di messaggi scambiati (per esempio, di richiesta o di risposta)
  - La sintassi dei vari tipi di messaggio (per esempio, quali sono i campi nel messaggio e come vengono descritti)
  - La semantica dei campi, ossia il significato delle informazioni che contengono
  - Le regole sulla comunicazione per determinare quando e come un processo invia e risponde ai messaggi.

Protocollo a livello di applicazione (2)

- I protocolli di livello applicazione più utilizzati sono:
  - Protocolli di servizio
    - Domain Name System (**DNS**)
    - Dynamic Host Configuration Protocol (**DHCP**)
  - Protocolli di trasferimento file:
    - File Transfer Protocol (**FTP**)
  - Protocolli del web
    - Hypertext Transfer Protocol (**HTTP**)
    - Hypertext Transfer Protocol over Secure Socket Layer (**HTTPS**)
  - Protocolli di accesso a terminali remoti:
    - **Telnet**
    - Secure Shell (**SSH**)
  - Protocolli usati per realizzare il servizio di posta elettronica:
    - Simple Mail Transfer Protocol (**SMTP**)
    - Post Office Protocol (**POP**)
    - Internet Message Access Protocol (**IMAP**)

L’interfaccia tra il processo e la rete

- La maggior parte delle applicazioni consiste di coppie di processi comunicanti che si scambiano messaggi.

- Un processo invia messaggi nella rete e riceve messaggi dalla rete attraverso **un’interfaccia software detta socket**.

Indirizzamento

- Per scambiare messaggi le applicazioni hanno bisogno di indirizzi.
- In Internet, gli host vengono identificati attraverso i loro **indirizzi IP** *(Internet Protocol address)*
- Un indirizzo IP è un numero di 32 bit che identifica univocamente un host di Internet.
- **Internet Protocol:** protocollo del livello di rete

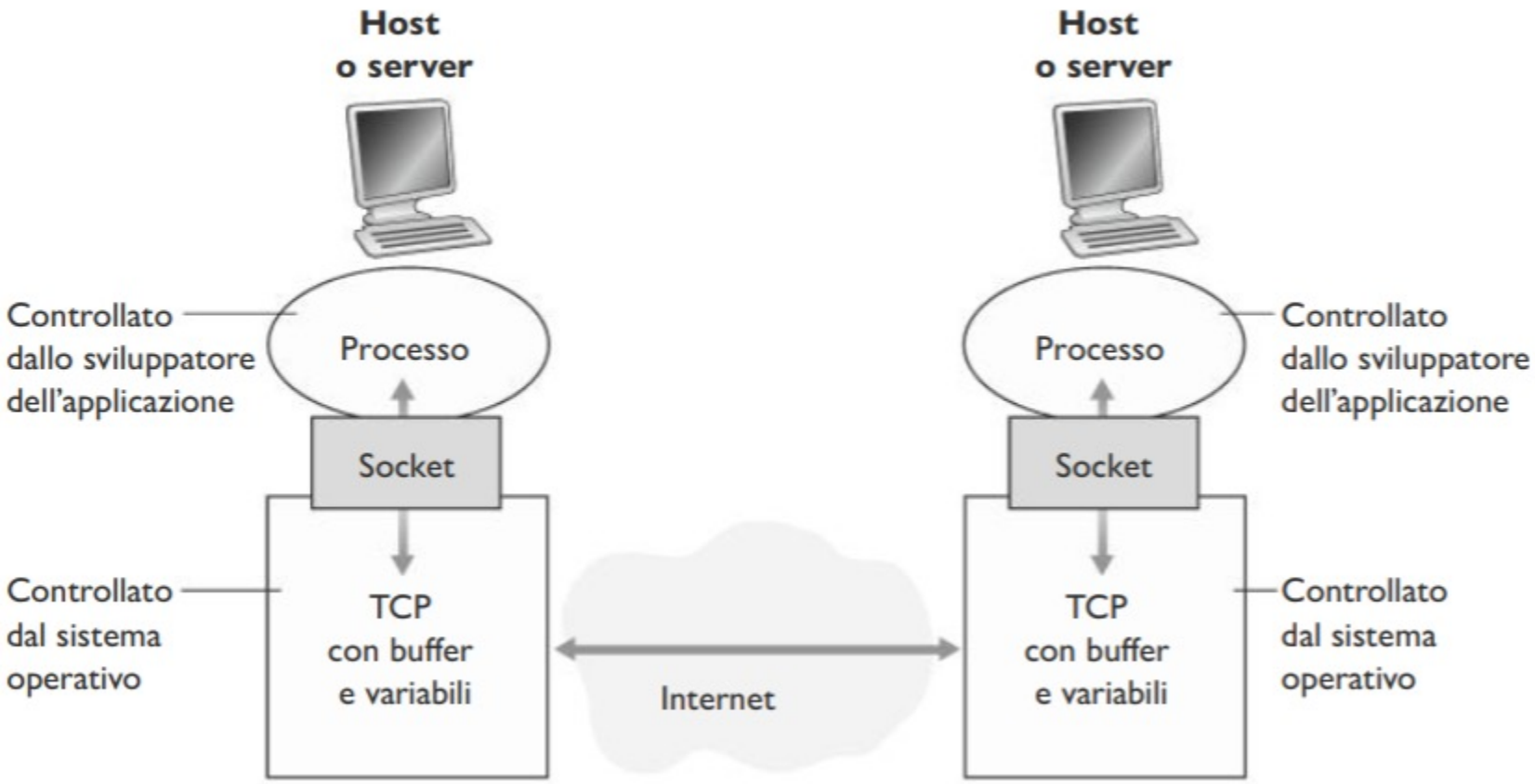
Socket

- Il mittente deve anche identificare il processo destinatario, più specificatamente la socket che deve ricevere il dato.

- La **socekt** è definita con un **numero di porta di destinazione** (numero con una grandezza di 16 bit).

- La socket quindi rappresenta l'interfaccia tra il protocollo applicativo e il protocollo di trasporto, cioè il luogo dove avviene lo scambio di informazione tra i due protocolli.
- La **socket in un sistema terminale** è definita come una coppia di parametri **<NA, PN>** in cui:
  - **NA** è l'indirizzo IP di rete dell'host dove viene eseguito il processo
  - **PN** è il numero di porta associato al processo

Socket (2)


Host
o server
Host
o server
Controllato
dallo sviluppatore
dell'applicazione
Processo
Processo
Controllato
dallo sviluppatore
dell'applicazione
Socket
Socket
Controllato
dal sistema
operativo
TCP
con buffer
e variabili
Internet
TCP
con buffer
e variabili
Controllato
dal sistema
operativo

Numero di porta

- I numeri di porta hanno 16 bit, si hanno quindi 2^{16} = 65536 porte.
- Quelli **minori di 1024** (0-1023) sono i cosiddetti numero di porta noti (*well-known port number*), riservati per i servizi standard ed assegnati dalla **Internet Assigned Number Authority (IANA)**
- Esempi:
  - Servizi di login
    - 23: Telnet
    - 22: SSH
  - Posta:
    - 25: SMTP
    - 110: POP
    - 143: IMAP
  - Web:
    - 80: HTTP

Numero di porta (2)

- I numeri di porta compresi **tra 1024 e 49151** non sono controllati direttamente dall'associazione IANA.
- È possibile tuttavia registrarne presso l'associazione il loro utilizzo per evitare possibili conflitti con altri utenti.
- Sono definiti numeri di porta registrati.
- I numeri di porta compresi **tra 49152 e 65536** non sono controllati dall'associazione IANA e non possono essere registrati.
- Le porte con tali numeri sono definite porte dinamiche.
- Questi numeri possono essere utilizzati liberamente.

Servizi per le applicazioni

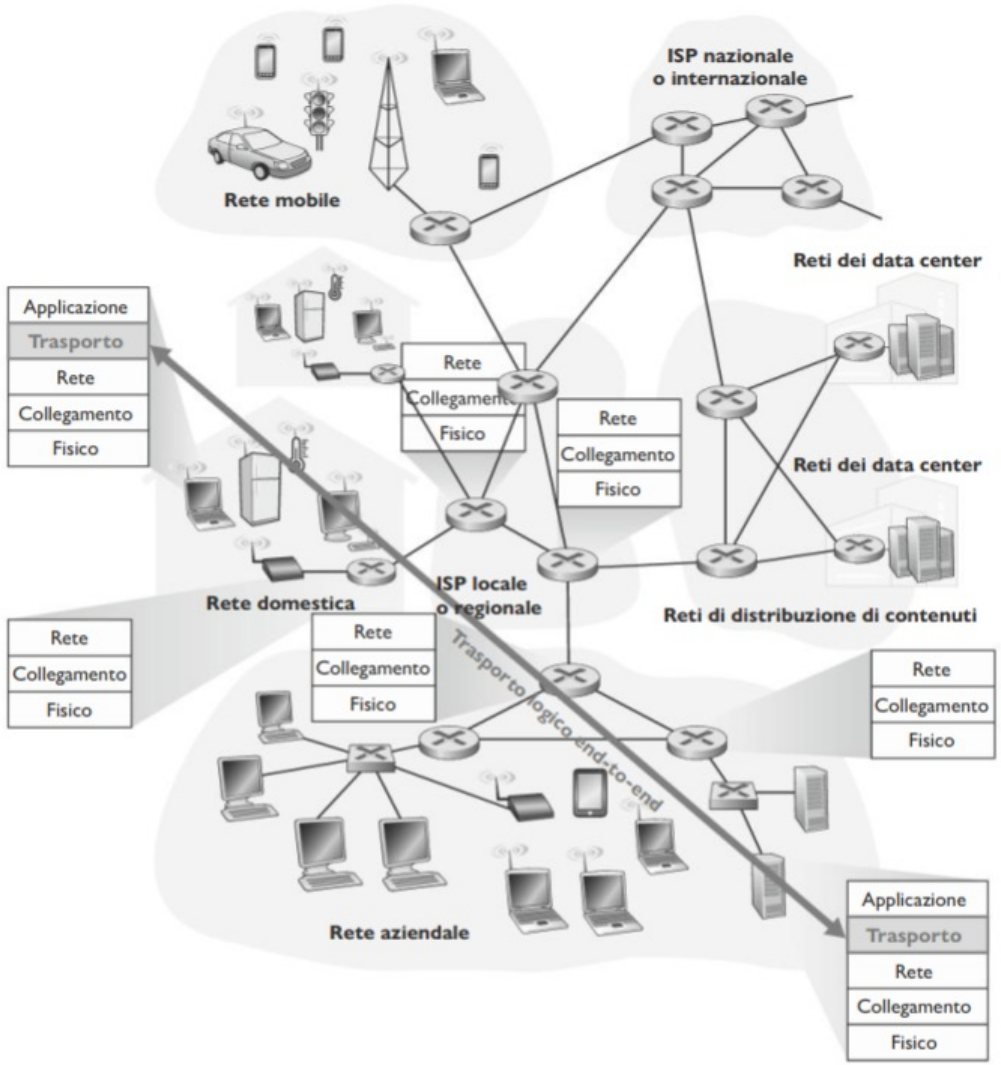
- Trasferimento dati affidabili
- Applicazioni che tollerano le perdite
- Requisiti di throughput (applicazioni sensibili alla banda)
- Applicazioni elastiche (posso tollerare variazioni di throughput)
- Temporizzazione (applicazioni sensibili ai ritardi)
- Sicurezza

Quali servizi può offrire il protocollo a livello di trasporto a un'applicazione che li invoca?

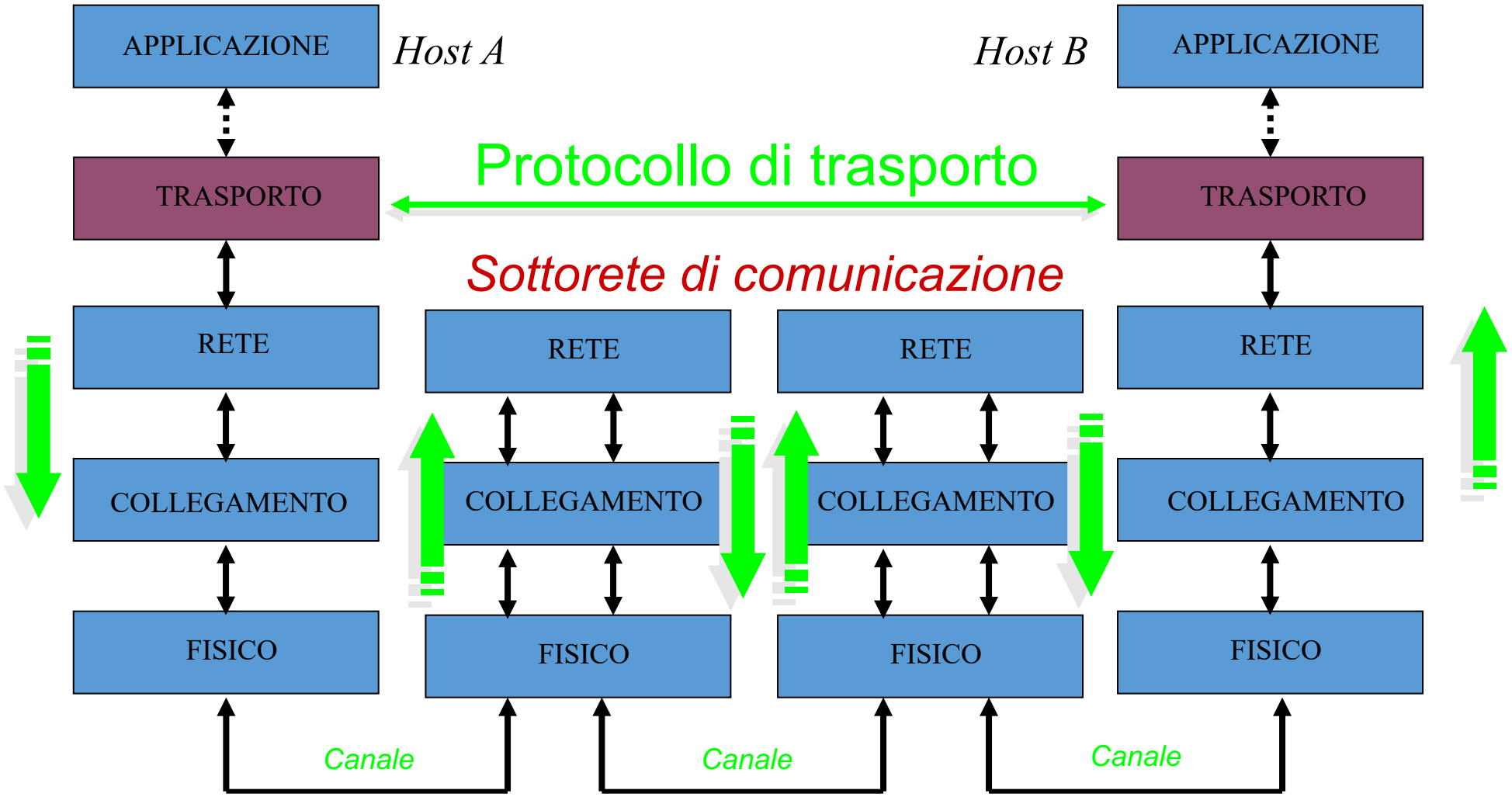
Livello
di trasporto

Protocollo a livello di trasporto

- Un protocollo a livello di trasporto mette a disposizione una **comunicazione logica** (*logical communication*) tra processi applicativi di host differenti.



Protocollo a livello di trasporto (2)



Livello di trasporto di Internet

- Internet mette a disposizione del livello di applicazione due diversi protocolli:
  - UDP (**User Datagram Protocol**), che fornisce alle applicazioni un servizio non affidabile e non orientato alla connessione.
  - TCP (**Transmission Control Protocol**), che offre un servizio affidabile e orientato alla connessione.

Livello di trasporto di Internet (2)

- Il servizio offerto dal livello di rete è generalmente inaffidabile (perdita pacchetti, ecc.).
- I protocolli a livello di rete non garantiscono la consegna dei segmenti (nome dei pacchetti nel livello di trasporto).

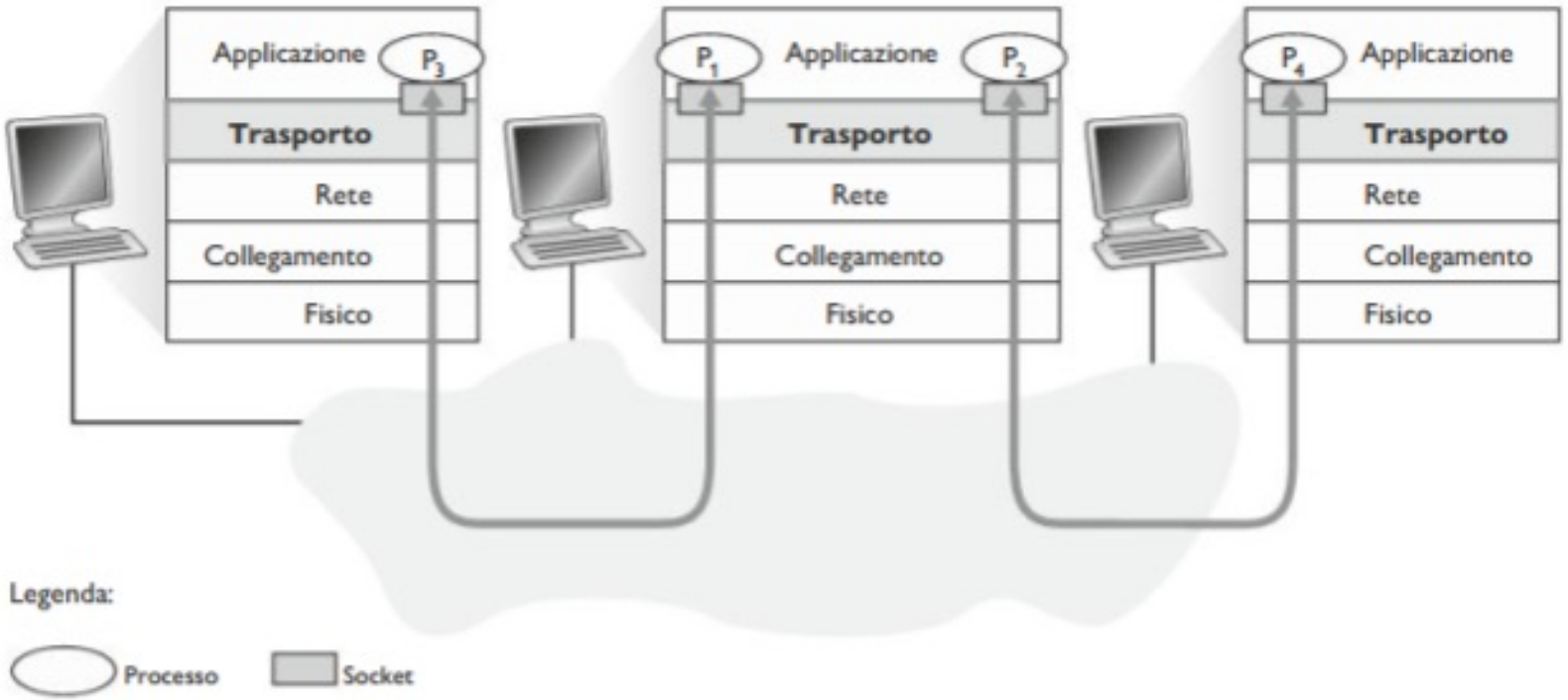
- Un **servizio affidabile** garantisce che i dati inviati da un processo arrivino al processo destinatario e arrivano intatti.

- Un servizio affidabile è gestito dal protocollo TCP del livello di trasporto.

Multiplexing e demultiplexing

- Il **multiplexing e il demultiplexing a livello di trasporto** estende la spedizione da host a host fornito dal livello di rete a un servizio di trasporto da processo a processo per le applicazioni in esecuzione sugli host.
- Nell'host destinatario il livello di trasporto riceve segmenti dal livello di rete immediatamente sottostante (il livello di rete).
- Il livello di trasporto ha il compito di consegnare i dati di questi segmenti al processo applicativo appropriato in esecuzione nell'host.

Multiplexing e demultiplexing (2)



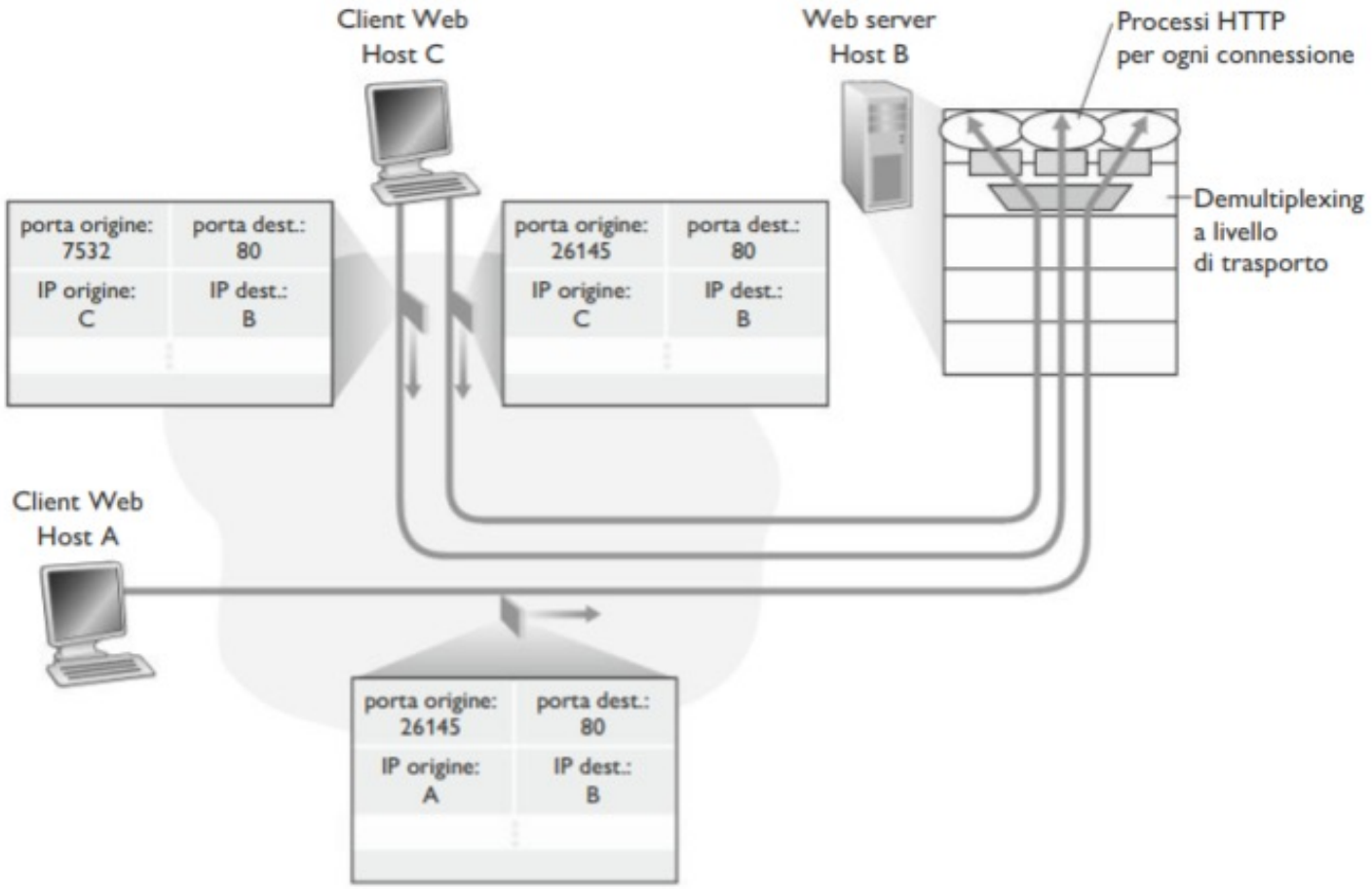
Multiplexing e demultiplexing (3)

- Il compito di trasportare i dati dei segmenti a livello di trasporto verso la giusta socket viene detto **demultiplexing.**

- Il compito di radunare frammenti di dati da diverse socket sull'host di origine e incapsulare ognuno con intestazioni a livello di trasporto (che verranno poi utilizzate per il demultiplexing) per creare dei segmenti e passarli al livello di rete, viene detto **multiplexing**.

- Il multiplexing e demultiplexing sono realizzati negli host con i protocolli UDP e TCP utilizzando il **numero di porta di origine** e il **numero di porta di destinazione**.

Multiplexing e demultiplexing (4)



UDP

- **UDP** è un protocollo di trasporto semplice.
- UDP è non orientato alla connessione (*connectionless*).
- UDP fornisce un servizio di trasferimento dati non affidabile.
- Quando un processo invia un messaggio tramite la socket UDP, il protocollo non garantisce che questo raggiunga il processo di destinazione. Inoltre i messaggi potrebbero giungere a destinazione non in ordine.
- UDP non include un meccanismo di controllo del flusso e della congestione.

UDP (2)

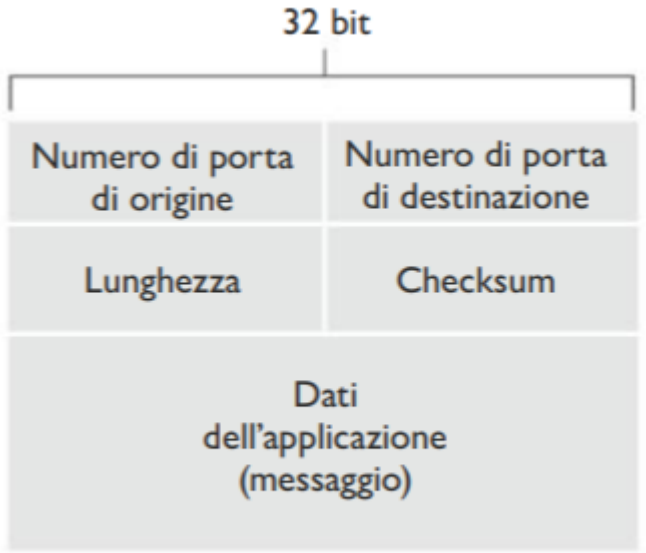
- Quando è utile usare UDP per le applicazioni:
  - Controllo più preciso a livello di applicazione su quali dati sono inviati e quando (es. applicazioni in tempo reale).
  - Nessuna connessione stabilita (UDP non introduce alcun ritardo nello stabilire una connessione).
  - Nessuno stato di connessione (un server dedicato a una particolare applicazione può generalmente supportare molti più client attivi quando l'applicazione utilizza UDP anziché TCP).
  - Minor spazio usato per l'intestazione del pacchetto (l'intestazione dei pacchetti TCP aggiunge 20 byte, mentre UDP solo 8 byte).

UDP (3)

- Esempi di applicazioni in cui è utile UDP:
  - Applicazioni in tempo reale (es. videochiamate) in cui non si può ritrasmettere.
  - Monitoraggio (es. acquisizione dati da sensori o altre sorgenti)
  - Distribuzione messaggi agli utenti di una rete (es. broadcast).
- Esempi di protocolli che storicamente usano UDP:
  - Domain Name System (**DNS**) (risoluzione dei nomi)
  - Simple Network Management Protocol (**SNMP**) (gestione di apparati collegati in una rete)
  - Network File System (**NFS**) (directory condivise da server remoti)
  - **H.323** (telefonia su Internet)

Struttura dei segmenti UDP

- L'intestazione UDP presenta solo quattro campi di due byte ciascuno.
- I **numeri di porta** consentono all'host di destinazione di trasferire i dati applicativi al processo corretto (ossia di effettuare il demultiplexing).
- Il campo **Lunghezza** specifica il numero di byte del segmento UDP (intestazione più dati).
- Il **Checksum** UDP serve per il rilevamento degli errori.



Checksum UDP

- Lato mittente UDP effettua il complemento a 1 della somma di tutte le parole da 16 bit nel segmento, e l'eventuale riporto finale viene sommato al primo bit.
- Somma bit:

A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A e B: bit da sommare

S: somma

R: riporto

esempio

R	1	0	0	1
A		1	0	0	1
B		1	1	0	1
S	1	0	1	1	0

Checksum UDP (2)

Esempio, supponiamo di avere le seguenti tre parole di 16 bit:

0110011001100000

0101010101010101

1000111100001100

La somma delle prime due è:

0110011001100000

0101010101010101

1011101110110101

Sommando la terza parola al risultato precedente otteniamo (senza riporto finale, in rosso):

1011101110110101

1000111100001100

10100101011000001

Checksum UDP (3)

Il riporto finale va infine sommato al primo bit:

0100101011000010

Il complemento a 1 si ottiene convertendo i bit 0 in 1 e viceversa. Il checksum sarà

1011010100111101

In ricezione, si sommano le tre parole iniziali e il checksum. Se non ci sono errori nel pacchetto, l'addizione darà **1111111111111111**, altrimenti se un bit vale 0 sappiamo che è stato introdotto almeno un errore nel pacchetto.

Sebbene metta a disposizione tale controllo, UDP non fa nulla per risolvere le situazioni di errore; alcune implementazioni di UDP si limitano a scartare il segmento danneggiato, altre lo trasmettono all'applicazione con un avvertimento.

TCP

- Il **TCP** è il protocollo di Internet a livello di trasporto affidabile e orientato alla connessione (*connection-oriented*) in quanto, prima di effettuare lo scambio dei dati, i processi devono effettuare l'**handshake**, ossia devono inviarsi reciprocamente alcuni segmenti preliminari per stabilire i parametri del successivo trasferimento dati.

- Lo stato della connessione risiede completamente nei due sistemi periferici e non negli elementi di rete intermedi (router e switch a livello di collegamento).

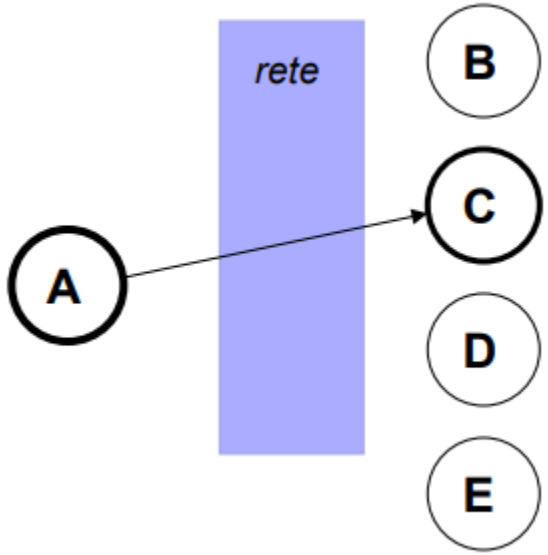
TCP (2)

- Una connessione TCP offre un servizio **full-duplex** (i dati a livello di applicazione possono fluire dal processo A al processo B nello stesso momento in cui fluiscono in direzione opposta).

- Una connessione TCP è punto a punto (*point-to-point*) , ossia ha luogo tra un singolo mittente e un singolo destinatario.

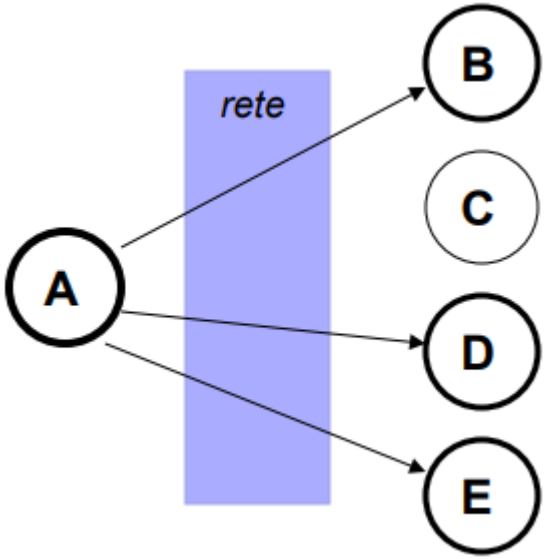
- La modalità **multicast**, ossia il trasferimento di dati da un mittente a molti destinatari in un'unica operazione, non è possibile con il TCP.

Unicast e multicast



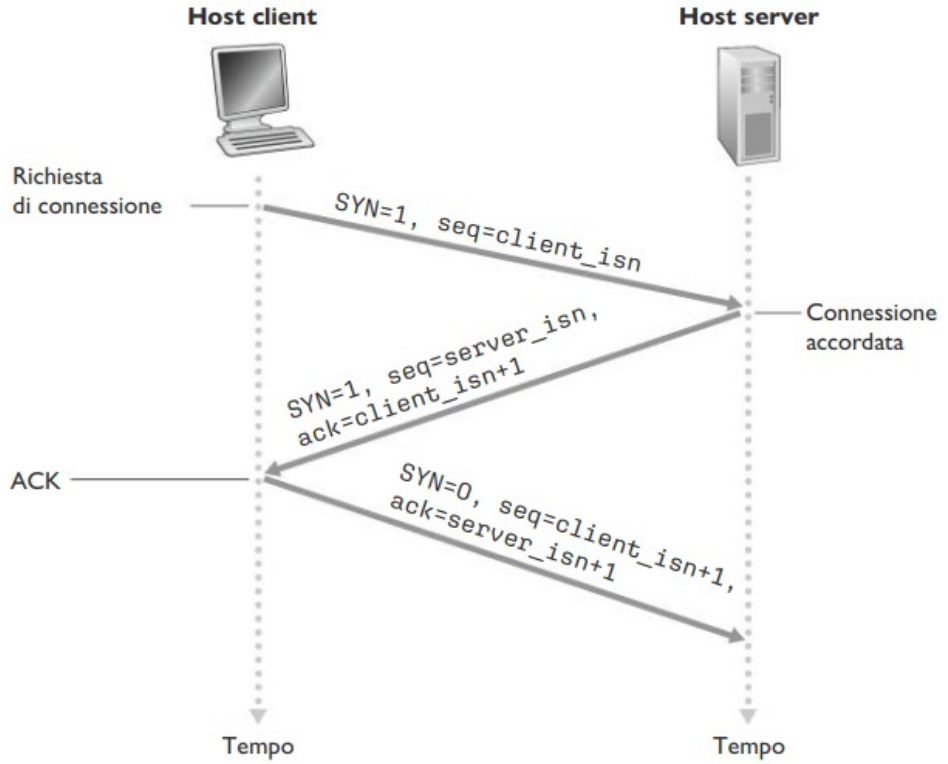
Unicast: i dati trasmessi dal mittente sono destinati e vengono ricevuti da un host specifico.

Multicast: i dati trasmessi dal mittente sono destinati e vengono ricevuti da un sottoinsieme specifico di tutti gli host connessi alla rete.



Connessione TCP

- La connessione TCP avviene con un handshake a tre vie:
  - Il client invia un segmento TCP speciale per la connessione (SYN = 1 e ACK=0) che non contiene dati. Il client sceglie un numero di sequenza iniziale (client_isn)
  - Il server invia al client un segmento di autorizzazione alla connessione (SYN=1, ACK=1) che non contiene dati, il campo di riscontro pari a client_isn+1 e il campo sequenza inizializzato con server_isn.
  - Dopo la ricezione del pacchetto che autorizza la connessione il client invia un ulteriore segmento che ha SYN=0, ACK=1, il campo riscontro con il valore server_isn+1 e il campo sequenza pari a client_isn+1.

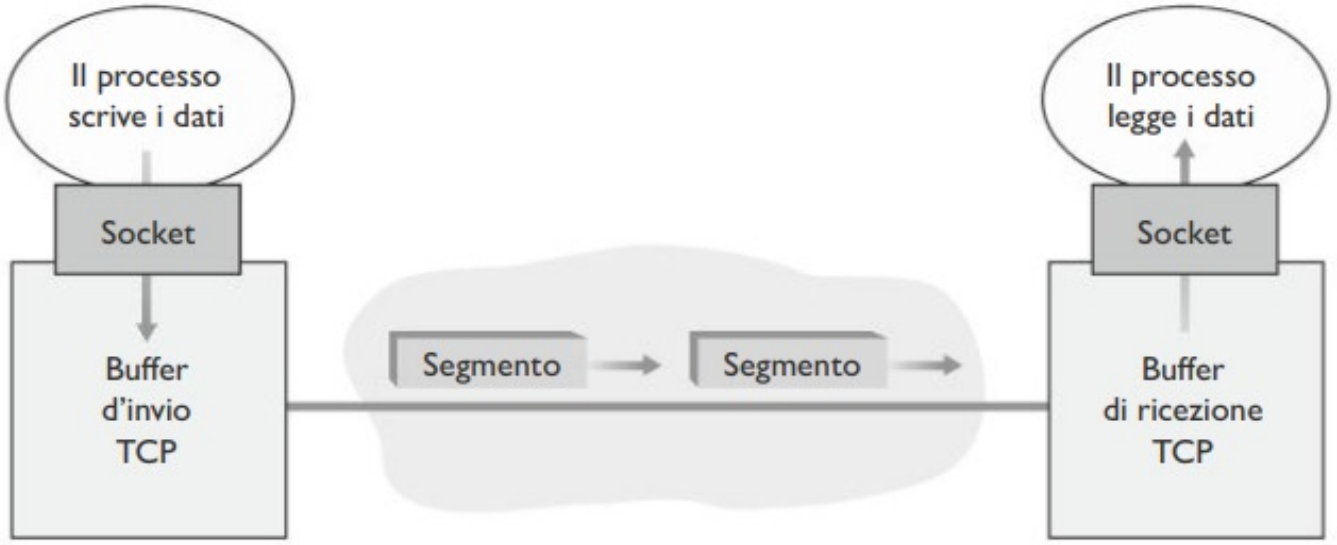


Connessione TCP (2)

- L’handshake a tre vie risolve il problema dei duplicati ritardatari.
- Ad esempio, a causa di ripetuti ritardi nell'invio degli acknowledgment può succedere che i segmenti vengano duplicati.
- I duplicati ritardatari possono provocare una attivazione di due connessioni con un handshake a due vie, e quindi tutto lo svolgimento delle attività ha luogo due volte.

Buffer TCP

- Lo scambio di dati fra host avviene utilizzando un buffer di invio e uno di ricezione

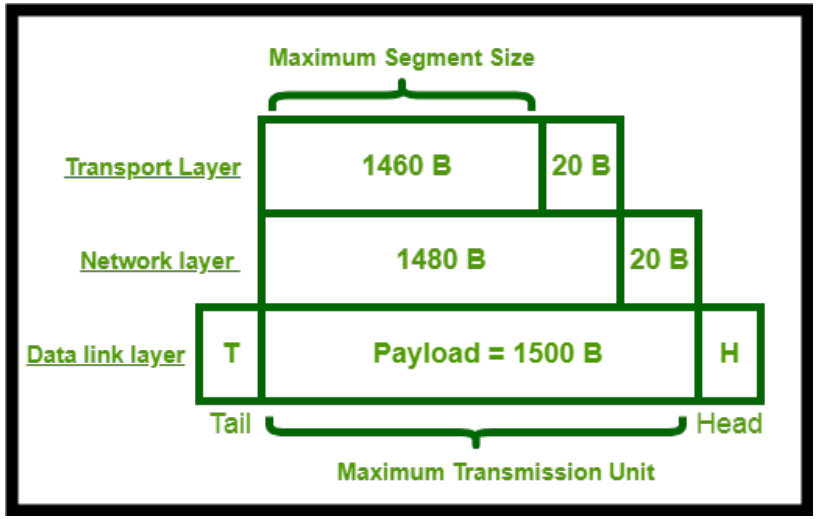


Segmento TCP

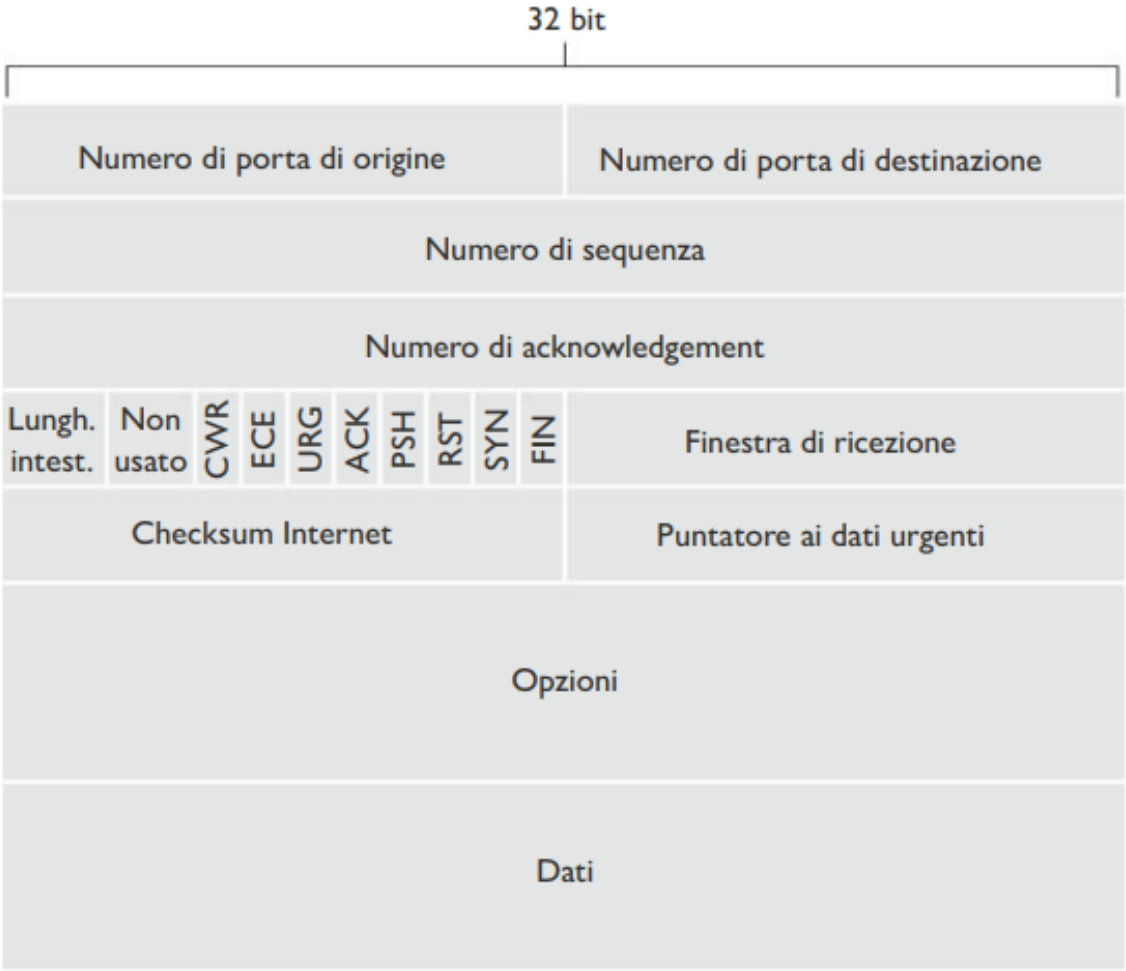
- La massima quantità di dati prelevabili e posizionabili in un segmento viene limitata dalla dimensione massima di segmento (**MSS**, *Maximum Segment Size*).
- Questo valore viene generalmente impostato determinando prima la lunghezza del frame più grande che può essere inviato a livello di collegamento dall'host mittente locale, la cosiddetta unità trasmissiva massima (**MTU**, Maximum Transmission Unit) e poi scegliendo un MSS tale che il segmento TCP (una volta incapsulato in un datagramma IP) stia all'interno di un singolo frame a livello di collegamento.

Segmento TCP (2)

- I protocolli a livello collegamento hanno valori tipico di **MTU** di 1500 byte.
- L'intestazione (*header*) TCP ha dimensione 20 byte (e 20 byte per il livello di rete).
- Un valori tipico di **MSS** è quindi 1460 byte.



Struttura dei segmenti TCP



Struttura dei segmenti TCP (2)

- I **numeri di porta di origine e di destinazione** sono utilizzati per il multiplexing/demultiplexing dei dati da e verso le applicazioni del livello superiore
- Il **Checksum** è usato per il controllo degli errori.
- Il campo **Lunghezza dell'intestazione** (*header length field*), di 4 bit, specifica la lunghezza dell'intestazione TCP in multipli di 32 bit. L'intestazione TCP ha lunghezza variabile a causa del campo delle opzioni TCP. Generalmente, il campo delle opzioni è vuoto e, pertanto, la lunghezza consueta è di 20 byte.

Struttura dei segmenti TCP (3)

- Il campo **Numero di sequenza** (*sequence number field*) e il campo **Numero di acknowledgment** (*acknowledgment number field*), entrambi di 32 bit, vengono utilizzati dal mittente e dal destinatario TCP per implementare il trasferimento dati affidabile.

- Il campo **Finestra di ricezione** (*receive window field*), di 16 bit, viene utilizzato per il controllo di flusso (il numero di byte che il destinatario è disposto ad accetta).

- Il campo **Opzioni** (options) è facoltativo e di lunghezza variabile.

Struttura dei segmenti TCP (4)

- Il bit **ACK** viene usato per indicare che il valore trasportato nel campo di acknowledgment è valido; ossia, il segmento contiene un acknowledgment per un segmento che è stato ricevuto con successo. I bit **RST**, **SYN** e **FIN** vengono utilizzati per impostare e chiudere la connessione. I bit **CWR** ed **ECE** sono usati nel controllo di congestione.

- I campi **PSH**, **URG** e il **Puntatore ai dati urgenti** sono relativi alla gestione di dati marcati come urgenti (nella pratica non vengono usati).

Numero di sequenza

- Il **numero di sequenza per un segmento** (*sequence number for a segment*) è il numero nel flusso di byte del primo byte del segmento.

- Esempio:

  - Si supponga che un processo sull'host A voglia inviare un flusso di dati ad un processo dell'host B attraverso una connessione TCP. Il flusso di dati è costituito da 500 000 byte e l'MSS è di 1000 byte.

Numero di sequenza (2)

- Esempio:
  - Il TCP costruisce i segmenti dividendo il flusso dati per l'MSS:
    - Numero segmenti = 500 000 / 1000 = 500
  - I numeri di sequenza dei segmenti (supponendo di iniziare la numerazione da 0) sono:
    - 0 (il primo segmento contiene i byte da 0 a 999)
    - 1000 (il secondo segmento contiene i byte da 1000 a 1999)
    - 2000 (il terzo segmento contiene i byte da 2000 a 2999)
    - 3000 (il quarto segmento contiene i byte da 3000 a 3999)
    - ecc.

Numero di acknowledgment

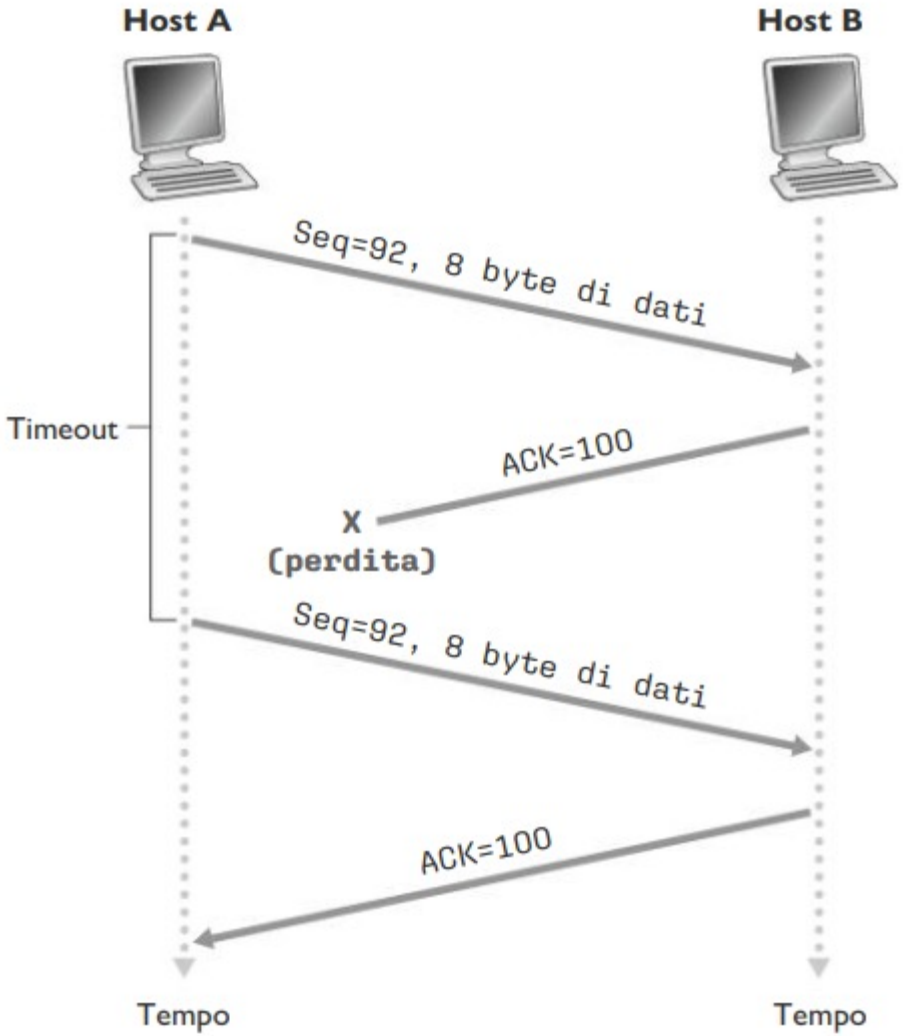
- Il **numero di acknowledgment** che l'Host A scrive nei propri segmenti è il numero di sequenza del byte successivo che l'Host A attende dall'Host B.

- Esempio: supponiamo che l'Host A abbia ricevuto da B tutti i byte numerati da 0 a 535 e che A stia per mandare un segmento all'Host B. L'Host A è in attesa del byte 536 e dei successivi byte nel flusso di dati di B. Pertanto, l'Host A scrive 536 nel campo del numero di acknowledgment del segmento che spedisce a B.

Timeout

- TCP utilizza un meccanismo di timeout (tempo scaduto) e ritrasmissione per recuperare i segmenti persi.

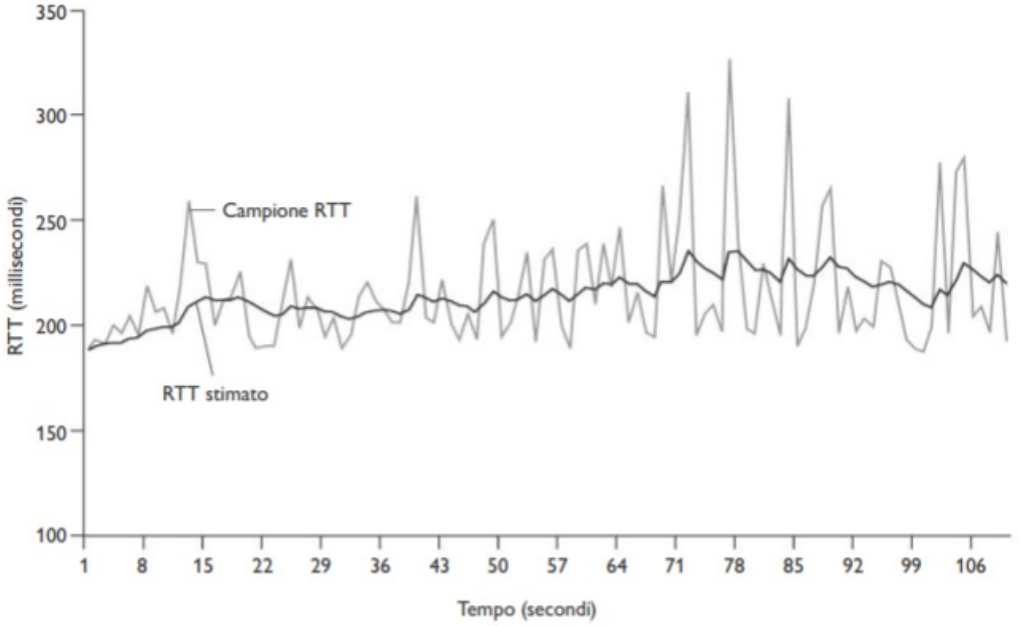
- L'Host A non riceve l' acknowledgment e ritrasmette lo stesso segmento dopo il timeout. Ovviamente, quando l'Host B riceve la ritrasmissione, rileva dal numero di sequenza che il segmento contiene dati che sono già stati ricevuti. Quindi, l'Host B scarta i byte del segmento ritrasmesso.



Tempo di andata e ritorno

- Il timeout dovrebbe essere più grande del tempo di **andata e ritorno sulla connessione** (**RTT**, *Round-Trip Time*), ossia del tempo trascorso da quando si invia un segmento a quando se ne riceve l'acknowledgment.

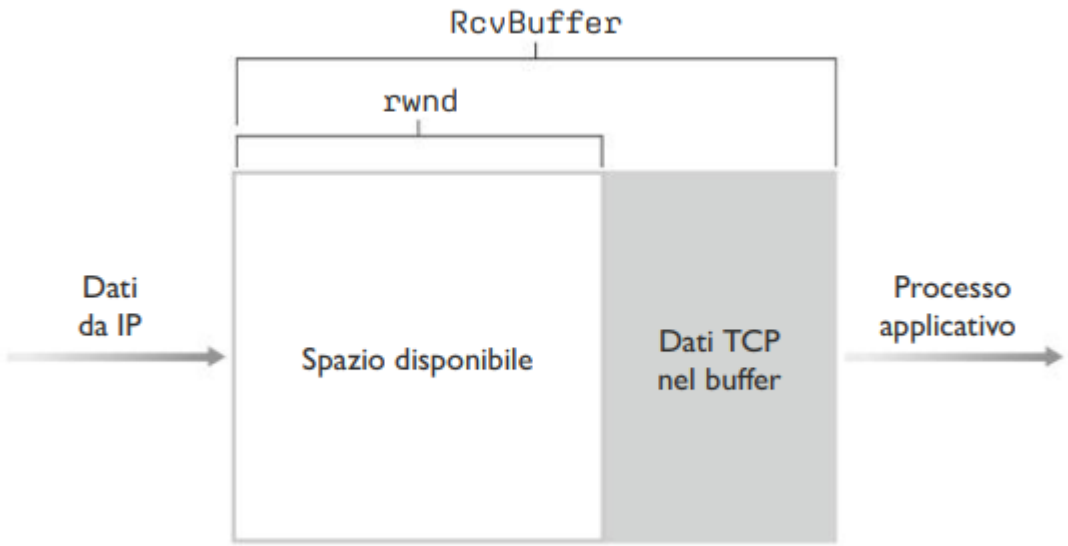
RTT: è la quantità di tempo che intercorre tra l'istante di invio del segmento (ossia quando viene passato al livello di rete) e quello di ricezione dell'acknowledgment del segmento.



Il TCP stima l'RTT calcolando una media dei valori

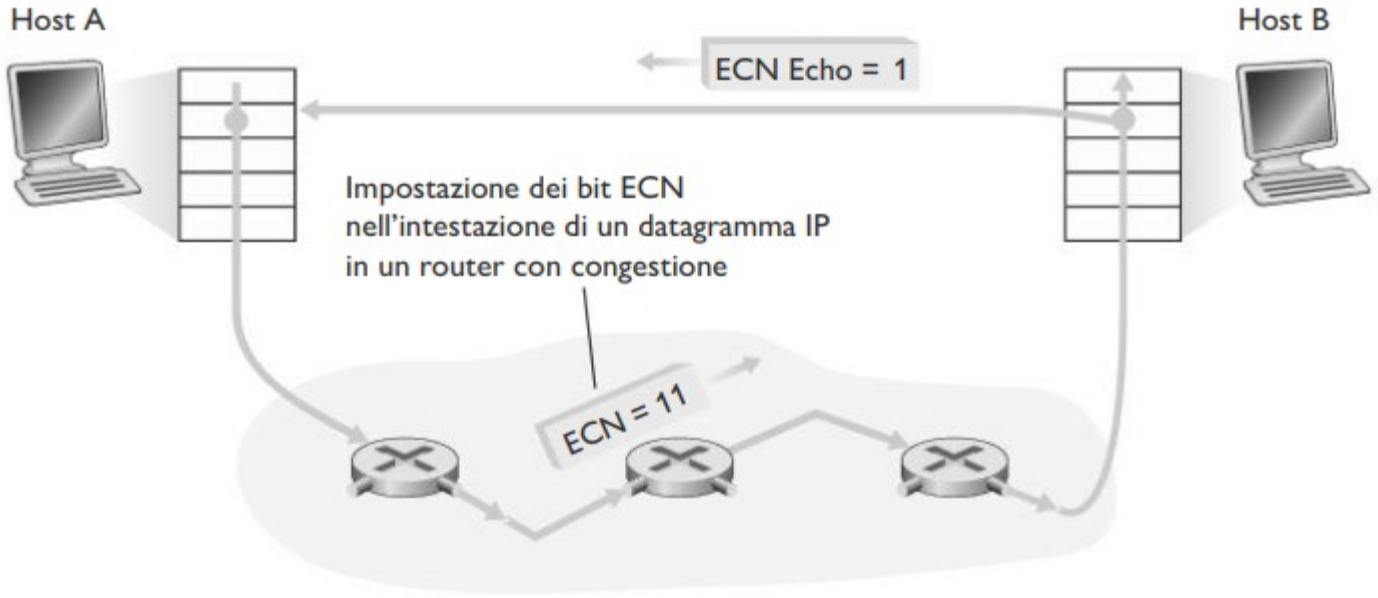
Controllo di flusso

- TCP offre il **controllo di flusso** facendo mantenere al mittente una variabile chiamata finestra di ricezione (*receive window*) che fornisce al mittente un'indicazione dello spazio libero disponibile nel buffer del destinatario.



Controllo di congestione

- Il meccanismo di **controllo di congestione TCP** fa tener traccia agli estremi della connessione di una variabile aggiuntiva: la finestra di congestione (*congestion window*), che impone un vincolo alla velocità di immissione di traffico sulla rete da parte del mittente.



algoritmo di controllo di congestione di TCP

Il mittente TCP dimezza la finestra di congestione e imposta il bit **CWR** nell'intestazione del successivo segmento che invia al ricevente.

Chiusura di una connessione TCP