Università degli Studi di Trieste

Corso di Laurea Magistrale in
INGEGNERIA CLINICA

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

Dipartimento di
Ingegneria e Architettura

Reti di calcolatori

Corso di Informatica Medica

Docente: Aleksandar Miladinović

Livello 4 — Trasporto

TCP/UDP, porte ed affidabilità

Obiettivi del livello di trasporto

- Fornire **comunicazione end-to-end tra processi** (non solo tra host).
- **Multiplexing/demultiplexing** tramite **porte**.
- (Con TCP) garantire **affidabilità, ordine, controllo di flusso** su una rete L1–L3 **best-effort**.

Obiettivi del livello di trasporto

- La rete sotto (L1–L3) è **best-effort**: frame/pacchetti possono essere **scartati**, **duplicati** o **arrivare fuori ordine**.
- Serve un livello che offra **comunicazione end-to-end tra processi** (non solo tra host)

- Due strade: **TCP** (affidabile, ordinato) e **UDP** (minimo overhead, nessuna garanzia).

Cenno storico (ARPANET → TCP)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

- Anni ’70: reti a pacchetto **senza affidabilità** “in rete”; l’affidabilità viene spostata agli **endpoint.**
- Problema pratico: **host eterogenei** (mainframe vs microcomputer) con **buffer** molto diversi.
- TCP nasce per:
  - **capire** quanto il destinatario può ricevere (**flow control**),
  - **adattarsi** alla capacità della rete (**congestion control**).

Best-effort: cosa può andare storto

- A **L2** (Ethernet): frame con CRC errato → **scarto** (niente ritrasmissione automatica).
- A **L3** (IP): code piene, **drop**; percorsi diversi → **ri-ordinamento**.
- Risultato: i dati **possono mancare** o **arrivare sfasati** → a sistemarli è **L4** (TCP) o l'applicazione (se usa **UDP**).

Che cosa fa L4, in pratica

- Aggiunge **porta sorgente** e **porta destinazione** al dato dell'applicazione.
- Identifica la **coppia di processi**: IPsrc:PORTsrc → IPdst:PORTdst.
- Due scelte principali:
  - **TCP**: orientato alla connessione, affidabile, in ordine.
  - **UDP**: senza connessione, minimo overhead, nessuna garanzia.

TCP: affidabilità e ordine

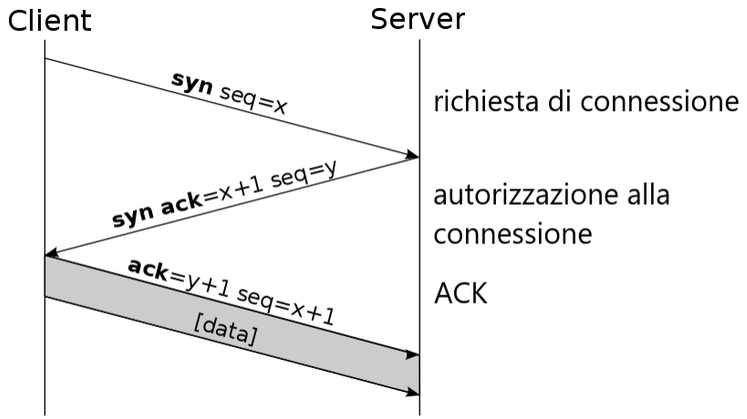
- **3-Way Handshake** apre la connessione; **FIN/ACK** la chiude.
- **Sequence number + ACK cumulativi** → ricostruzione ordine.
- **Ritrasmissione** su timeout o **dup-ACK**.
- **Flow control**: il ricevente annuncia la **finestra (rwnd)** disponibile.

TCP: caratteristiche chiave

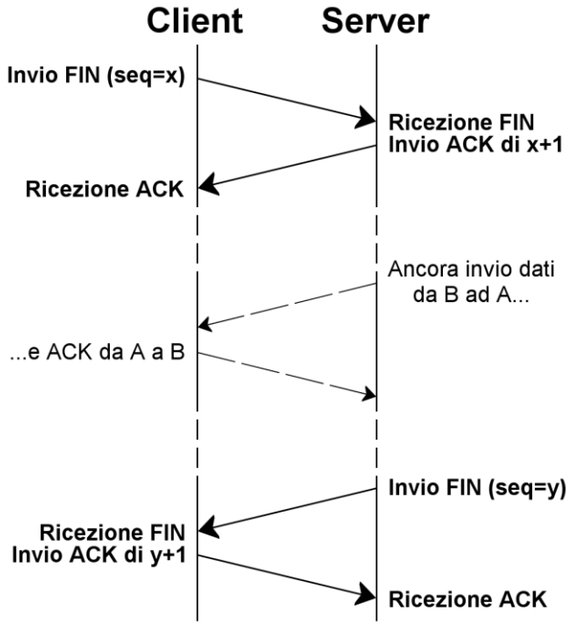
- **Connessione** (stato condiviso), **ordine** (Sequence Number), **affidabilità** (ACK/ritrasmissione).
- **Controllo di flusso** (finestra del ricevente): evita di saturare il buffer dell'altro.
- **(Cenno) Controllo di congestione**: adatta la velocità alla rete reale.

TCP: apertura e chiusura

Apertura (3-Way Handshake)



Chiusura (Four-Way): ogni lato invia **FIN** e riceve **ACK**.



Connessione tra A e B chiusa

TCP: ACK e finestra

- **ACK cumulativi**: confermano “tutto fino a N”.
- **Finestra di ricezione**: quanti byte il ricevente è pronto a ricevere.
- **Ritrasmissione** su **timeout** o **dup-ACK** (perdita rilevata).

Il problema “finestra/buffer” (flow control)

- Host diversi ⇒ **buffer** diversi. Se il mittente spinge troppo, il ricevente **travasa**.
- Soluzione TCP: il ricevente **annuncia rwnd** (bytes liberi).
- Il mittente **non supera** la finestra annunciata.
- **Effettiva velocità** limitata da: **min(rwnd, cwnd)** (vedi slide dopo).

Congestion control: perché serve

- ARPANET (anni ’80): piccoli link saturati → **collassi di congestione**.
- Anche con flow control, la rete può **intasarsi** a metà percorso.
- TCP introduce **meccanismi end-to-end** per limitare il traffico in funzione dello **stato della rete**.

Congestion control: perché serve

- TCP mantiene una **congestion window (cwnd)**, stima “quanto la rete regge”.
- **Slow Start**: parte piccolo (≈1 MSS), **raddoppia** ogni RTT finché non raggiunge **ssthresh**.
- **Congestion Avoidance**: crescita **lineare** (≈ +1 MSS per RTT).
- **Segnale di perdita**:
- **Timeout** ⇒ ssthresh = cwnd/2, cwnd = 1 MSS (riparte in slow start).
- **3 dup-ACK** ⇒ **Fast Retransmit/Recovery**: ssthresh = cwnd/2, cwnd = ssthresh (salta parte della slow start).
- **Finestra effettiva di invio**: min(rwnd, cwnd).

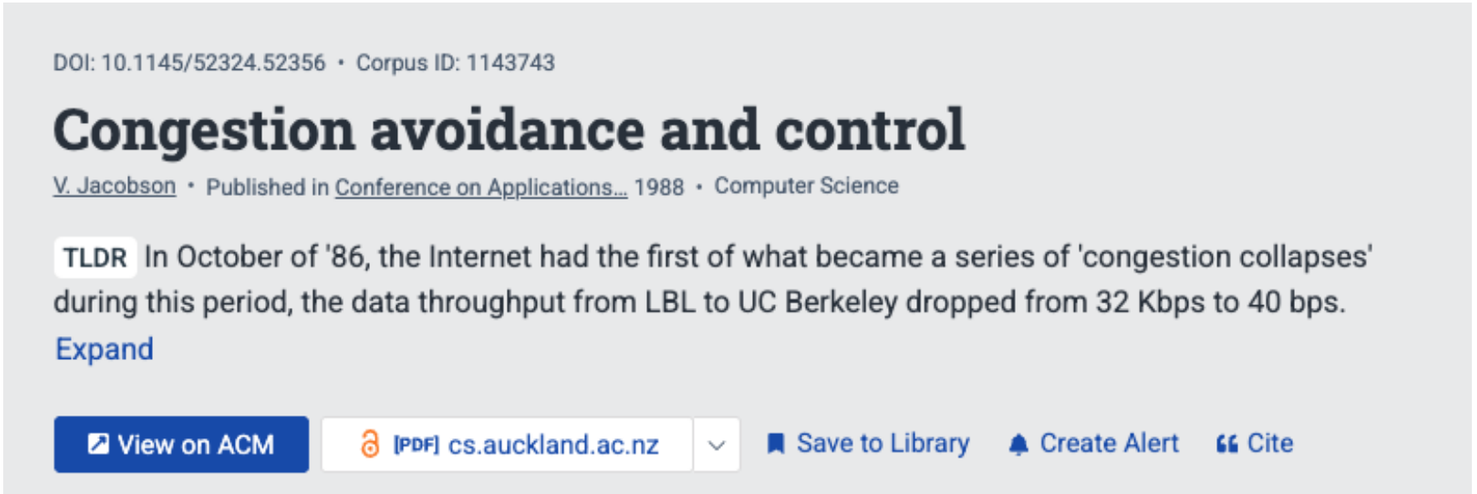
TCP: controllo di flusso vs congestione

- **Flow control (rwnd)**: protegge **il ricevente** (buffer endpoint).
- **Congestion control (cwnd)**: protegge **la rete** (code dei router).
- L'invio massimo è **vincolato** dal **più restrittivo** dei due.

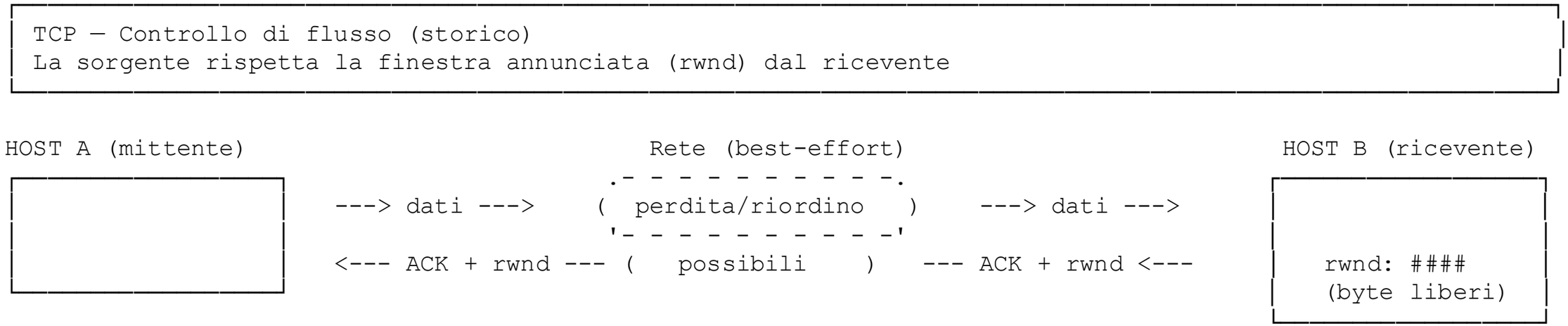
UNIVERSITÀ DEGLI STUDI DI TRIESTE

- **Il controllo di flusso**
  - Uno dei problemi principali dei protocolli di trasporto è **regolare il flusso di dati in base alla capacità del destinatario di ricevere ed elaborare i pacchetti** in ogni dato momento.

- **Il controllo della congestione**
  - Se la finestra di ricezione serve per regolare la velocità di invio in base alla capacità del ricevitore, TCP prevede anche l'uso di una **finestra di congestione per regolare la velocità di trasferimento in base alla capacità della rete**. La finestra di trasmissione effettiva è data quindi dal valore minimo tra le due finestre ed è quella che determina nella pratica la banda utilizzata da TCP.



TCP — Controllo di flusso (versione “storica” con sola rwnd)



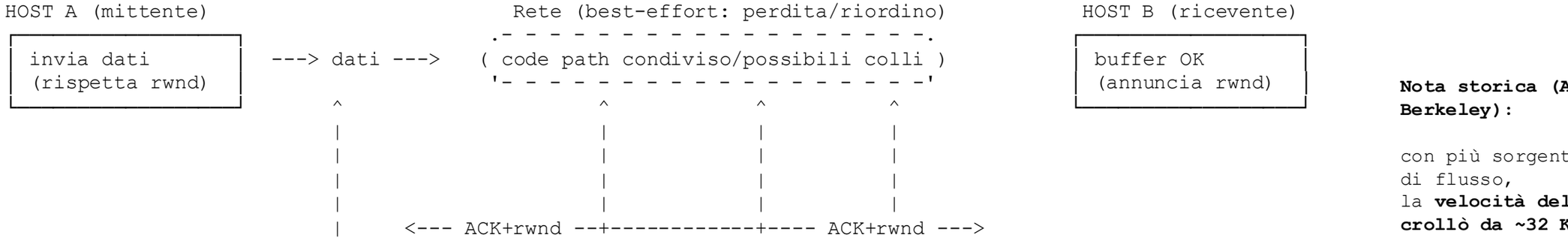
Legenda:
- “dati” : segmenti inviati da A verso B
- “ACK+rwnd”: risposte di B con conferme e finestra disponibile (receive window)

In pratica:
- Il mittente NON supera la finestra annunciata dal ricevente (rwnd).
- Funziona anche su reti best-effort, ma NON evita congestione nella rete.

TCP — Controllo di flusso (storico) e perché NON basta (caso multi-sorgente)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

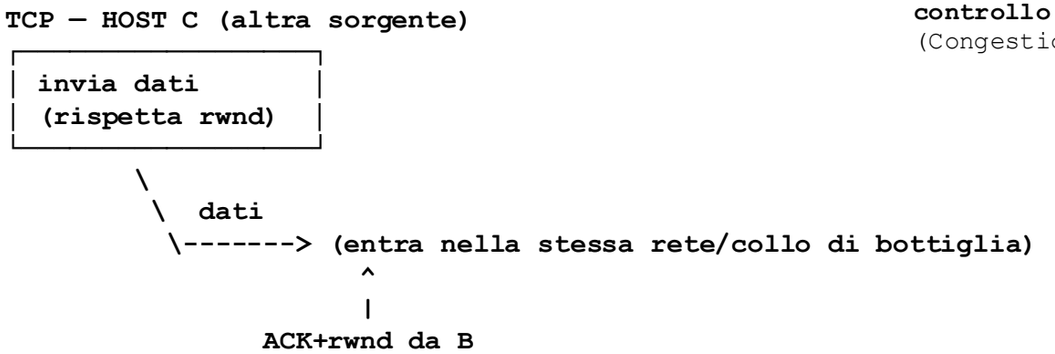
TCP — Controllo di flusso (storico): rwnd funziona… finché non compaiono altre sorgenti
Problema: la rete è best-effort; rwnd regola il buffer del ricevente, NON la capacità di rete



Nota storica (ARPANET, LBL ↔ UC Berkeley):

con più sorgenti e solo controllo di flusso,
la **velocità della connessione crollò da ~32 Kbps a ~40 bps** (!).

Questo portò all'introduzione del **controllo di congestione** in TCP (Congestion Avoidance).



Cosa succede:

- A e C rispettano entrambi rwnd (il ricevente può assorbire dati).
- Ma la rete nel mezzo si congestiona: accodamenti, perdite, riordino, timeouts.
- Solo rwnd NON misura né regola la capacità del percorso ⇒ ritrasmissioni + collasso prestazionale.

TCP — Controllo di congestione + flusso (moderno: cwnd + rwnd)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

```
TCP — Controllo di congestione + flusso
Invio effettivo = min(rwnd, cwnd). Crescita controllata e recupero perdite.
```

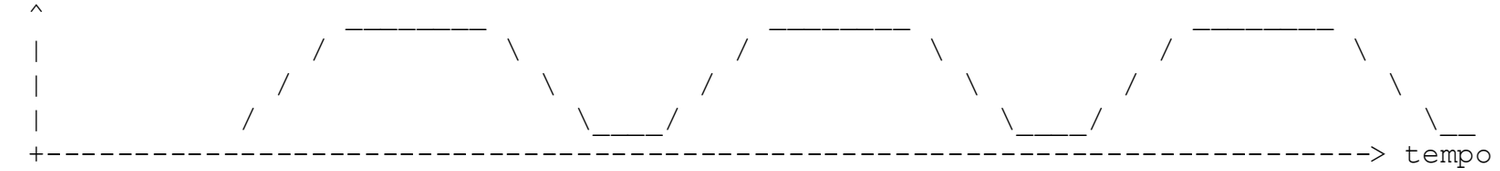
```
HOST A (mittente)                          Rete (best-effort)                          HOST B (ricevente)

 cwnd: ######      ---> dati --->        ( perdita/riordino )    ---> dati --->       rwnd: ####
 (stima capacità                          ( possibili )                              (buffer libero)
 rete)             <--- ACK + rwnd ---                          --- ACK + rwnd <---
```

Invio effettivo = min(rwnd, cwnd)

- rwnd: quanto può ricevere B (buffer disponibile).
- cwnd: quanto "si fida" A della rete (stima capacità senza congestione).

Evoluzione tipica di cwnd (schematica):



- Slow Start: crescita esponenziale finché cwnd < ssthresh.
- Congestion Avoidance: crescita "lineare" (più cauta) sopra ssthresh.
- Perdita (timeout o 3 dup-ACK): Fast Retransmit/Recovery e ssthresh ← cwnd/2.

Perché serve il controllo di congestione

Contesto:

- L4 (TCP) non conosce la banda del percorso né la tecnologia dei link.
  - Potrebbe esserci fibra a 10 Gbit/s, Ethernet a 1 Gbit/s, Wi-Fi, LTE o un modem a 56 kbps.
  - Non esiste un meccanismo standard e affidabile per scambiarsi queste info end-to-end.

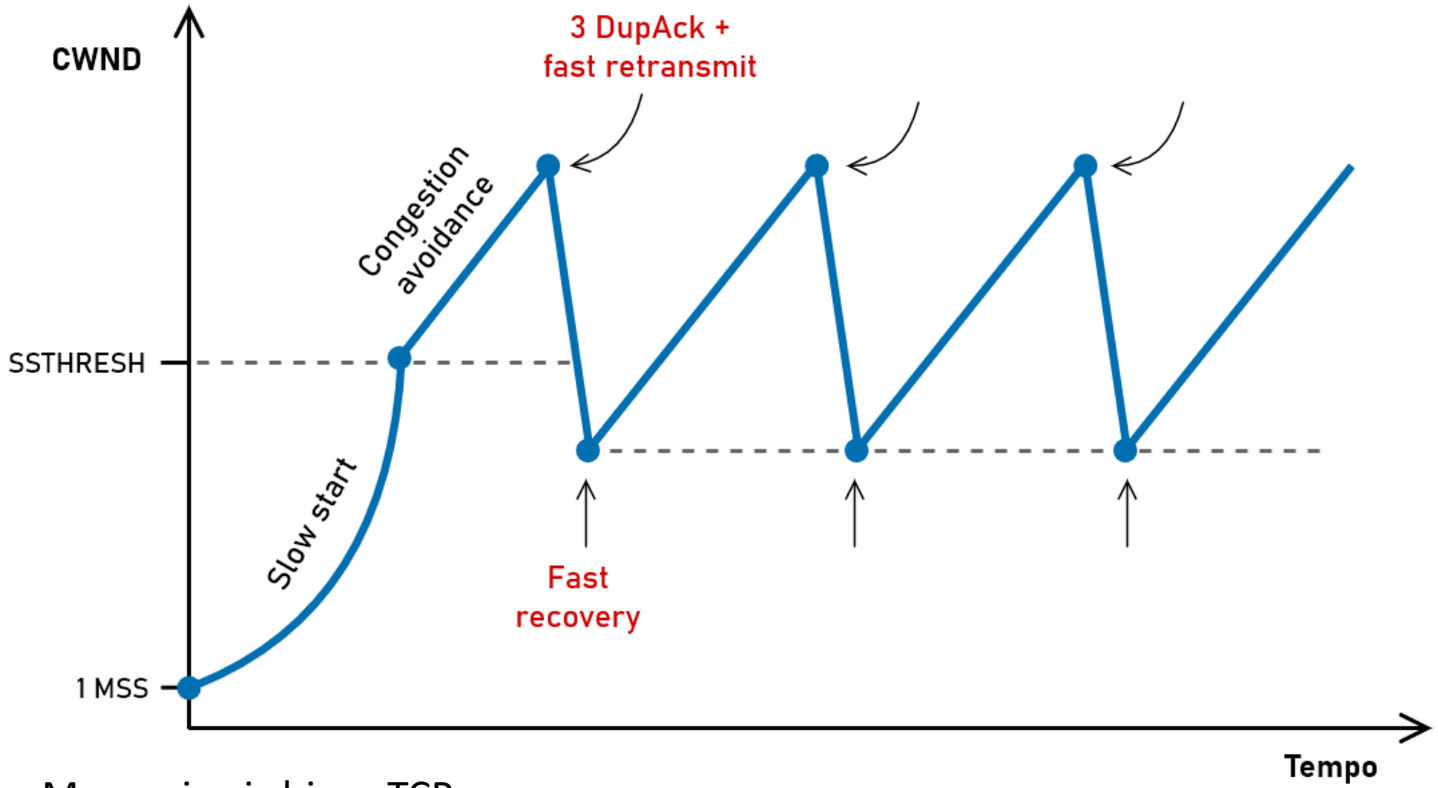
- Anche se ci fosse, sarebbe complicatissimo:
  - Dovresti combinare capacità/ritardi/codici QoS di TUTTI i link sul percorso.
  - Il percorso può cambiare (routing dinamico), e la capacità utile varia nel tempo (code che si riempiono, traffico incostante, buffer e politiche di rete).

- Quindi L4 deve “dedurre” lo stato della rete osservando SOLO i segnali end-to-end:
  - RTT, perdite, ACK duplicati, ECN (se disponibile).

Soluzione (principio “probe & adapt”):

- Spingi (aumenta la velocità/cwnd) finché non vedi segnali di congestione…
- …poi frena (riduci cwnd), e riparti più cauto.
- Invio effettivo = min(rwnd (buffer ricevente), cwnd (stima capacità della rete))

Perché serve il controllo di congestione



Meccanismi chiave TCP:

- Slow Start: crescita esponenziale finché cwnd < ssthresh.
- Congestion Avoidance (AIMD): ↑ additiva, ↓ moltiplicativa al segnale di congestione.
- Fast Retransmit / Fast Recovery: recupero rapido dopo perdite (dup-ACK/timeout).
- Obiettivi: stabilità della rete, equità fra flussi concorrenti, alto throughput senza riempire i buffer.

Esempio (narrato)

- Mittente apre TCP (SYN/SYN-ACK/ACK).
- Ricevente annuncia rwnd = 32 KB.
- Mittente parte in **slow start** con cwnd ≈ 1 MSS, poi raddoppia.
- Appaiono **3 dup-ACK** → **Fast Retransmit** del segmento perso, ssthresh = cwnd/2.
- Prosegue in **Congestion Avoidance** con crescita lineare, sempre ≤ min(rwnd, cwnd).

Cosa succede agli errori tipici

- **Perdita**: ritrasmissione (timeout o dup-ACK).
- **Fuori ordine**: buffer e riordino via sequence/ACK.
- **Ricevente lento**: **rwnd** scende → il mittente **rallenta**.
- **Rete satura**: **cwnd** scende → il mittente **rallenta** (evita il collasso).

Esempio pratico: client → web server


```
Client: 198.51.100.34:50432   →   Server: 203.0.113.10:80 (TCP)
1) SYN → 2) SYN-ACK → 3) ACK      (connessione aperta)
→ GET /...                        (dati in ordine, ACK cumulativi)
← 200 OK ...                      (finestra/ACK regolano il ritmo)
FIN/ACK, ACK                      (chiusura ordinata)
```

UDP (User Datagram Protocol)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

- **Senza connessione**: nessun handshake, nessun riordino/ritrasmissione.
- **Pro**: semplice, **bassa latenza**.
- **Tipico**: DNS query, streaming live, giochi/telemetria (tollerano perdite).

User Datagram Protocol (UDP)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

- **Nessuna connessione**: Prima di inviare i datagrammi, UDP non stabilisce una connessione tra mittente e destinatario, a differenza del TCP, che usa un processo di "handshake" per assicurare che il ricevente sia pronto. Questo rende la trasmissione più veloce.
- **Invio diretto**: Il mittente invia i datagrammi direttamente al dispositivo di destinazione. I datagrammi vengono incapsulati in pacchetti IP per essere instradati sulla rete.
- **Indirizzamento**: Ogni pacchetto UDP contiene un'intestazione (*header*) con i numeri di porta di origine e destinazione, che consentono al sistema operativo di identificare a quale applicazione indirizzare i dati.
- **Nessuna garanzia**: Una volta che un datagramma è stato spedito, UDP non si preoccupa se è stato ricevuto correttamente o meno. Non verifica l'ordine di arrivo dei pacchetti e non richiede una conferma di ricezione (acknowledgment).
- **Controllo degli errori (opzionale)**: Se un datagramma arriva corrotto, può essere scartato, ma non viene rispedito. L'UDP non offre funzionalità di correzione degli errori o di ritrasmissione.

User Datagram Protocol (UDP)

- User Datagram Protocol (UDP) è un protocollo di trasporto che opera nel livello 4 del modello OSI e fa parte della suite di protocolli Internet.
- A differenza del TCP, è un protocollo *senza connessione*, noto per la sua velocità ed efficienza, ma non garantisce l'affidabilità della consegna dei dati.

TCP Segment Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

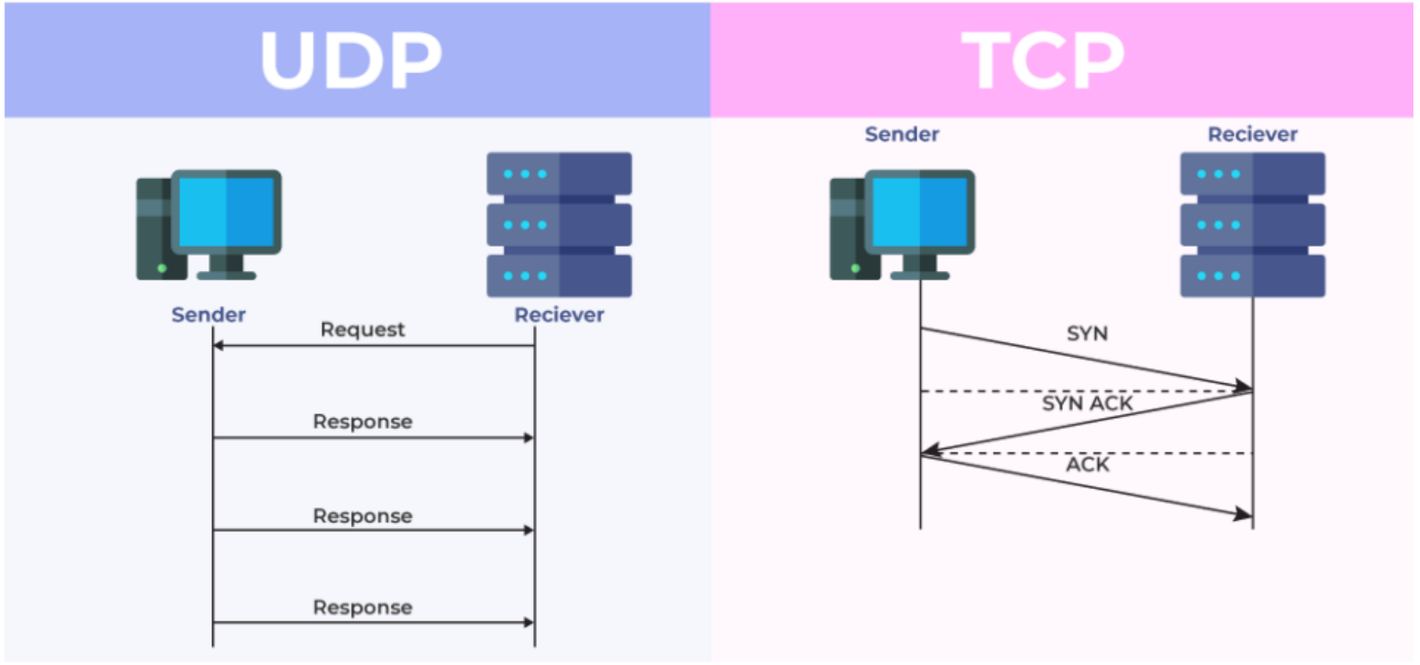
UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Quando TCP, quando UDP

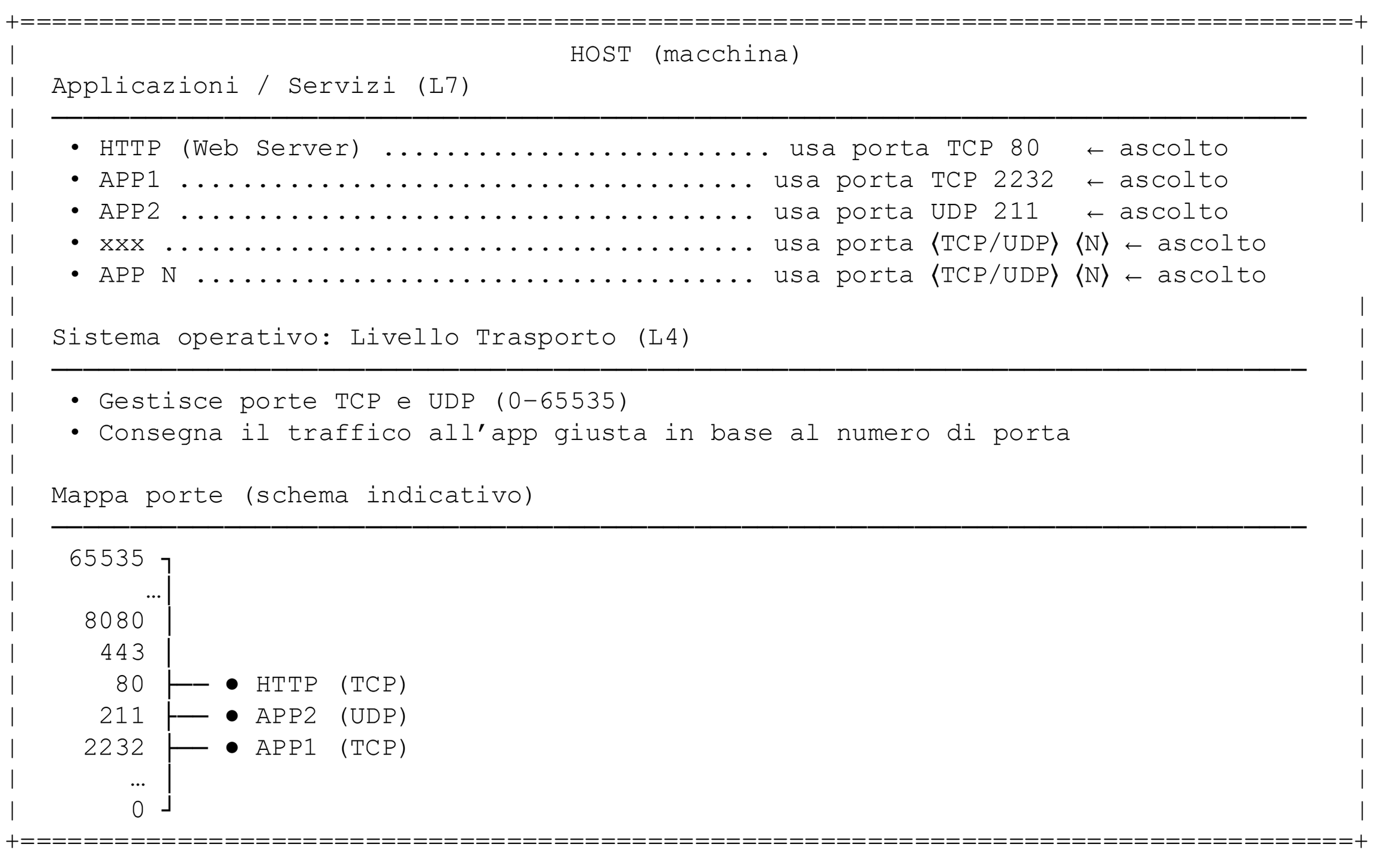
UNIVERSITÀ DEGLI STUDI DI TRIESTE

- **TCP**: web/REST, file transfer, e-mail, DB → servono **integrità e ordine**.
- **UDP**: voce/video live, sensori/telemetria, discovery locale → serve **latenza bassa** più che affidabilità.



Porte (16 bit) e servizi

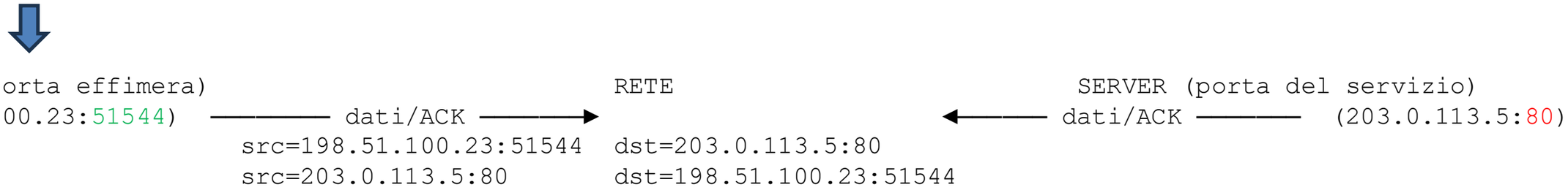
- Intervallo: **1–65535**.
- **Well-known** (1–1023): es. 80 (HTTP), 443 (HTTPS), 22 (SSH), 53 (DNS).
- **Ephemeral**: scelte dal client per connessioni in uscita.
- Notazione comune: IP:porta (es. 203.0.113.42:80).


HOST (macchina)
Applicazioni / Servizi (L7)
• HTTP (Web Server) ..... usa porta TCP 80 ← ascolto
• APP1 ..... usa porta TCP 2232 ← ascolto
• APP2 ..... usa porta UDP 211 ← ascolto
• xxx ..... usa porta ⟨TCP/UDP⟩ ⟨N⟩ ← ascolto
• APP N ..... usa porta ⟨TCP/UDP⟩ ⟨N⟩ ← ascolto
Sistema operativo: Livello Trasporto (L4)
• Gestisce porte TCP e UDP (0–65535)
• Consegna il traffico all’app giusta in base al numero di porta
Mappa porte (schema indicativo)
65535
…
8080
443
80
211
2232
…
0
• HTTP (TCP)
• APP2 (UDP)
• APP1 (TCP)

Socket e flusso bidirezionale

- Una connessione è identificata dalla **quintupla**:
- (IPsrc, PORTsrc, IPdst, PORTdst, protocollo)
- Traffico **bidirezionale**: ogni lato invia/risponde usando **le proprie porte**.

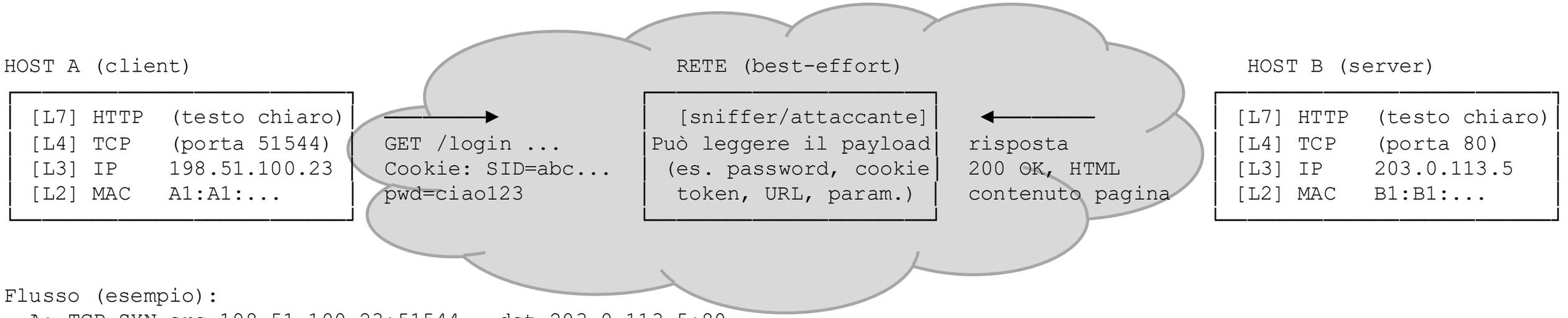
Server, ti chiedo la connessione sulla porta 80. Se vuoi rispondermi, io ascolto sulla porta 51544: manda gli ACK (e i dati) a quella porta che ho scelto!!!



Port number	Process name	Protocol used	Description
20	FTP-DATA	TCP	File transfer—data
21	FTP	TCP	File transfer—control
22	SSH	TCP	Secure Shell
23	TELNET	TCP	Telnet
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP and UDP	Domain Name System
69	TFTP	UDP	Trivial File Transfer Protocol
80	HTTP	TCP and UDP	Hypertext Transfer Protocol
110	POP3	TCP	Post Office Protocol 3
123	NTP	TCP	Network Time Protocol
143	IMAP	TCP	Internet Message Access Protocol
443	HTTPS	TCP	Secure implementation of HTTP

Sicurezza (cenno operativo)

- L4 **non cifra** i dati. La **cifratura** avviene a L7 (es. **HTTPS/TLS** su TCP).
- Alcuni protocolli applicativi usano **UDP + cifratura** a livello app



Flusso (esempio):

A: TCP SYN src=198.51.100.23:51544 → dst=203.0.113.5:80

B: TCP SYN/ACK …

A: HTTP POST /login (Body: user=alice&password=ciao123) ⟶ VISIBILE in CHIARO

B: HTTP 200 OK (Set-Cookie: SID=abc123) ⟵ VISIBILE in CHIARO

Da ricordare

UNIVERSITÀ DEGLI STUDI DI TRIESTE

- **Porte** identificano **processi**; IP identifica **host/interfacce**.
- **TCP**: affidabile, ordinato, con controllo di flusso.
- **UDP**: semplice e veloce, nessuna garanzia.
- L4 **colma** i limiti del sotto-strato **best-effort**.

Glossario essenziale

- **Best-effort**: la rete **non garantisce** consegna/ordine/ritardo.
- **Porta**: numero a 16 bit che identifica un **processo** sul nodo.
- **Socket**: endpoint identificato da IP + porta + protocollo.
- **ACK**: conferma di ricezione (cumulativa in TCP).
- **rwnd**: receive window (buffer disponibile del **ricevente**).
- **cwnd**: congestion window (stima capacità della **rete**).
- **ssthresh**: soglia che separa slow start da congestion avoidance.
- **MSS/MTU**: dimensione massima segmento/trasmissione sul link.
- **dup-ACK**: ACK duplicati (indizio di perdita fuori ordine).