



POLITECNICO
MILANO 1863

Fondamenti di TELECOMUNICAZIONI

Prof. Marco Mezzavilla

Lezione 12 – Livello di Trasporto 3

INDICE

12. LIVELLO DI TRASPORTO 3

1. **Servizio di trasporto**

2. **Protocollo UDP**

Parte I (l'altro ieri)

3. **Trasporto affidabile I**

4. **Trasporto affidabile II**

Parte II (ieri)

5. **Protocollo TCP**

1. Generalità

2. Formato

3. Controllo d'errore

4. Controllo di congestione

Parte III (oggi)

GENERALITÀ

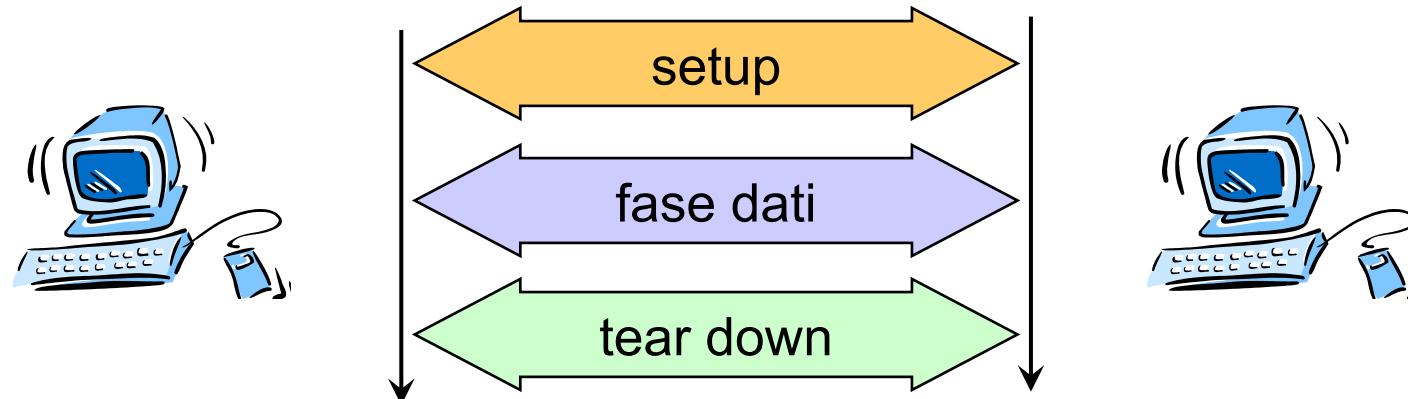
01

Transmission Control Protocol (TCP)

- ❑ Il TCP è un protocollo di trasporto che assicura il **trasporto affidabile**
 - ❑ In corretta **sequenza**
 - ❑ **Senza errori/perdite** dei dati
- ❑ Mediante TCP è possibile costruire applicazioni che si basano sul trasferimento di file senza errori tra host remoti (web, posta elettronica, ecc.)
- ❑ E' alla base della **filosofia originaria** di Internet: servizio di rete semplice e non affidabile, servizio di trasporto affidabile
- ❑ Il TCP effettua anche un **controllo di congestione end-to-end** che limita il traffico in rete e consente agli utenti di condividere in modo equo le risorse

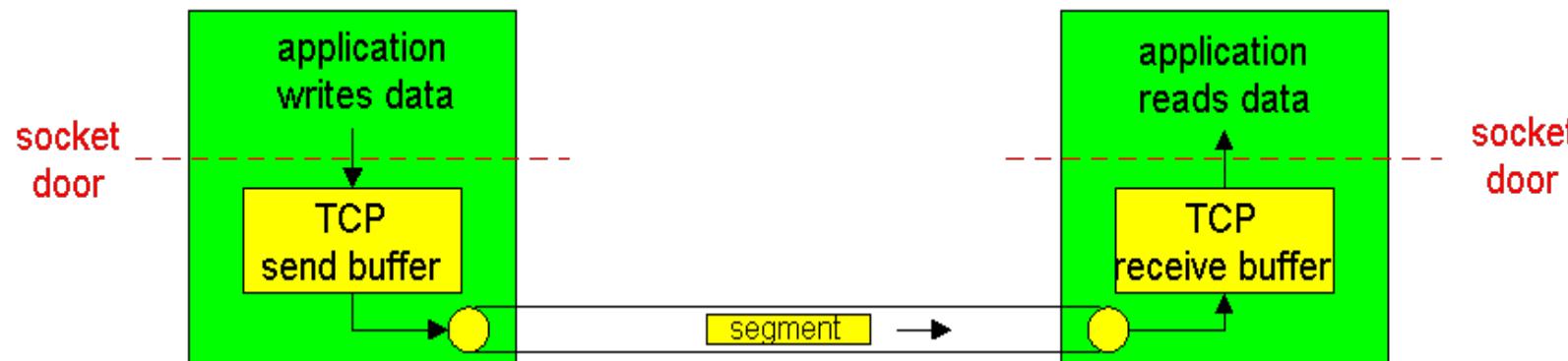
TCP: connection-oriented

- ❑ Il TCP è orientato alla connessione (**connection-oriented**):
 - ❑ Prima del trasferimento di un flusso dati occorre **instaurare** una connessione mediante opportuna segnalazione
 - ❑ Le connessioni TCP sono di tipo **full-duplex** (esiste sempre un flusso di dati in un verso e nel verso opposto, anche se questi possono essere quantitativamente diversi)



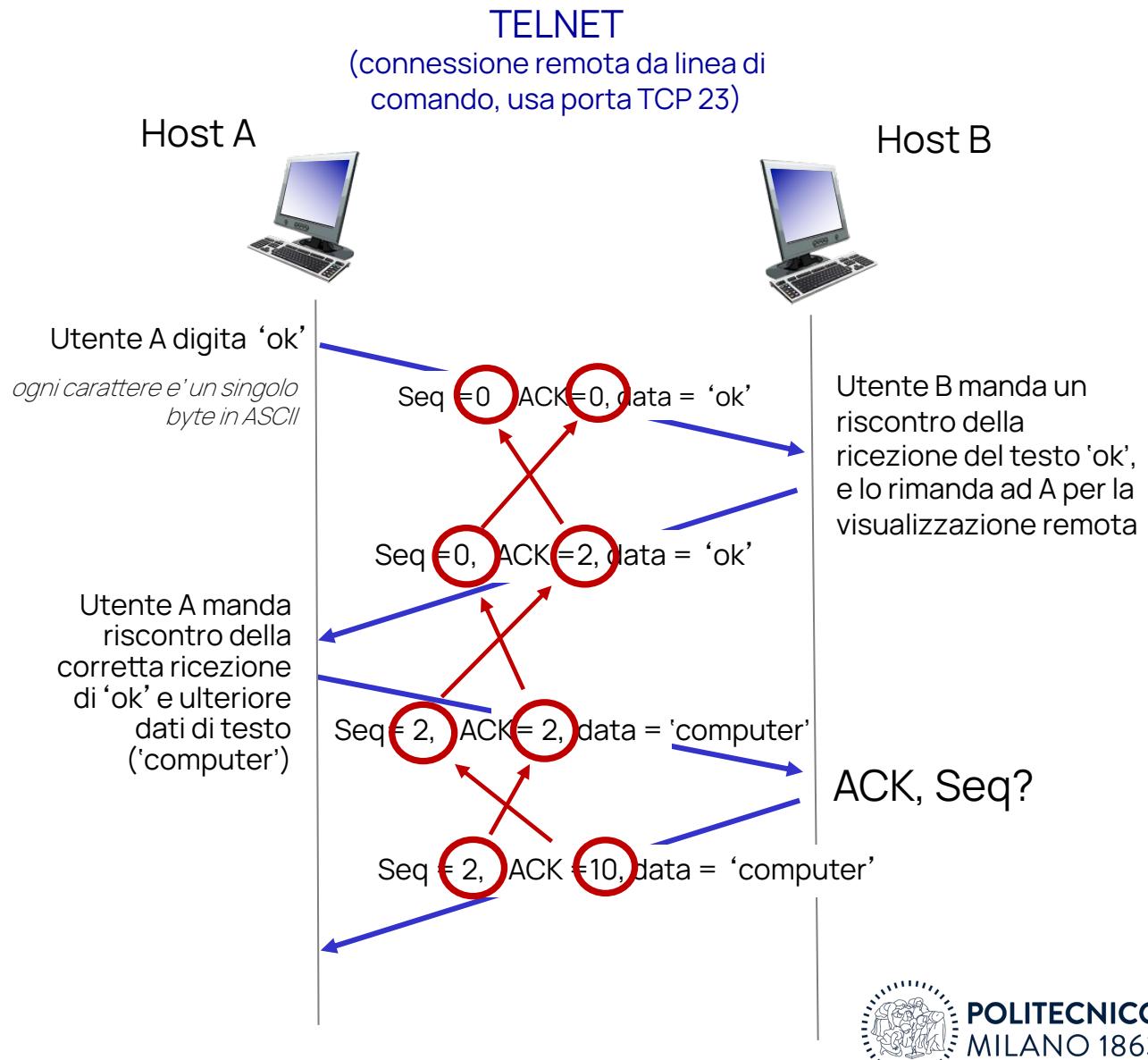
TCP: flusso dati

- Il TCP è orientato alla trasmissione di flussi continui di dati (**stream di byte**)
- Il TCP converte il flusso di dati in **segmenti che possono essere trasmessi in IP**
- Le dimensioni dei segmenti sono **variabili**
- L'applicazione trasmittente passa i dati (byte) a TCP e **TCP li accumula in un buffer**
- Periodicamente, o quando avvengono particolari condizioni, il TCP prende una parte dei dati nel buffer e **forma un segmento**
- La **dimensione del segmento è critica** per le prestazioni, per cui il TCP cerca di attendere fino a che un ammontare ragionevole di dati sia presente nel buffer di trasmissione



TCP: numerazione byte e riscontri

- Il TCP adotta un meccanismo per il controllo delle perdite di pacchetti di tipo **Go-Back-N**
- Sistema di numerazione e di riscontro dei dati inviati
 - TCP numera ogni byte trasmesso, per cui **ogni byte ha un numero di sequenza**
 - Nell'header del segmento TCP è trasportato il numero di sequenza del **primo byte nel segmento stesso**
 - Il ricevitore deve riscontrare i dati ricevuti inviando il numero di sequenza del **prossimo byte che ci si aspetta di ricevere**
 - Se un riscontro non arriva entro un dato timeout, i dati sono ritrasmessi



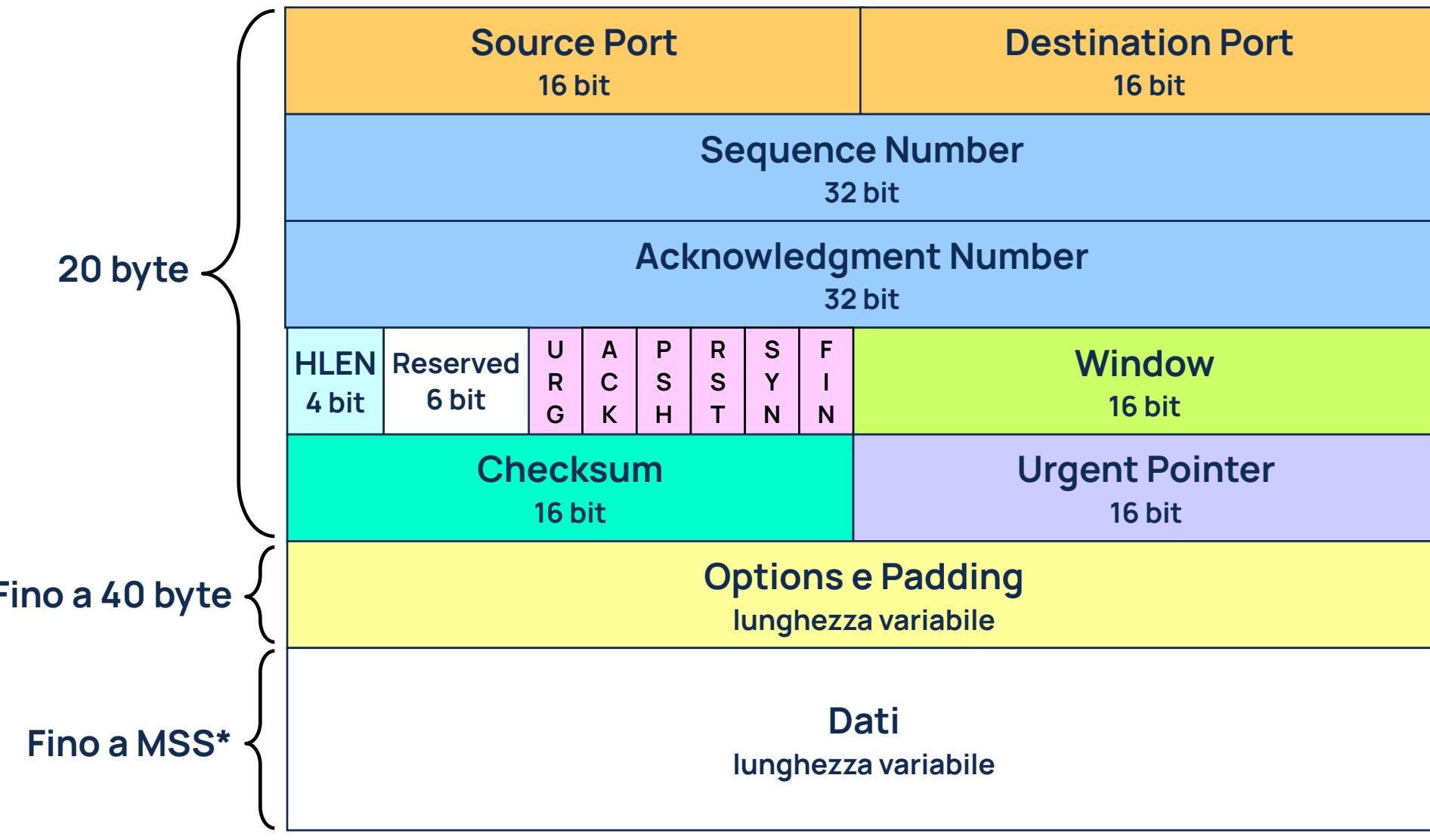
Motivazione per l'uso della numerazione byte

- **Maggiore precisione:** TCP può identificare esattamente quali byte sono stati ricevuti o persi, facilitando la gestione delle ritrasmissioni parziali.
- **Ritrasmissioni efficaci:** In caso di perdita, TCP può ritrasmettere solo i byte mancanti o corrotti, senza dover ritrasmettere segmenti interi che possono includere dati già ricevuti.
- **Controllo della congestione e del flusso:** Il controllo a livello di byte rende più precisa la gestione della finestra di congestione e della finestra di ricezione, permettendo una trasmissione dati più efficiente.

FORMATO

02

Segmento TCP



- *Il TCP MSS (Maximum Segment Size) è un parametro del protocollo TCP che indica la dimensione massima dei dati (payload) che possono essere trasportati in un singolo segmento TCP, escludendo l'header TCP
- L'MSS è basato sulla MTU (Maximum Transmission Unit) della rete, che è la dimensione massima del pacchetto IP (incluso l'header IP) che può essere trasmesso su un link di rete
- L'MSS è calcolato sottraendo la dimensione degli header IP e TCP dalla MTU: $MSS = MTU - 40 \text{ byte}$ (dove 40 byte è la somma degli header IP (20 byte) e TCP (20 byte) standard)
- Ad esempio, su una rete Ethernet con una MTU di 1500 byte, il valore di MSS sarebbe: $1500 - 40 = 1460 \text{ byte}$
- Il valore di default è 536 byte dato che la MTU minima per IPv4 equivale a 576 byte

Header Segmento TCP

- ❑ **Source port, Destination port:** indirizzi di porta sorgente e porta destinazione di 16 bit
- ❑ **Sequence Number:** il numero di sequenza del primo byte nel payload
- ❑ **Acknowledge Number:** numero di sequenza del prossimo byte che si intende ricevere (numero valido solo se flag ACK valido)
- ❑ **HLEN:** contiene la lunghezza complessiva dell'header TCP (20 B base + max 40 B di O&P) espressa in gruppi da 4 B (32 bit) → massimo 60 B (480 bit, 15 gruppi da 32 bit)
 - ❑ Esempio: Header TCP 28 B → HLEN 0111 (ovvero 7, che risulta da $28*8/32$)
- ❑ **Reserved** 6 bit: non utilizzati
- ❑ 6 “**flag**” di valore 1 o 0 per funzioni di *servizio*:
 - ❑ **URG** è 1 se ci sono dati urgenti
 - ❑ **ACK** è 1 se il pacchetto è un ACK valido (l'acknowledge num con numero valido)
 - ❑ **PSH** è 1 quando TX vuole usare il comando di PUSH
 - ❑ **RST** reset, resetta la connessione senza un tear down
 - ❑ **SYN** synchronize, usato durante il setup per comunicare numeri di sequenza iniziali
 - ❑ **FIN** usato per la chiusura esplicita di una connessione
- ❑ **Window:** contiene il valore della finestra di ricezione come comunicato dal ricevitore al trasmettitore
- ❑ **Checksum:** il medesimo di UDP, calcolato in maniera uguale
- ❑ **Options and Padding:** da 0 a 40 byte, da riempire (con multipli di 32 bit) per campi opzionale, per es. durante il setup per comunicare il MSS

Servizi e porte

- La divisione tra porte note, assegnate e dinamiche è la stessa che per UDP
- Alcuni delle applicazioni più diffuse:

22	SSH
21	FTP signalling
20	FTP data
23	telnet
25	SMTP
53	DNS
80	HTTP
110	POP
143	IMAP

CONTROLLO D'ERRORE

02

Controllo d'errore

- Il meccanismo di controllo d'errore del TCP serve a **recuperare pacchetti persi o ritardati eccessivamente** in rete
- La causa della perdita è l'**overflow di una delle code** dei router attraversati a causa della **congestione**
- La causa del ritardo eccessivo è la **lunghezza di una delle code** dei router attraversati a causa della **congestione**
- Il meccanismo di ritrasmissione è di tipo **Go-back-N con Timeout**
- La *finestra di trasmissione* (valore di N) dipende dal meccanismo di **controllo di flusso** e di **congestione**
- L'orologio per la ritrasmissione di un segmento viene inizializzato al momento della trasmissione e determina la ritrasmissione quando raggiunge il valore del Timeout

Gestione del timeout

- ❑ Uno dei problemi è stabilire il **valore ottimo del timeout**:
 - ❑ Timeout **troppo breve** riempirà il canale di ritrasmissioni di segmenti
 - ❑ Timeout **troppo lungo** impedisce il recupero veloce di reali errori
- ❑ Il valore ottimale **dipende fortemente dal ritardo in rete** (rete locale o collegamento satellitare?)
- ❑ Il TCP **calcola dinamicamente** un valore opportuno per il timeout **stimando il RTT** (Round Trip Time)

CONTROLLO DI CONGESTIONE

04

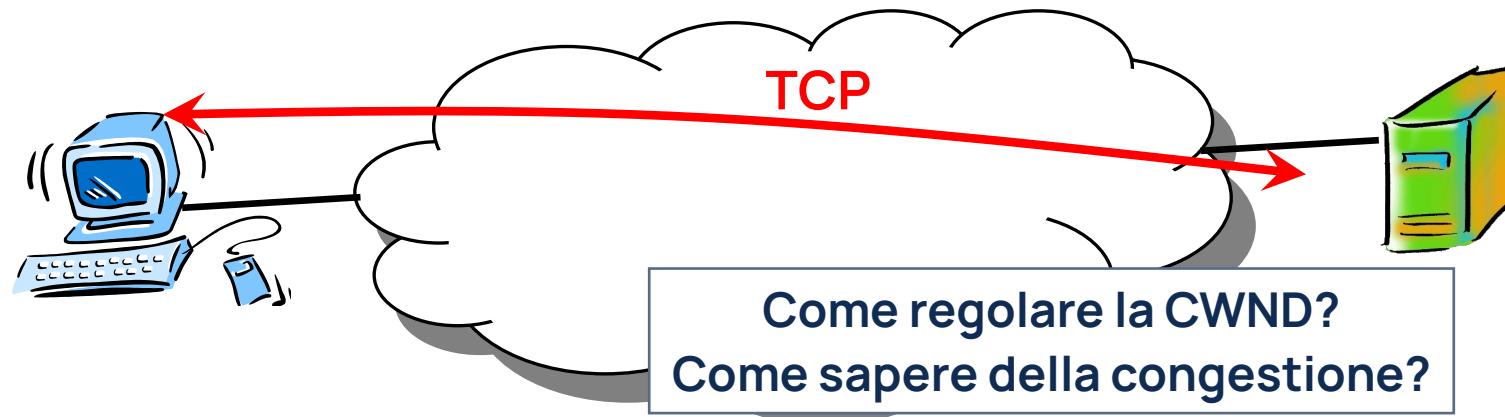
Controllo di congestione

- ❑ Il controllo di flusso (visto ieri)
 - ❑ Dipende solo dalla “**capacità**” del ricevitore
 - ❑ **Non è sufficiente** ad evitare la congestione nella rete
- ❑ Nella rete INTERNET attuale non ci sono meccanismi sofisticati di controllo di congestione a livello di rete (come ad esempio meccanismi di controllo del traffico in ingresso)
- ❑ **Il controllo di congestione è delegato al TCP!!!**
 - ❑ Se il traffico in rete porta a situazioni di congestione il TCP deve ridurre velocemente il traffico in ingresso
- ❑ Essendo il TCP implementato solo negli host il **controllo di congestione è di tipo end-to-end**

Controllo di congestione

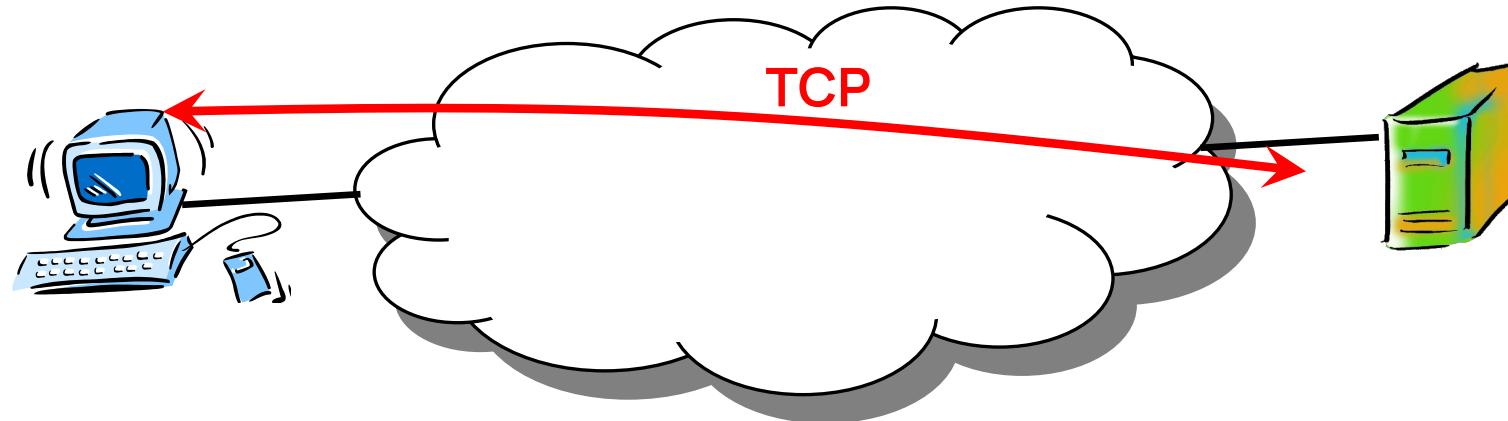
- Il modo più naturale per controllare il ritmo di immissione in rete dei dati per il TCP è quello di **regolare la finestra di trasmissione**
- Il trasmittitore mantiene una **Congestion Window (CWND)** che varia in base agli eventi che osserva (ricezione ACK, timeout)
- Il trasmittitore non può trasmettere più del minimo tra **RCVWND (Receive Window)**, i.e., lo spazio del buffer in ricezione disponibile per ricevere nuovi dati, e CWND

$$\min(CWND, RCVWND)$$



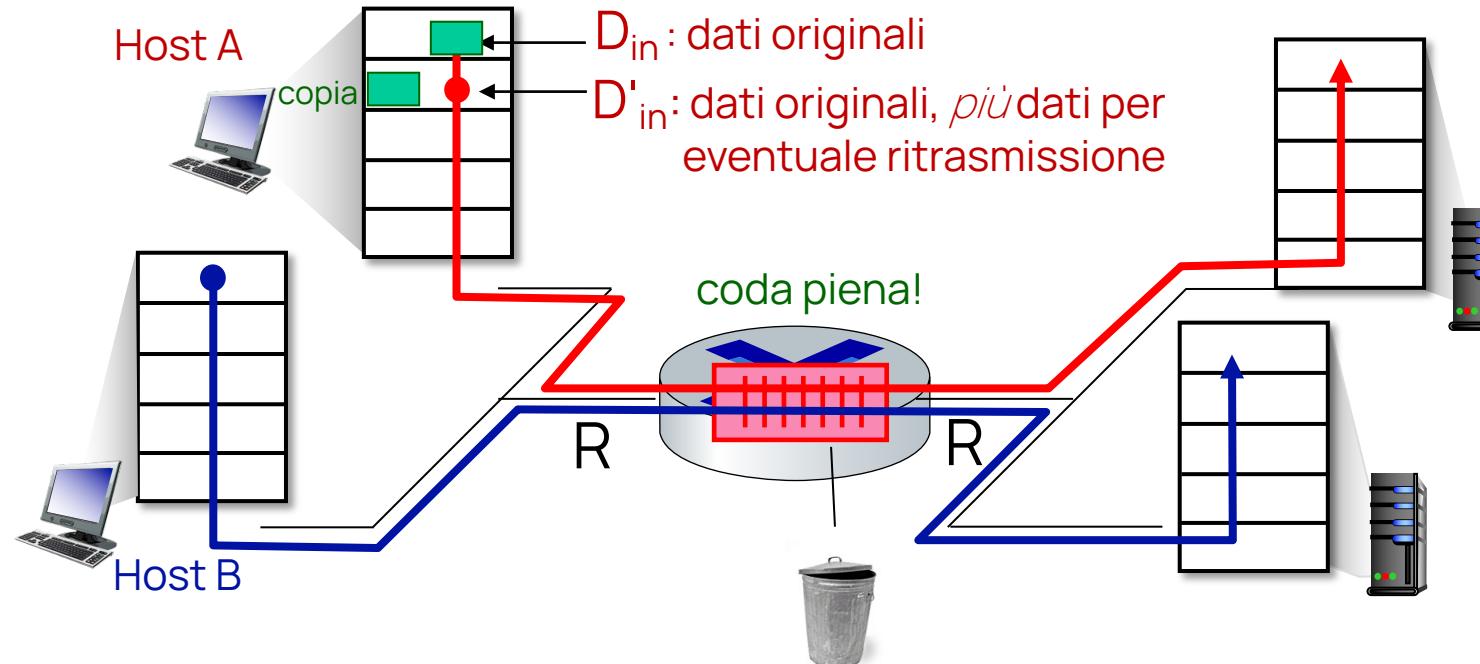
Controllo di congestione

- ❑ L'idea base del controllo di congestione del TCP è quella di interpretare la **perdita di un segmento**, segnalata dallo scadere di un timeout di ritrasmissione, come un **evento di congestione**
- ❑ Come accennato precedentemente, le **cause** sono essenzialmente due:
 - ❑ **Perdita** (coda piena) o **ritardo eccessivo** (coda lunga)
- ❑ La **reazione** ad un evento di congestione è quella di **ridurre la finestra (CWND)**



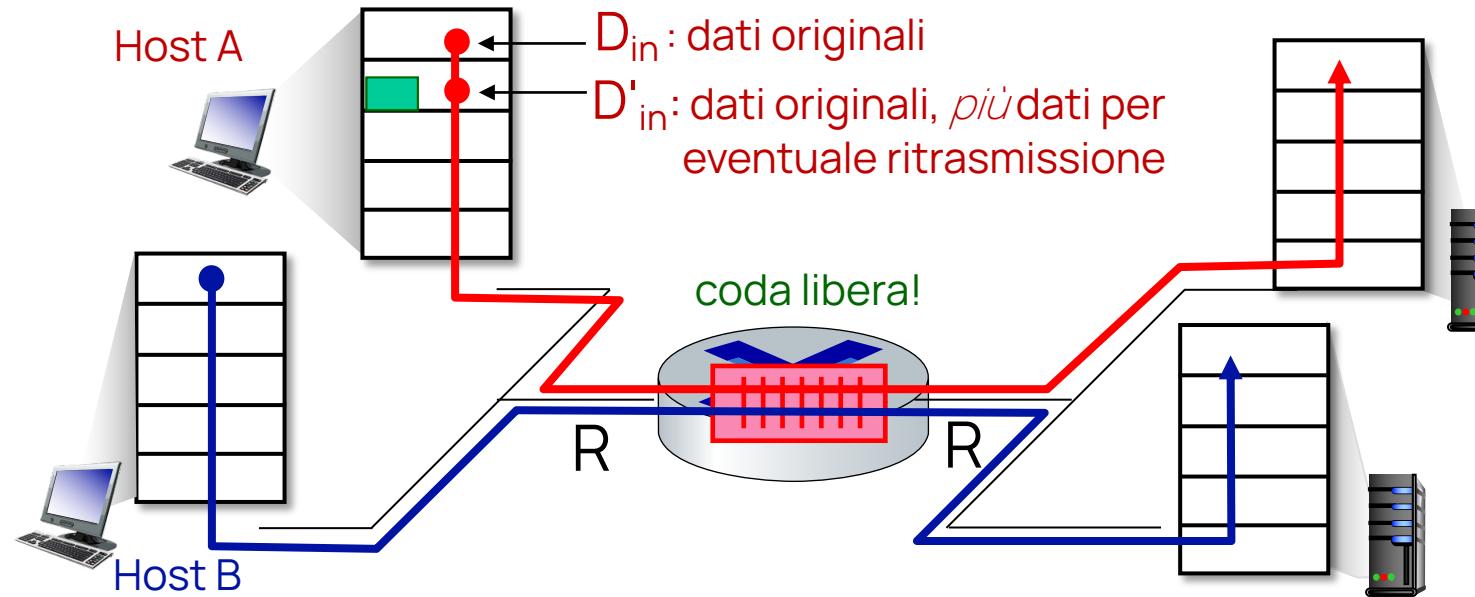
Cause di congestione - perdite

- Perdite di pacchetto dovute alle **code piene** nei routers



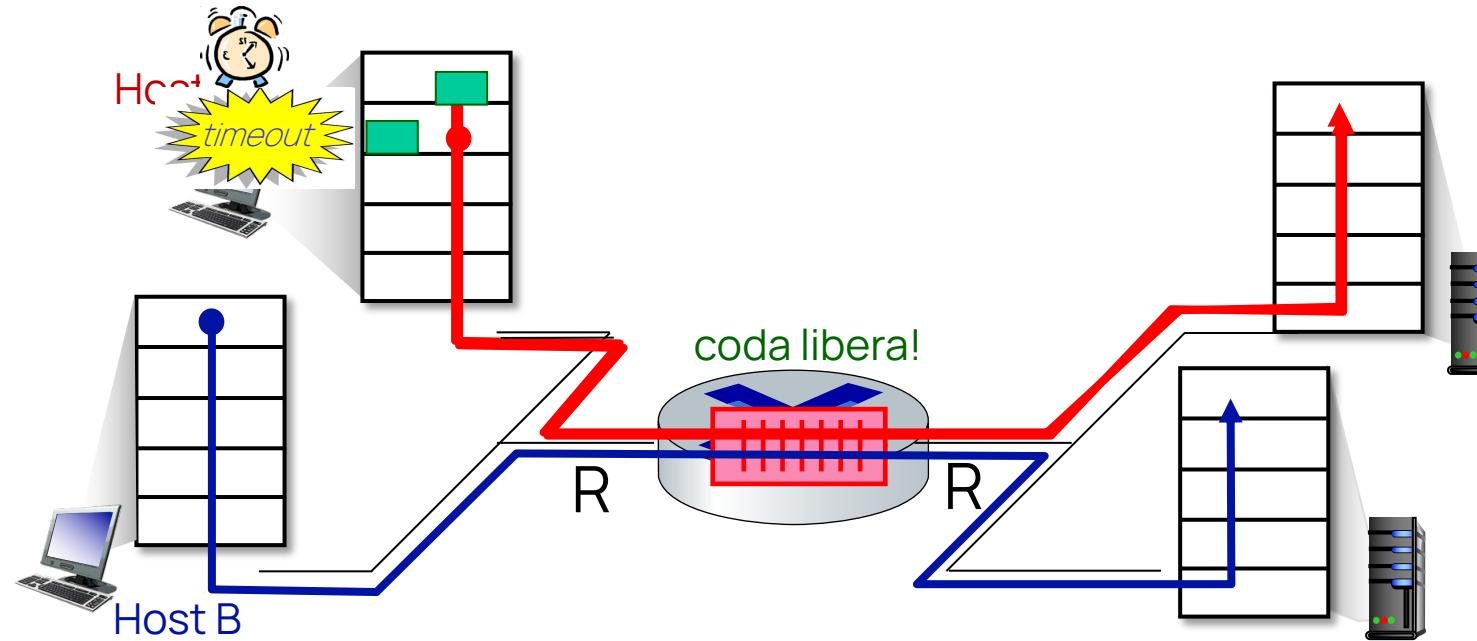
Cause di congestione - perdite

- Ritrasmissione post-perdita



Cause di congestione - ritardi

- Ritardi eccessivi dovuti alle **lunghe code** nei routers (ACK duplicati)



Slow Start & Congestion Avoidance

- ❑ Il valore della finestra **CWND** viene **aggiornato** dal trasmettitore TCP in base ad un **algoritmo**
- ❑ Il modo in cui avviene l'aggiornamento **dipende dalla fase (o stato)** in cui si trova il trasmettitore
- ❑ Esistono due fasi fondamentali:

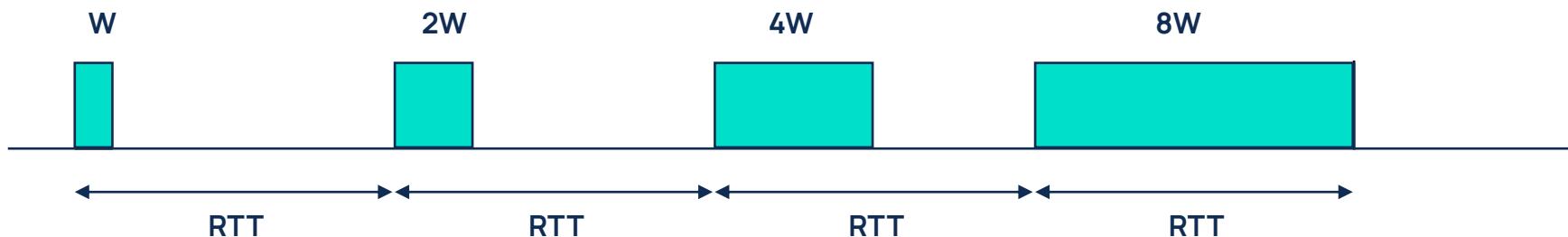


- ❑ **Slow Start**
- ❑ **Congestion Avoidance**

- ❑ La variabile **SSTHRESH** è mantenuta al trasmettitore per distinguere le due fasi:
 - ❑ se $\text{CWND} < \text{SSTHRESH}$ si è in **Slow Start**
 - ❑ se $\text{CWND} > \text{SSTHRESH}$ si è in **Congestion Avoidance**

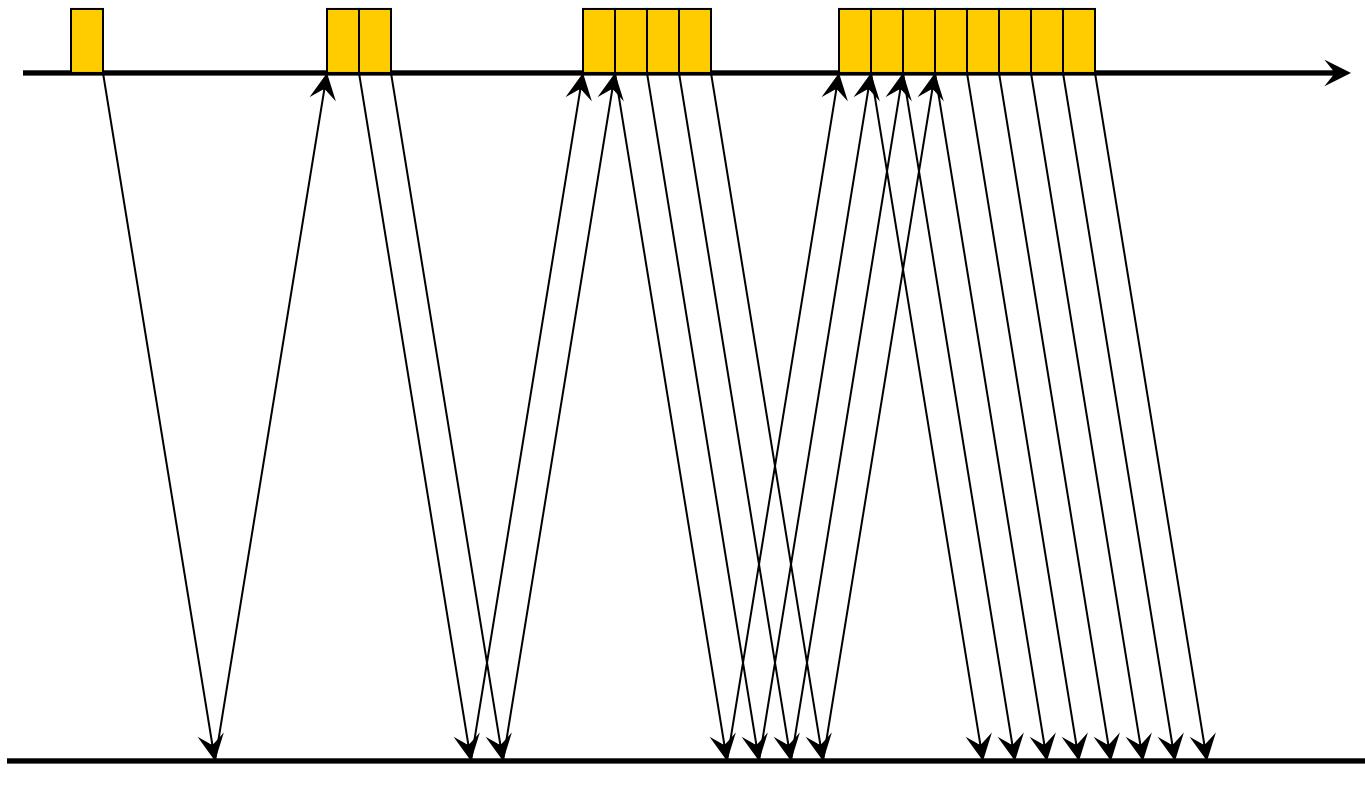
Slow Start

- All'inizio, il trasmettitore pone la **CWND** a 1 segmento (MSS) e la **SSTHRESH** ad un **valore di default molto elevato**
- Essendo **CWND < SSTHRESH** si parte in **Slow Start**
- In Slow Start, la **CWND** viene **incrementata di 1 per ogni ACK ricevuto**
 - Si invia un segmento e dopo RTT si riceve l'ACK
 - Si pone **CWND a 2**, si inviano **2 segmenti**, si ricevono **2 ACK**
 - Si pone **CWND a 4**, si inviano **4 segmenti**, si ricevono **4 ACK**
 - ...



Slow Start

- Al contrario di quanto il nome faccia credere **l'incremento** della finestra avviene in modo **esponenziale** (raddoppia ogni RTT)
- È comunque considerato lento (*cauto*) perché non parte ad un rate alto, parte appunto da 1 per poi crescere gradualmente



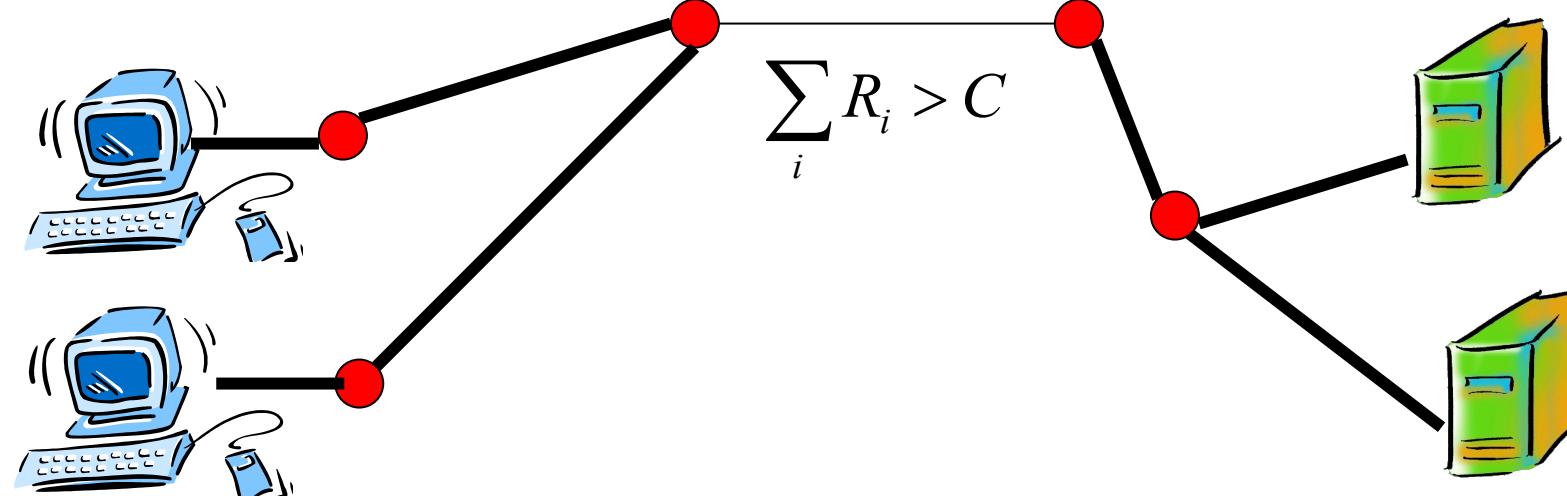
Slow Start

- ❑ L'incremento può andare avanti fino a che
 - ❑ Si riscontra il primo evento di **congestione**, oppure
 - ❑ **CWND > STHRESH**
 - ❑ **CWND > RCWND**
 - ❑ Ricordiamo infatti che la finestra di trasmissione è data da $\min(CWND, RCVWND)$
- ❑ Insieme alla finestra aumenta il ritmo (o rate) di trasmissione che può essere stimato come:

$$R = \frac{CWND}{RTT} \left[\frac{\text{bit}}{\text{s}} \right]$$

Evento di congestione

- Un evento di congestione si verifica quando il **ritmo di trasmissione porta in congestione** un link sul percorso in rete verso la destinazione
- Un link è congestionato quando la **somma dei ritmi di trasmissione** dei flussi che lo attraversano è **maggiori della sua capacità**



Evento di congestione

- ❑ Scade un timeout di ritrasmissione
 - ❑ Il TCP reagisce ponendo **SSTHRESH uguale alla metà dei “byte in volo”** (byte trasmessi ma non riscontrati); più precisamente

$$SSTHRESH = \max\left(2MSS, \frac{FlightSize}{2}\right)$$

valore minimo per STHRESH

se 'FlightSize', ovvero CWND al momento della scadenza del timeout, è dispari, arrotondiamo per difetto

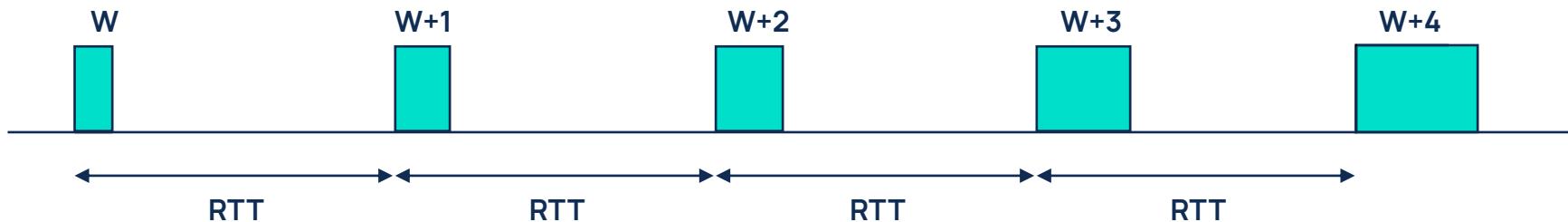
- ❑ E ponendo **CWND a 1 MSS**
- ❑ Si noti che di solito i “byte in volo” sono con buona approssimazione pari a CWN

Evento di congestione

- ❑ Come risultato:
 - ❑ **CWND < SSTHRESH** e si entra nella fase di **Slow Start**
 - ❑ Il trasmettitore invia un segmento e la sua CWND è incrementata di 1 ad ogni ACK
- ❑ Il trasmettitore trasmette tutti i segmenti a partire da quello per cui il timeout è fallito (**politica Go-Back-N**)
- ❑ Il valore a cui è posta la **SSTHRESH** è una **stima della finestra ottimale** che eviterebbe futuri eventi di congestione

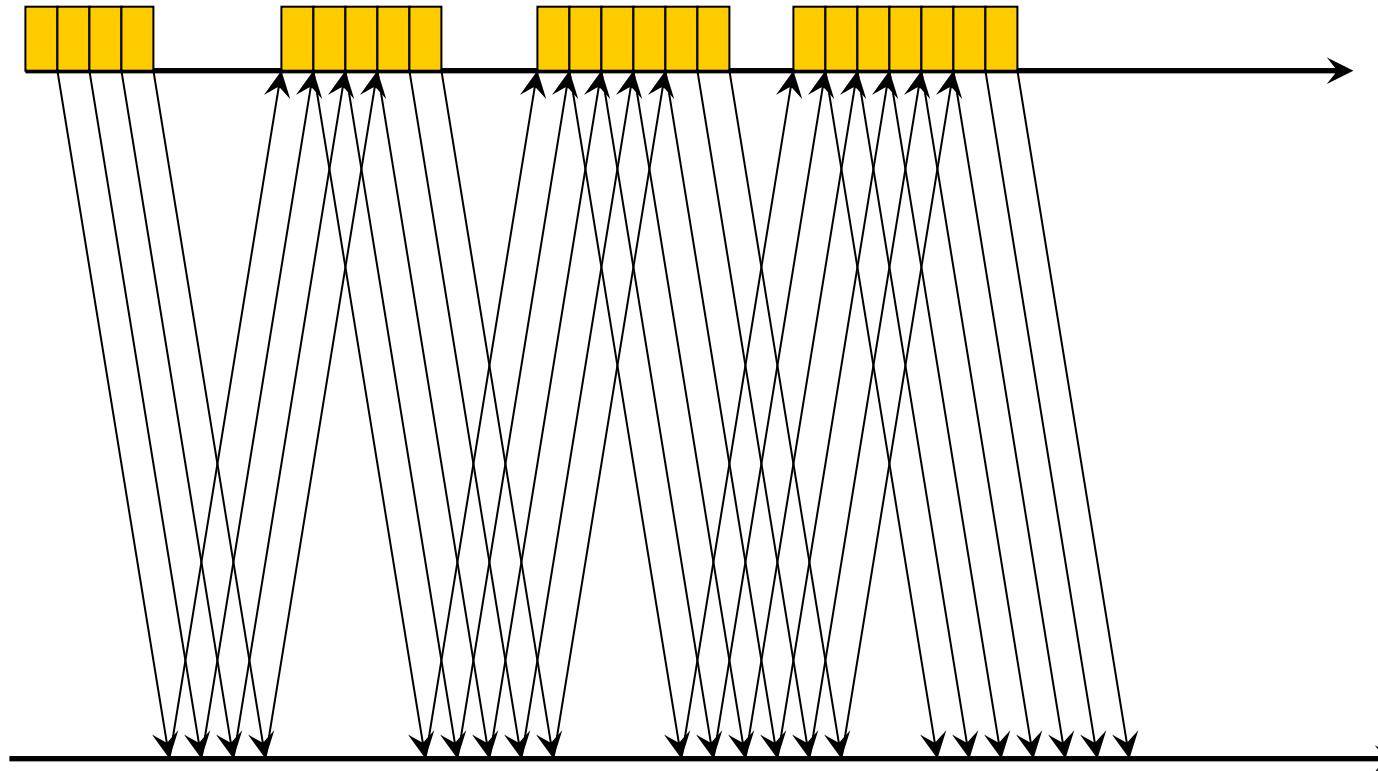
Congestion Avoidance

- ❑ Lo slow start continua fino a che **CWND diventa grande come STHRESH** e poi parte la fase di Congestion Avoidance
- ❑ Durante il Congestion Avoidance:
 - ❑ **Si incrementa la CWND di 1/CWND ad ogni ACK ricevuto**
- ❑ Se la CWND consente di trasmettere N segmenti, la ricezione degli ACK relativi a tutti gli N segmenti porta la CWND ad aumentare di 1 segmento
- ❑ In Congestion Avoidance si attua un **incremento lineare della finestra di congestione**

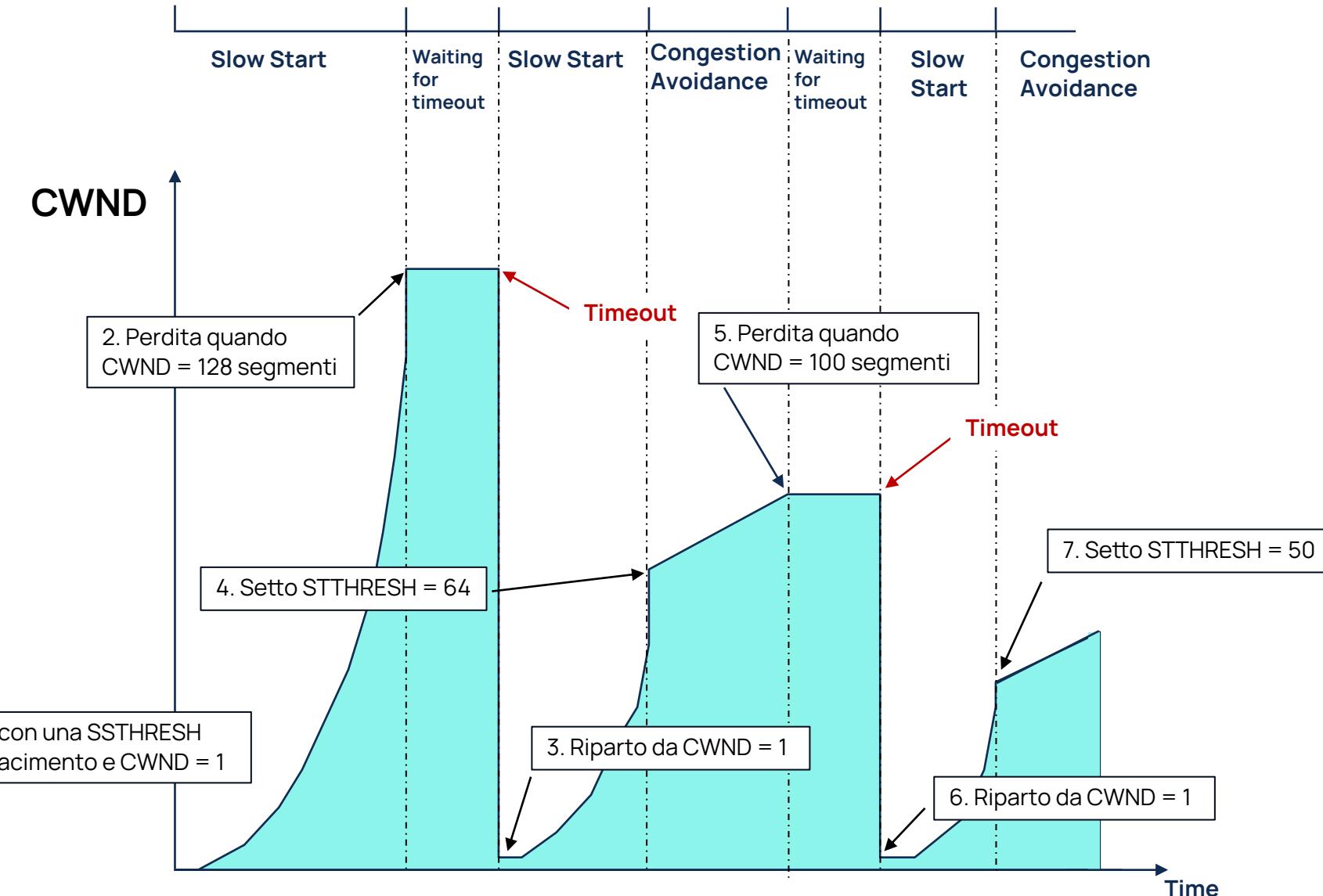


Congestione Avoidance

- Dopo aver raggiunto STHRESH la finestra continua ad aumentare ma molto più lentamente



Esempio di funzionamento



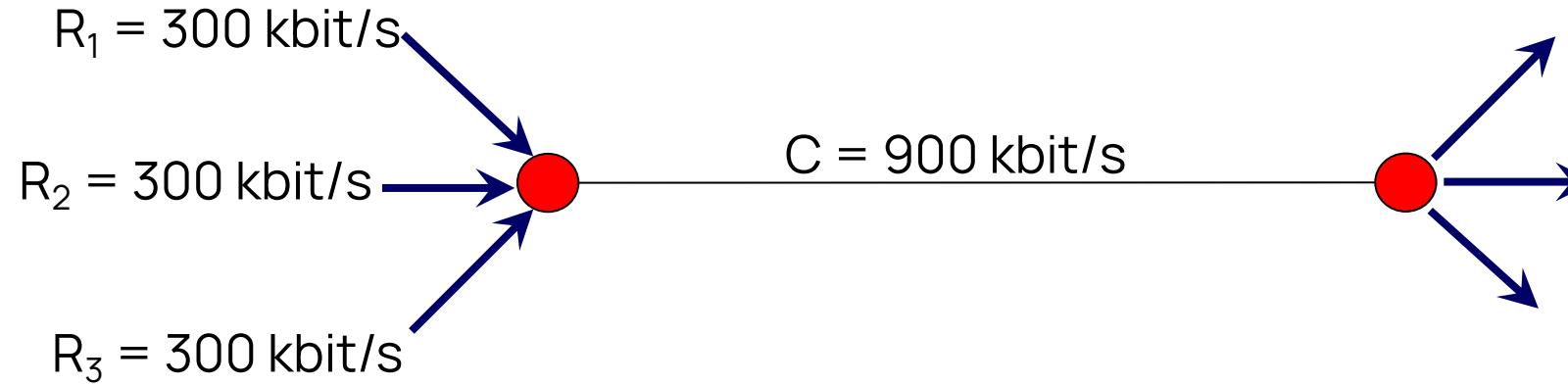
Condivisione equa delle risorse

- ❑ In condizioni ideali il meccanismo di controllo del TCP è in grado di
 - ❑ Limitare la congestione in rete
 - ❑ Consentire di dividere in modo equo la capacità dei link tra i diversi flussi
- ❑ Le condizioni ideali sono alterate tra l'altro da
 - ❑ Differenti RTT per i diversi flussi
 - ❑ Dimensione dei Buffer

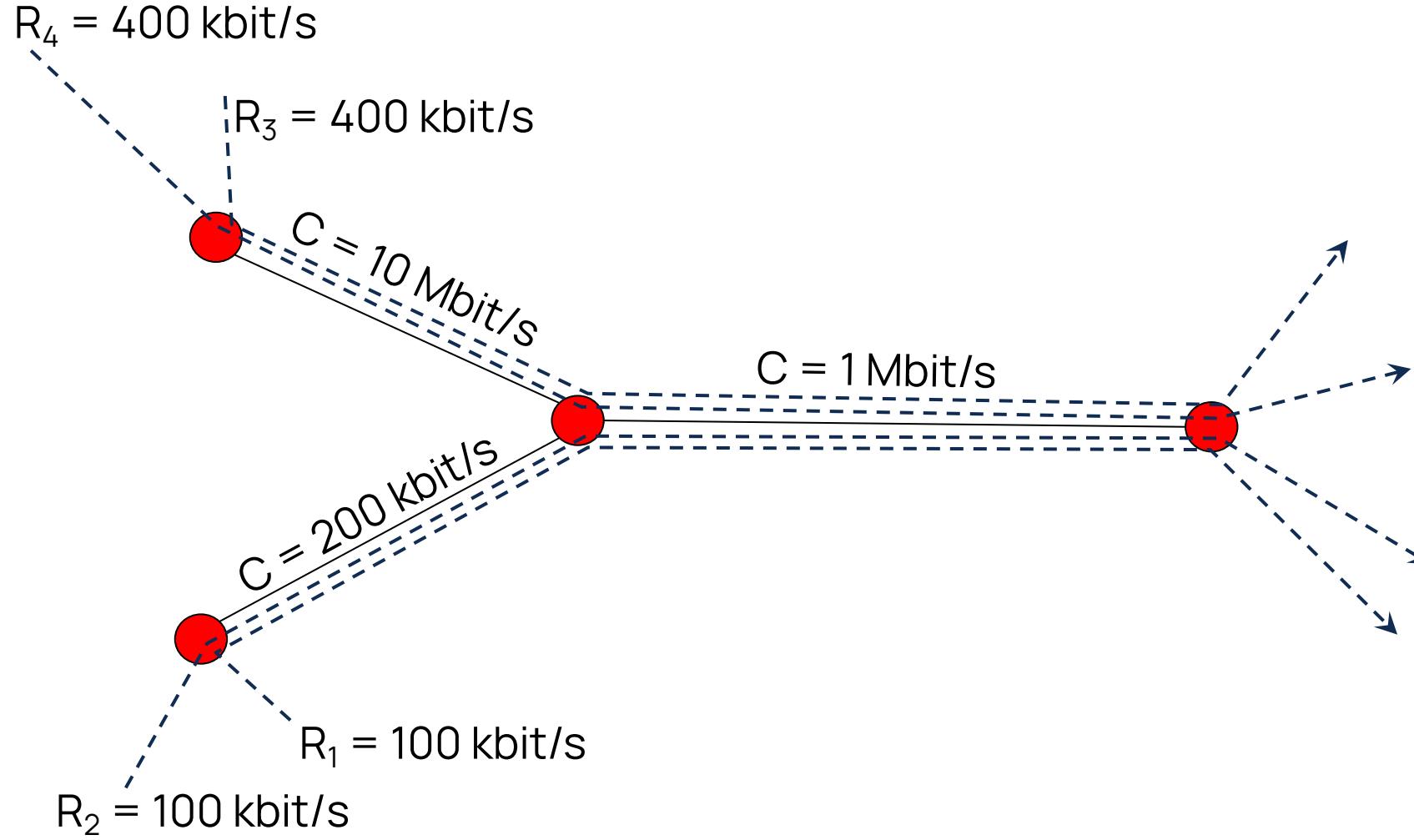


Condivisione equa delle risorse

- ❑ I valori dei rate indicati sono solo valori medi e valgono solo in condizioni ideali
- ❑ Il ritmo di trasmissione in realtà cambia sempre e in condizioni non ideali la condivisione può non essere equa



Condivisione equa delle risorse

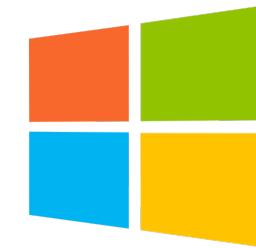


TCP e sistemi operativi

- ❑ Esistono diverse versioni di protocollo TCP
- ❑ TCP è implementato nel sistema operativo
- ❑ Versione base è TCP **Tahoe** (versione presentata qui)

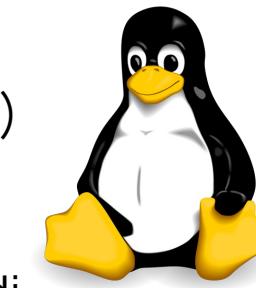
❑ Windows

- ❑ In passato **Compound TCP**; a partire da Windows 10 e Windows Server 2016 **TCP CUBIC** di default



❑ Linux

- ❑ **TCP CUBIC** di default, **TCP BBR** o **Reno** (a discrezione dell'utente)



❑ Apple

- ❑ Prima protocolli proprietari, per esempio **MacTCP**, poi **NewReno** di default, ora stanno integrando **TCP CUBIC**



Esempio

- ❑ Ogni lato della connessione TCP (cioè il dispositivo Windows e il dispositivo Apple) gestisce in modo indipendente il proprio controllo della congestione:
 - ❑ Se il terminale Windows usa l'algoritmo **CUBIC** (algoritmo di default per Windows 10), regolerà la sua finestra di congestione secondo le regole di **CUBIC**, basate su una crescita più aggressiva in reti ad alta capacità.
 - ❑ Se il terminale Apple usa invece **NewReno** o **BBR** (che Apple sta implementando su alcuni dispositivi), la sua CWND crescerà o si ridurrà secondo le regole di quell'algoritmo
- ❑ Quindi, è possibile e comune che i due dispositivi utilizzino algoritmi di controllo della congestione differenti nella stessa connessione TCP.
- ❑ Questo non causa problemi alla connessione, perché ogni lato gestisce solo il proprio invio di dati e non interferisce con la gestione della congestione dell'altro.



Fondamenti di TELECOMUNICAZIONI

Prof. Marco Mezzavilla
marco.mezzavilla@polimi.it