Lezione 10 - Livello di Trasporto

Trasmissione dati affidabile vs Controllo della Congestione

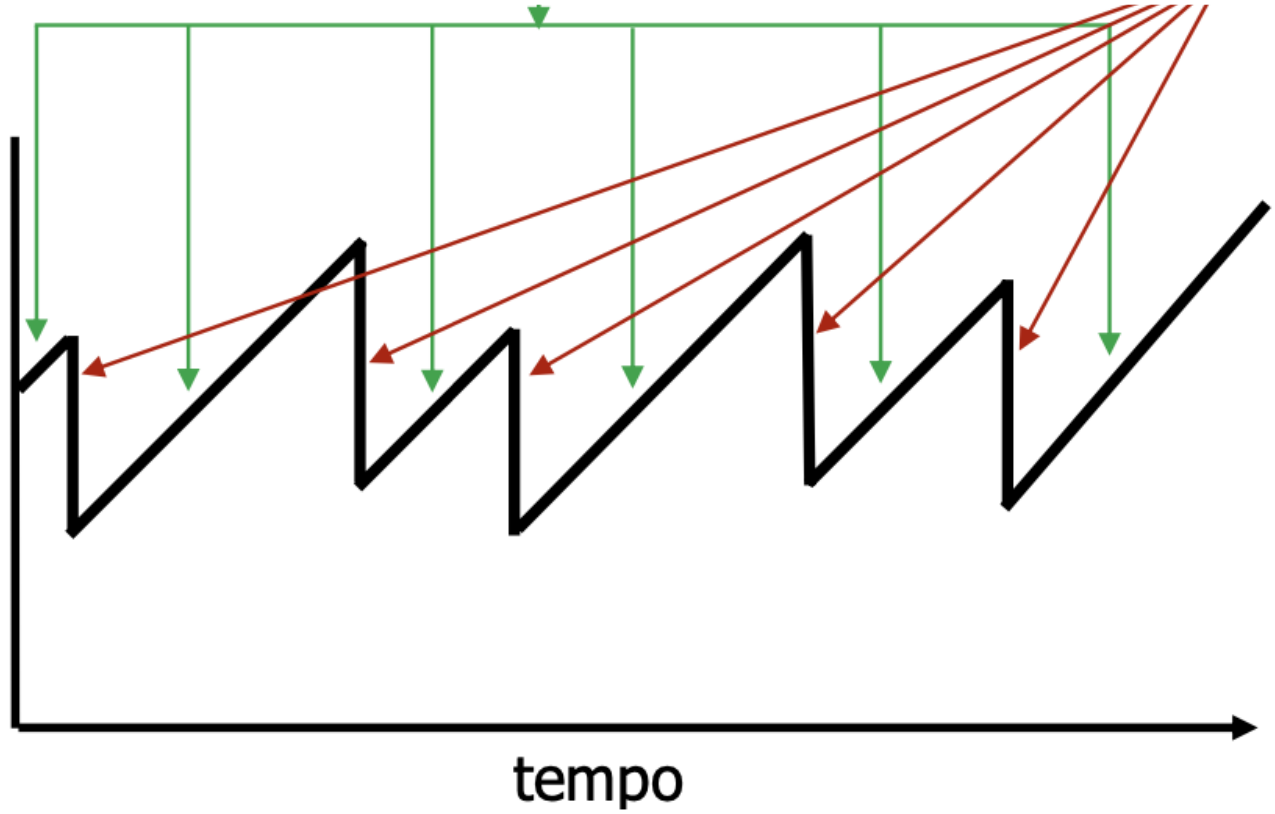
- Trasmissione dati affidabile
  - reagisce alla perdita (e alla corruzione) dei pacchetti, possibilmente causata dalla congestione
  - "tratta i sintomi della congestione"
- Controllo della congestione
  - "cura la malattia"
  - evita che la malattia si aggravi fino a degenerare fino allo scenario di "collasso di congestione"

Controllo di congestione TCP : incremento additivo e decremento moltiplicativo (AIMD)

Approccio : i mittenti possono aumentare il tasso di invio, fino a quando non si verifica la perdita di pacchetti (congestione), quindi diminuire la velocità di invio in caso di perdita.

Incremento additivo: aumentare la velocità di invio di 1 MSS ogni RTT fino a quando non viene rilevata una perdita
Decremento moltiplicativo : dimezzare la velocità di invio ad ogni evento

di perdita



TCP AIMD : di più

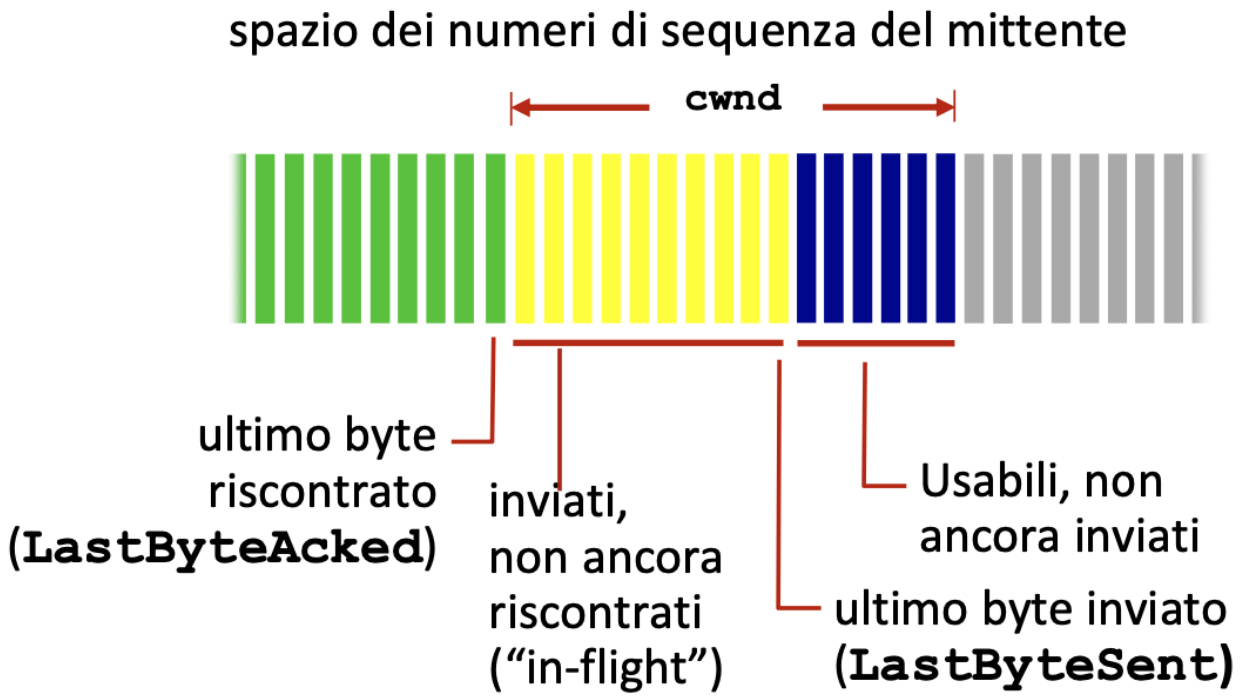
Dettaglio decremento moltiplicativo : In TCP Reno, il mittente riduce il tasso di invio in risposta a eventi di perdita:

- dimezzamento in caso di perdita rilevata da un triplo ACK duplicato (passando poi– per beve tempo – nella fase di fast recovery)
- taglio a 1 MSS ("maximum segment size") quando la perdita è rilevata dal timeout (ritornando poi nella fase di slow start)
  Una versione precedente, detta TCP Tahoe, la risposta a qualsiasi evento di perita era il taglio a 1 MSS e il passaggio a slow start

Perché AIMD?

- AIMD – un algoritmo asincrono, distribuito– è stato dimostrato che:
  - ottimizza i flussi congestionati in tutta la rete!
  - ha proprietà desiderabili di stabilità

Controllo della congestione TCP : dettagli



- Il mittente limita la trasmissione: $LastByteSent - LastByteAcked \leq cwnd$
- *cwnd* viene regolata dinamicamente in risposta alla congestione della rete osservata (implementando il controllo della congestione TCP)

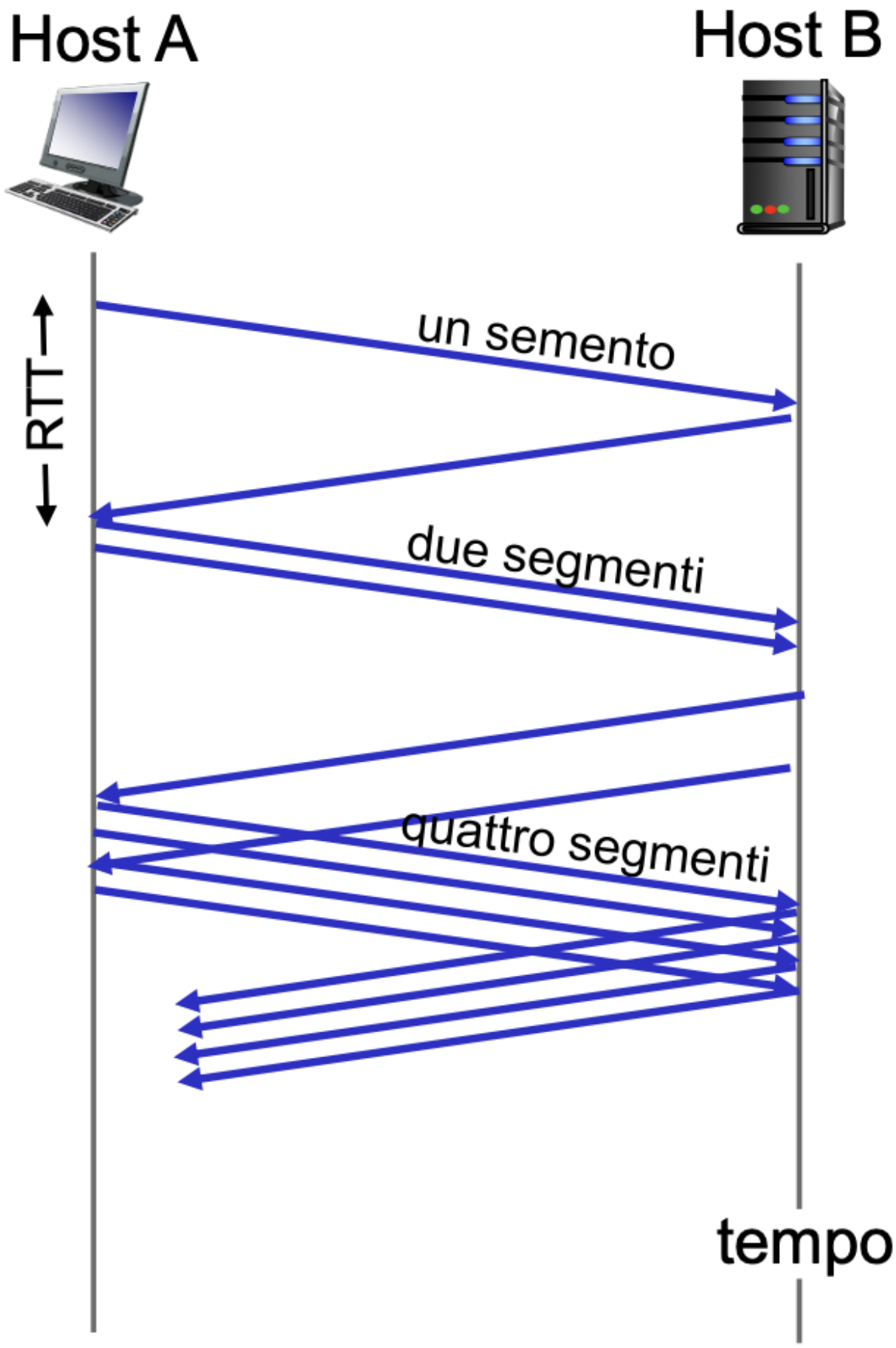
Relazione col controllo di flusso

- Il controllo del flusso regola la quantità e la velocità dei dati inviati in funzione della finestra di ricezione comunicata dal destinatario **rwnd** (byte liberi nel buffer di ricezione del destinatario)
- Quindi, $LastByteSent - LastByteAcked \leq cwnd$
- Combinando questo vincolo con quello visto in precedenza $LastByteSent - LastByteAcked \leq min(rwnd, cwnd)$

Assumendo che il buffer di ricezione sia sufficiente grande, possiamo trascurare il vincolo della finestra di ricezione (che assumiamo sempre maggiore della finestra di congestione)

Slow start (partenza lenta)

- Quando inizia la connessione, la frequenza aumenta in modo esponenziale fino a quando non si verifica un evento di perdita:
  - inizialmente **cwnd=** 1 MSS
  - raddoppia **cwnd** ogni RTT
  - fatto incrementando cwnd per ogni ACK ricevuto
- sintesi : il tasso iniziale è lento, ma aumenta in modo esponenziale e veloce.

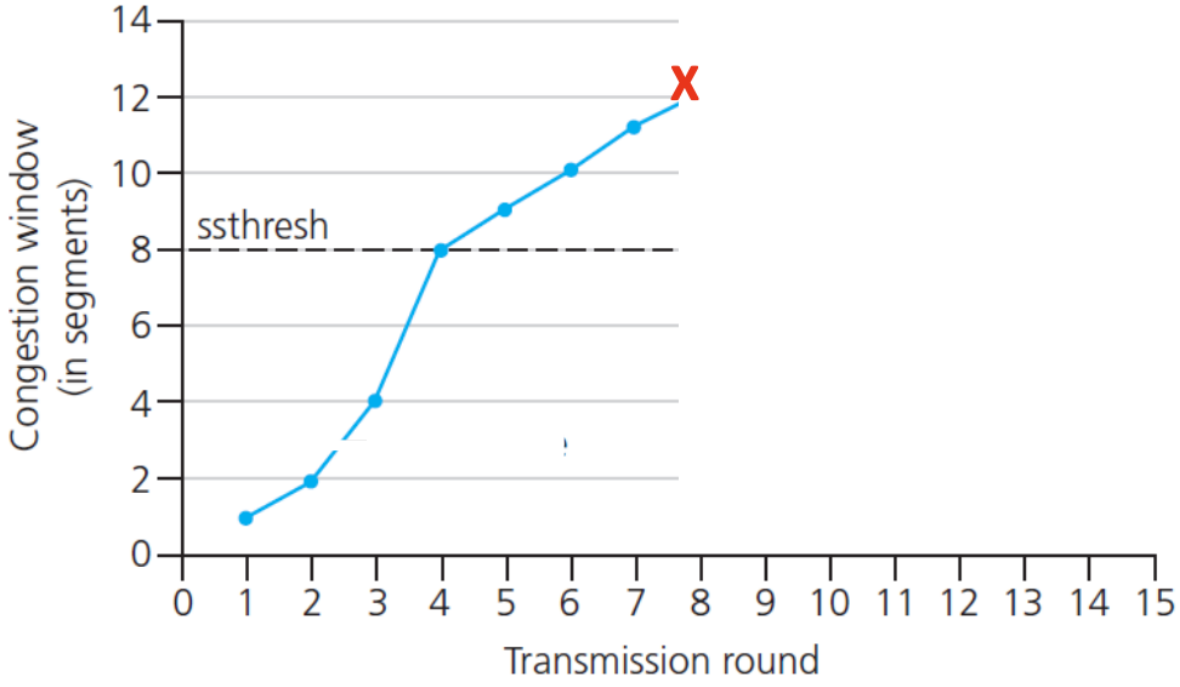


TCP : da slow start a congestion avoidance

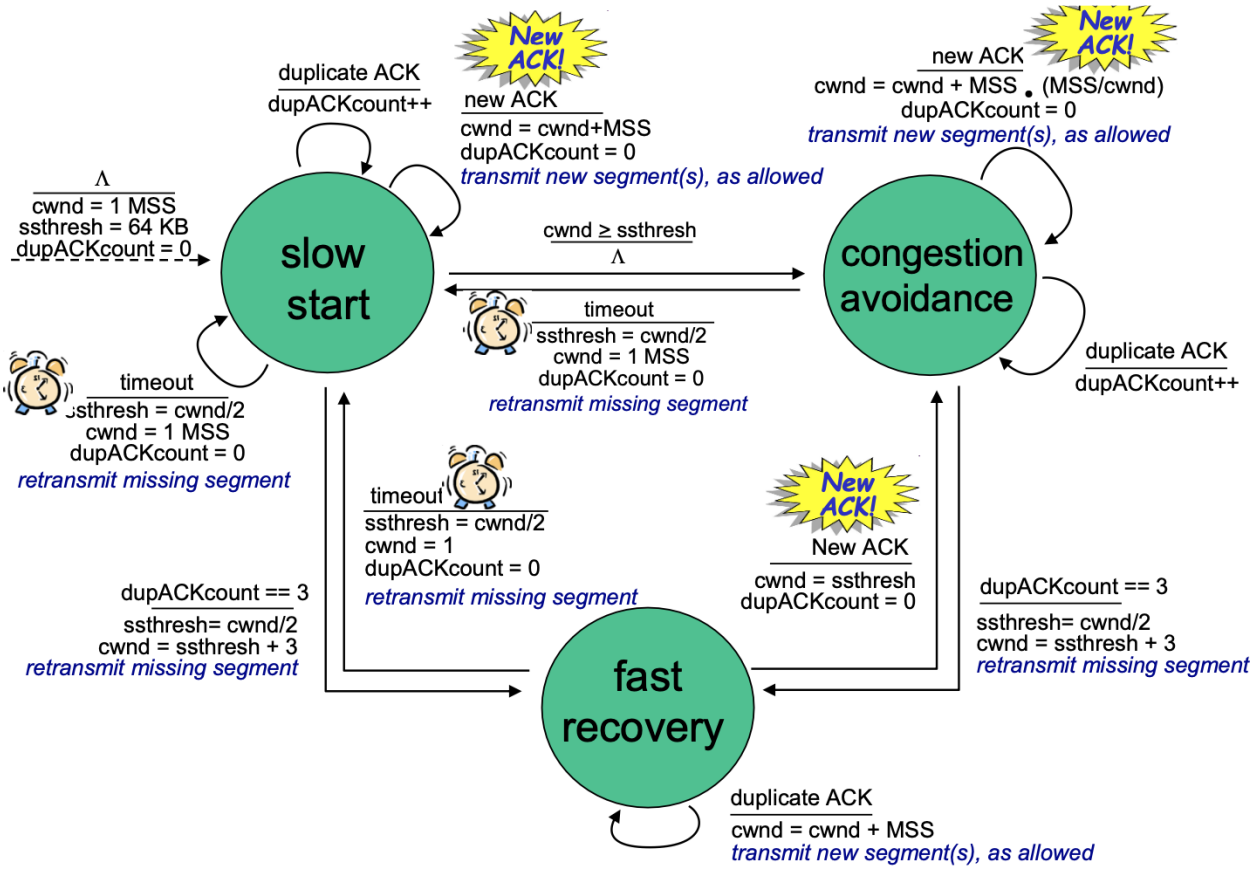
D : quando l'aumento esponenziale dovrebbe passare a quello lineare?
R : quando **cwnd** raggiunge 1/2 del suo valore prima del timeout.

Implementazione :

- **ssthresh** variabile (all'inizio 64 KB)
- in caso di evento di perdita, **ssthresh** è impostato a 1/2 **cwnd** giusto prima dell'evento di perdita

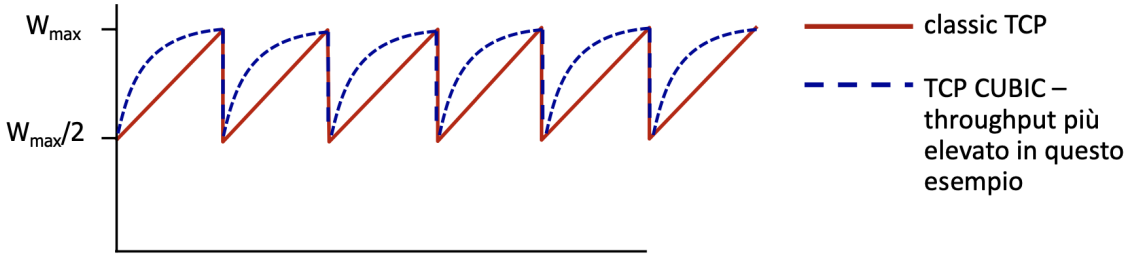


Riassunto: controllo della congestione

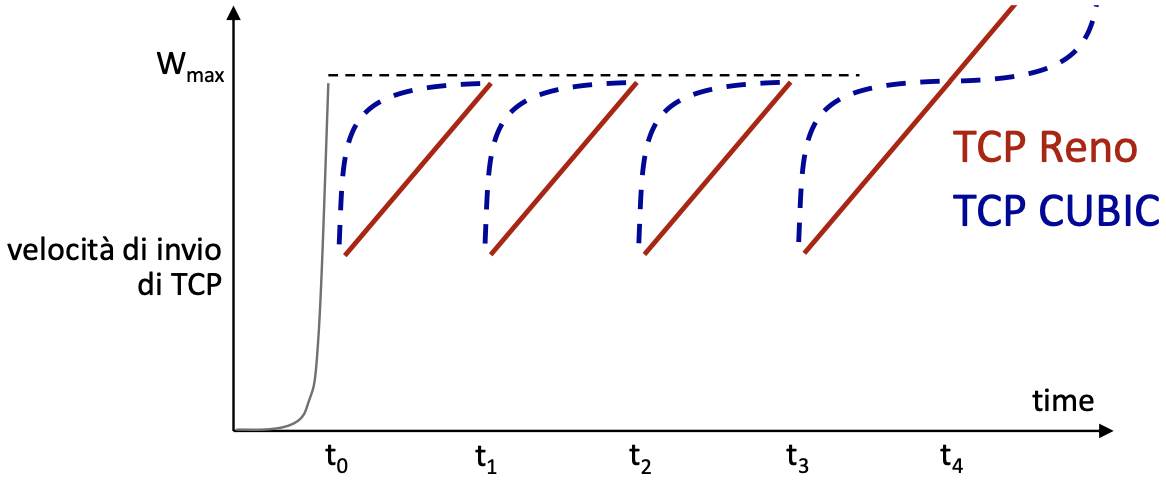


TCP CUBIC

- Esiste un modo migliore di AIMD per "sondare" la larghezza di banda utilizzabile?
- Intuizione:
  - W_{max}: la dimensione della finestra del controllo di congestione all'istante in cui viene rilevata la perdita
  - lo stato di congestione del collegamento bottleneck probabilmente (?) non è cambiato molto
  - dopo aver dimezzato la velocità/finestra in caso di perdita, inizialmente si sale verso W_{max} più velocemente, ma poi ci si avvicina a W_{max} più lentamente

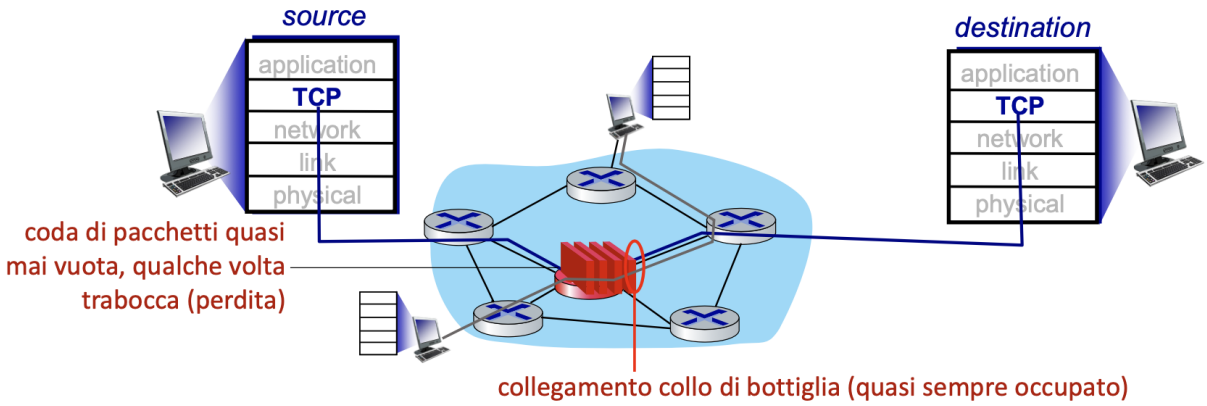


- K: l'istante nel futuro in cui la finestra TCP raggiungerà nuovamente W_{max}
  - K è determinato da diversi parametri
- aumenta W come una funzione del *cubo* della distanza tra l'istante corrente e K
  - aumenti maggiori quando ci si allontana maggiormente da K
  - aumenti minori (caduti) quando ci si avvicina a K
- TCP CUBIC predefinito in Linux, il TCP più diffuso per i server Web più comuni

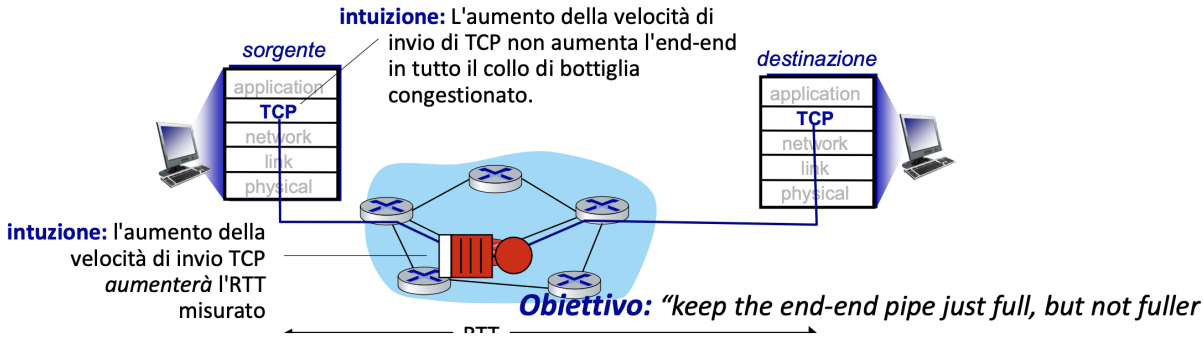


TCP e il collegamento "collo di bottiglia" congestionato

- TCP (classic, CUBIC) aumenta la velocità di invio di TCP finché non si verifica una perdita di pacchetti all'uscita di un router: il collegamento "collo bottiglia" (bottleneck)

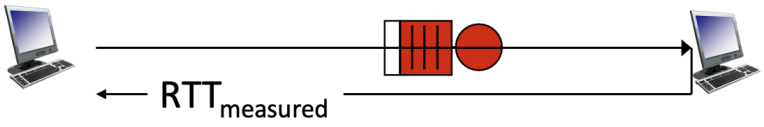


- comprendere la congestione: utile concentrarsi sul collegamento con collo di bottiglia congestionato



Controllo di congestione basato sul ritardo

Mantenere la tubazione da mittente a destinatario "sufficientemente piena, ma non di più": mantenere il collegamento a collo di bottiglia impegnato nella trasmissione, ma evitare ritardi elevati/buffering.



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{\text{RTT}_{\text{measured}}}$$

Approccio basato sul ritardo (TCP Vegas) :

- $\mathrm{R}TT_{min} - RTT$ minimo osservato (percorso non congestionato)
- throughput non congestionato con finestra di congestione *cwnd* è cwnd/RTT_{min}

se il throughput misurato è "molto vicino" al throughput non congestionato aumentare linearmente il cwnd / *poiché il percorso non è congestionato /*

altrimenti se il throughput misurato è "molto inferiore" al throughput non congestionato diminuire linearmente cwnd / *poiché il percorso è congestionato /*

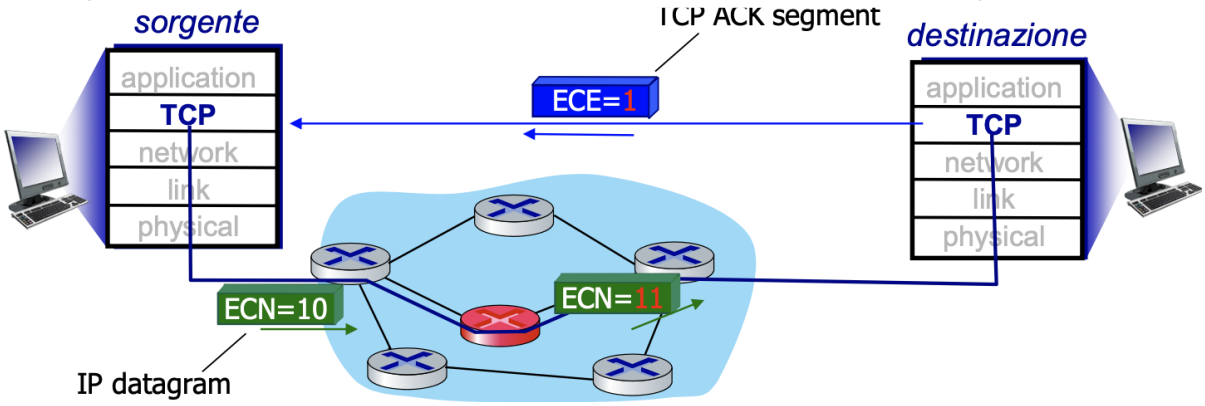
- controllo di congestione senza indurre/forzare perdite

- massimizzare il throughput (“keeping the just pipe full… ”) ma mantenendo il ritardo basso (“…but not fuller”)
- un certo numero di TCP distribuiti adottano un approccio basato sui ritardi
- BBR (Bottleneck Bandwidth and Round-trip propagation time) impiegato sulla rete dorsale (interna) di Google

Explicit congestion notification (ECN)

Le implementazioni di TCP spesso implementano un controllo della congestione assistito dalla rete:

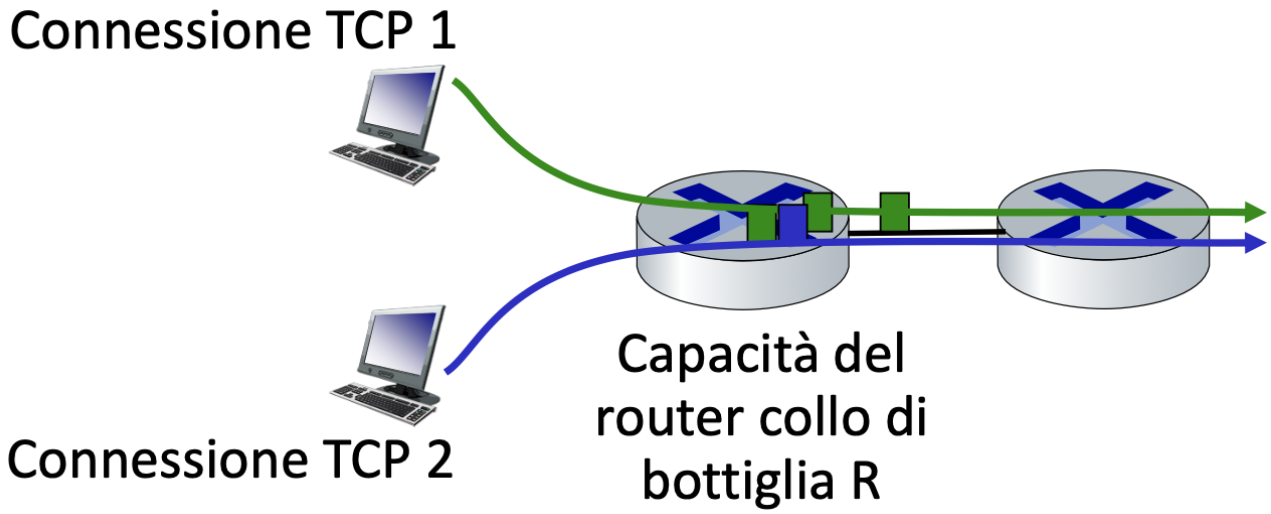
- due bit (ECN) nell'intestazione IP (all'interno del campo ToS) impostati da un router di rete per indicare la congestione
  - policy per determinare la marcatura scelta dall'operatore di rete
- indicazione di congestione portata a destinazione
- la destinazione imposta il bit ECE sul segmento ACK per notificare al mittente la presenza di una congestione
- coinvolge sia l'IP (marcatura dei bit ToS dell'intestazione IP) che il TCP (marcatura dei bit CWR,ECE dell'intestazione TCP).



- ECN viene negoziato in fase di instaurazione di una connessione TCP mediante opportune opzioni
- Il mittente imposta i bit ECN nell'intestazione IP per indicare che contengono i segmenti di una connessione in grado di gestire ECN

TCP fairness

Obiettivo di equità : se K sessioni TCP condividono lo stesso collegamento a collo di bottiglia con larghezza di banda R, ciascuna dovrebbe avere una velocità media di R/K



Fairness: tutte le app di rete devono essere "fair"?

Fairness e UDP

- Le app multimediali spesso non usano TCP
  - Non vogliono che la velocità sia ridotta dal controllo della congestione
- Usano invece UDP:
  - Inviano audio/video a velocità costante, tollerano la perdita di pacchetti
- Non esiste una "polizia di Internet" che controlli l'uso del controllo della congestione

Fairness, connessioni TCP parallele

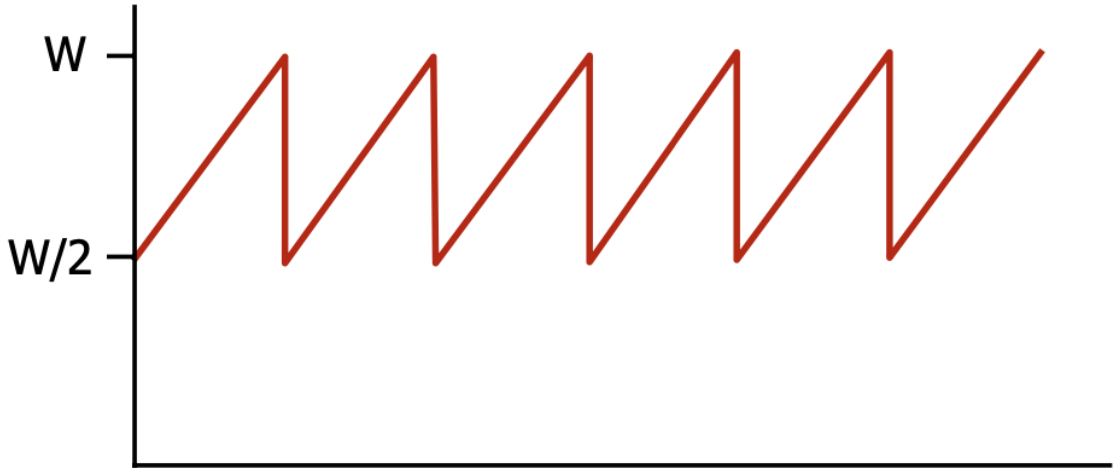
- L'applicazione può aprire più connessioni parallele tra due host

- I browser web lo fanno, ad esempio il link di velocità R con 9 connessioni esistenti:
  - nuova applicazione usa 1 connessione TCP, ottiene tasso R/10
  - nuova applicazione usa 11 connessioni TCP, ottiene tasso maggiore di R/2

Descrizione macroscopica del throughput di TCP

- Valore medio del throughput come funzione della dimensione della finestra e di RTT?
  - ignoriamo slow start, assumiamo che ci siano sempre dati da inviare
- W: dimensione della finestra (misurata in byte) quando si verifica una perdita
  - dimensione media della finestra (numero di byte "in volo") è ¾ W
  - throughput medio è 3/4W ogni RTT

$$\text{throughput TCP medio} = \frac{3}{4}\frac{W}{RTT} \text{ byte/s}$$



Evoluzione della funzionalità del livello di trasporto

- TCP, UDP: i principali protocolli di trasporto per 40 anni

- Sono stati sviluppate diversi "varianti" (*flavor*)di TCP, per scenari specifici:

Scenario	Sfide
Long, fat pipes (trasferimenti di dati di grandi dimensioni)	Molti pacchetti "in volo"; la perdita interrompe la pipeline
Reti wireless	Perdita dovuta a collegamenti wireless rumorosi, mobilità; il TCP la tratta come perdita di congestione
Long-delay links	RTT estremamente elevato
Reti dei data center	Sensibili alla latenza
Background traffic flows	Flussi TCP a bassa priorità, "in background"

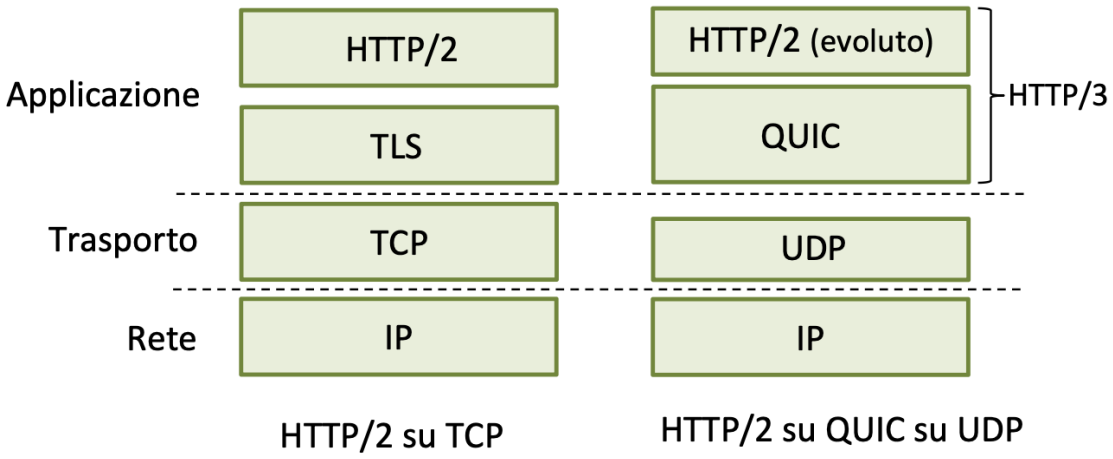
- Spostamento delle funzioni del livello di trasporto al livello di applicazione, in cima a UDP
  - HTTP/3 : QUIC

TCP su "long, fat pipes"

- esempio: segmenti di 1500 byte, RTT di 100ms, vogliamo un throughput di 10 Gbps
- richiede W = 83˙333 segmenti in volo
- throughput in termini della probabilità di perdita dei pacchetti, L
  $\mathrm{T}CPthroughput = \frac{1.22*MSS}{RTT\sqrt{L}}$
  $\rightarrow$ per raggiungere un throughput di 10 Gbps, occorre un tasso di perdita $\mathrm{L} = 2 * 10^{-10}$ (tasso di perdita davvero piccolo)

QUIC: Quick UDP Internet Connections

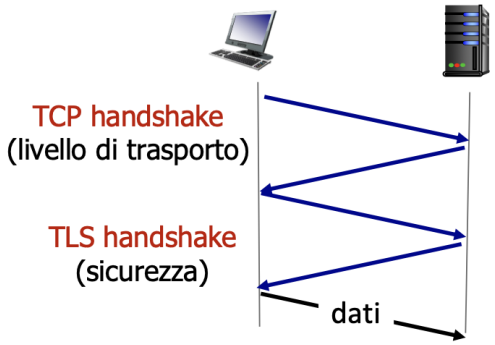
- protocollo di livello applicazione, sopra a UDP
  - aumenta le prestazioni di HTTP
  - impiegati in molti server e app di Google (Chrome, app mobile di YouTube)

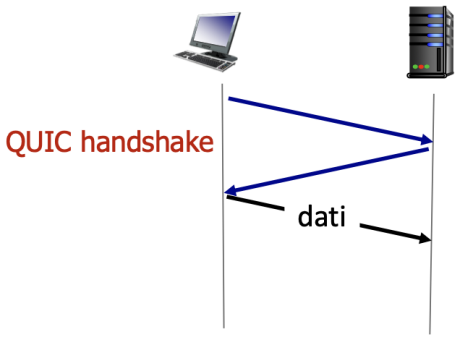


Adotta gli approcci che abbiamo studiato in queste lezioni per l'instaurazione della connessione, il controllo degli errori e il controllo della congestione

- **Controllo degli errori e della congestione:** "Readers familiar with TCP's loss detection and congestion control will find algorithms here that parallel well.know TCP ones."
- **Instaurazione della connessione:** affidabilità, controllo della congestione, autenticazione, cifratura, stato stability in un solo RTT
- multiplexing di molteplici "flussi" (stream) a livello di applicazione su una singola connessione QUIC
    - stato del trasferimento dati affidabile e della sicurezza separati
    - stato del controllo della congestione condiviso

QUIC: Instaurazione della connessione





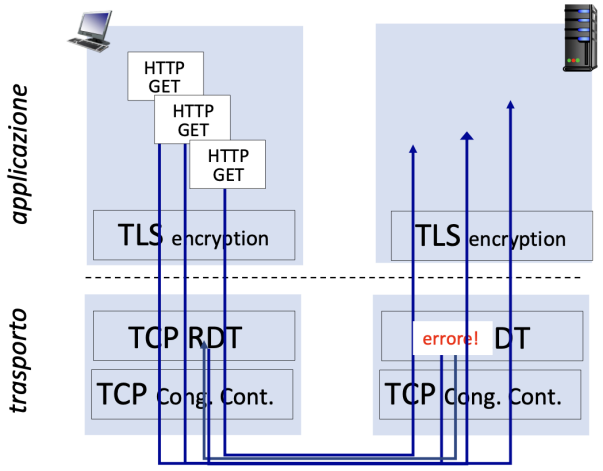
TCP (stato per il trasferimento affidabile e per il controllo della congestione) +
TLS (stato per l'autenticazione autenticazione e per la cifratura)

- 2 handshake in successione

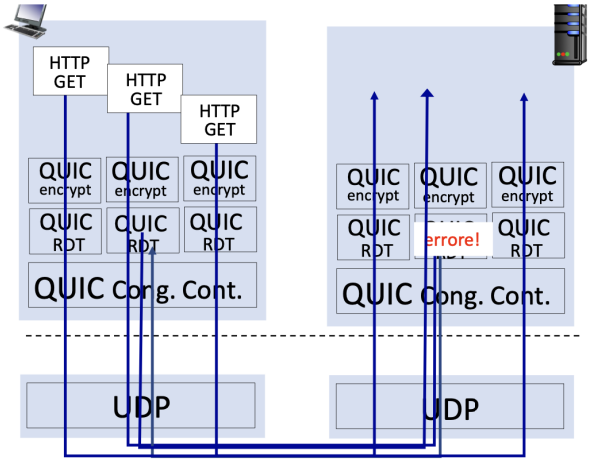
QUIC: stato della affidabilità, controllo della congestion, autenticazione e cifratura

- 1 handshake

QUIC: flussi: parallelismo, no blocco HOL



(a) HTTP 1.1



(b) HTTP/2 con QUIC: no HOL blocking