Reti

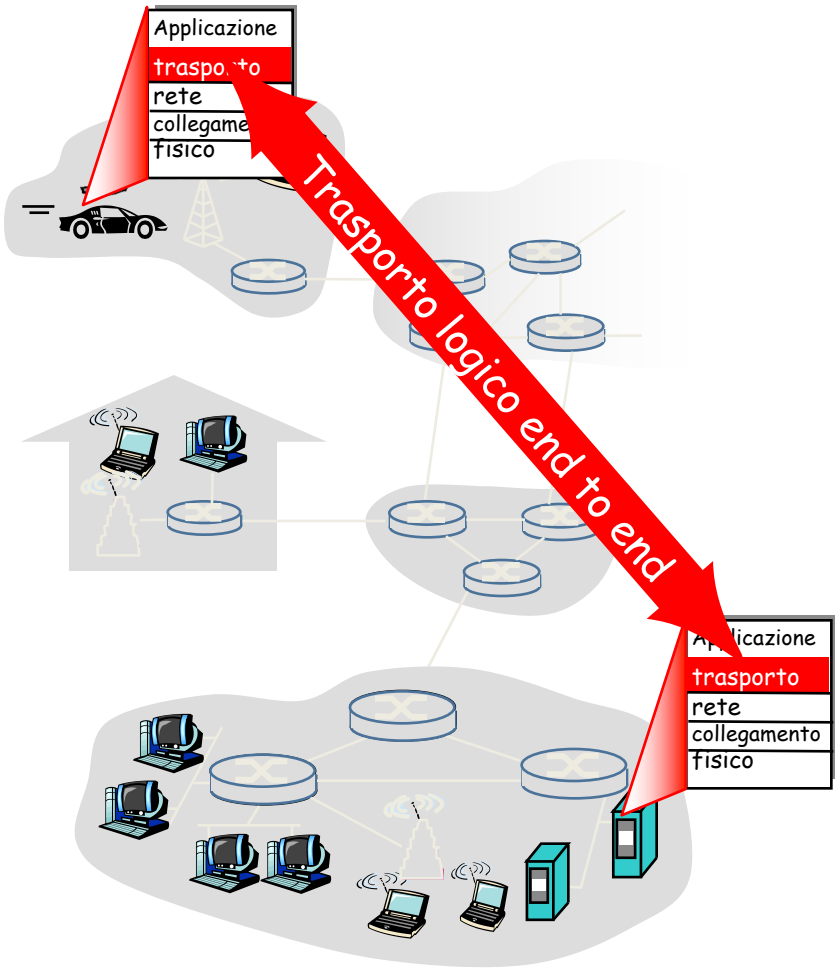
Il livello Trasporto: UDP e TCP

Renato Lo Cigno

http://disi.unitn.it/locigno/teaching-duties/reti

Copyright

- Forniscono la *comunicazione logica* tra processi applicativi di host differenti
- Eseguono nei sistemi terminali
  - lato invio: organizza i messaggi in segmenti e li passa al livello di rete
  - lato ricezione: riassembla i segmenti in messaggi e li passa al livello di applicazione

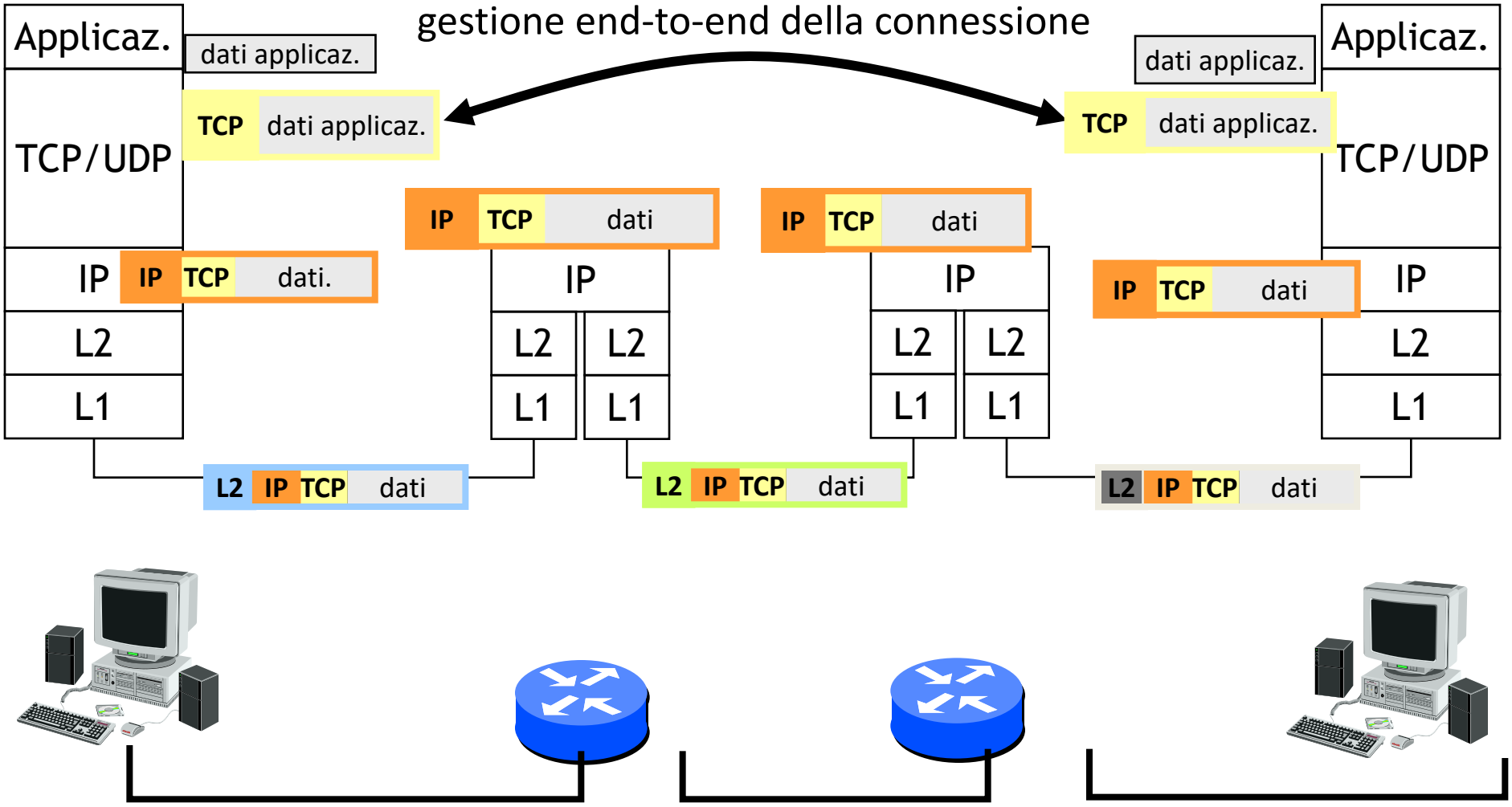


Livello trasporto in Internet

- Due protocolli di trasporto di base alternativi: TCP e UDP
- Modelli di servizio diversi
  - TCP orientato alla connessione, affidabile, controllo di flusso e congestione, mantiene lo stato della connessione
  - UDP non connesso, non-affidabile, senza stato
- Caratteristiche comuni:
  - multiplazione e demultiplazione mediante le porte
  - rilevazione (non correzione) errori su header
- Altri protocolli sono in fase di definizione/deployment:
  - SCTP (Stream Control Transmission Protocol)
  - DCCP (Datagram Congestion Control Protocol)
  - ...

Posizionamento e visione

- Il livello trasporto, come quello di applicazione, non ha visibilità dei dettagli della rete
- La semantica del protocollo è end-to-end e la rete viene trattata come una scatola nera
- Lo scopo “generale” del protocollo è fornire una astrazione di comunicazione ai protocolli di livello applicativo
  - attraverso servizi astratti dedicati (SAP – Service Access Point)
  - chiamati socket in gergo Internet (trattati a reti avanzate)


gestione end-to-end della connessione
Applicaz.
dati applicaz.
TCP/UDP
TCP
dati applicaz.
IP
IP
TCP
dati.
L2
L1
L2
IP
TCP
dati
IP
L2
L2
L1
L1
L2
IP
TCP
dati
IP
L2
L2
L1
L1
IP
TCP
dati
IP
L2
L1
Applicaz.
dati applicaz.
TCP/UDP
TCP
dati applicaz.
L2
IP
TCP
dati
L2
IP
TCP
dati
L2
IP
TCP
dati

- Fornisce un canale di trasporto end-to-end “astratto” tra due utenti, indipendentemente dalla rete
- Per compiere questo obiettivo, come tutti i livelli, il livello di trasporto offre, attraverso delle primitive, dei servizi al livello superiore e svolge una serie di funzioni
- Servizi offerti al livello applicativo:
  - connection-oriented affidabile
    - per il trasferimento dei dati viene attivata una connessione
    - ogni pacchetto (→ segmento) inviato viene “riscontrato” in modo individuale

    TCP
  - connectionless non affidabile
    - non viene attivata nessuna connessione
    - invio delle trame senza attendere alcun feedback dalla destinazione sulla corretta ricezione
      - se una trama viene persa non ci sono tentativi per recuperarla

    UDP

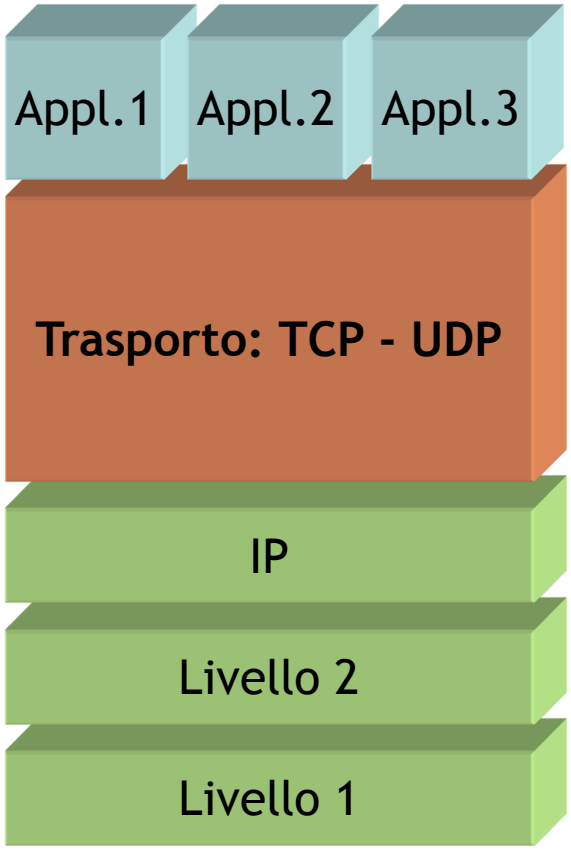
Funzioni svolte dal livello di trasporto

- Indirizzamento a livello applicativo / multiplazione / demultiplazione
  - multiplazione e indirizzamento a livello applicativo
  - scarto PDU malformate o con errori nell'header
- Instaurazione, gestione e rilascio delle connessioni
  - si preoccupa di gestire la connessione, ovvero lo scambio di informazioni necessarie per concordare l'attivazione di un canale di comunicazione
- Recupero degli errori (sui dati)
  - politiche implementate e azioni da svolgere in caso di errore o perdita di segmenti
- Consegna ordinata dei segmenti
- Controllo di flusso
  - azione preventiva finalizzata a limitare l'immissione di dati in rete a seconda della capacità end-to-end di questa
- Controllo della congestione
  - azioni da intraprendere come reazione alla congestione di rete

UDP

TCP

- Gli indirizzi di livello IP vengono utilizzati per l'instradamento dei pacchetti all'interno della rete e identificano univocamente gli host sorgente e destinazione
- Quando il pacchetto giunge alla destinazione e viene passato al livello di trasporto nasce un ulteriore problema:
  - poiché sul livello di trasporto si appoggiano più applicazioni, com'è possibile distinguere l'una dall'altra?
- Si introduce il concetto di porta, che non è altro che un codice che identifica l'applicazione



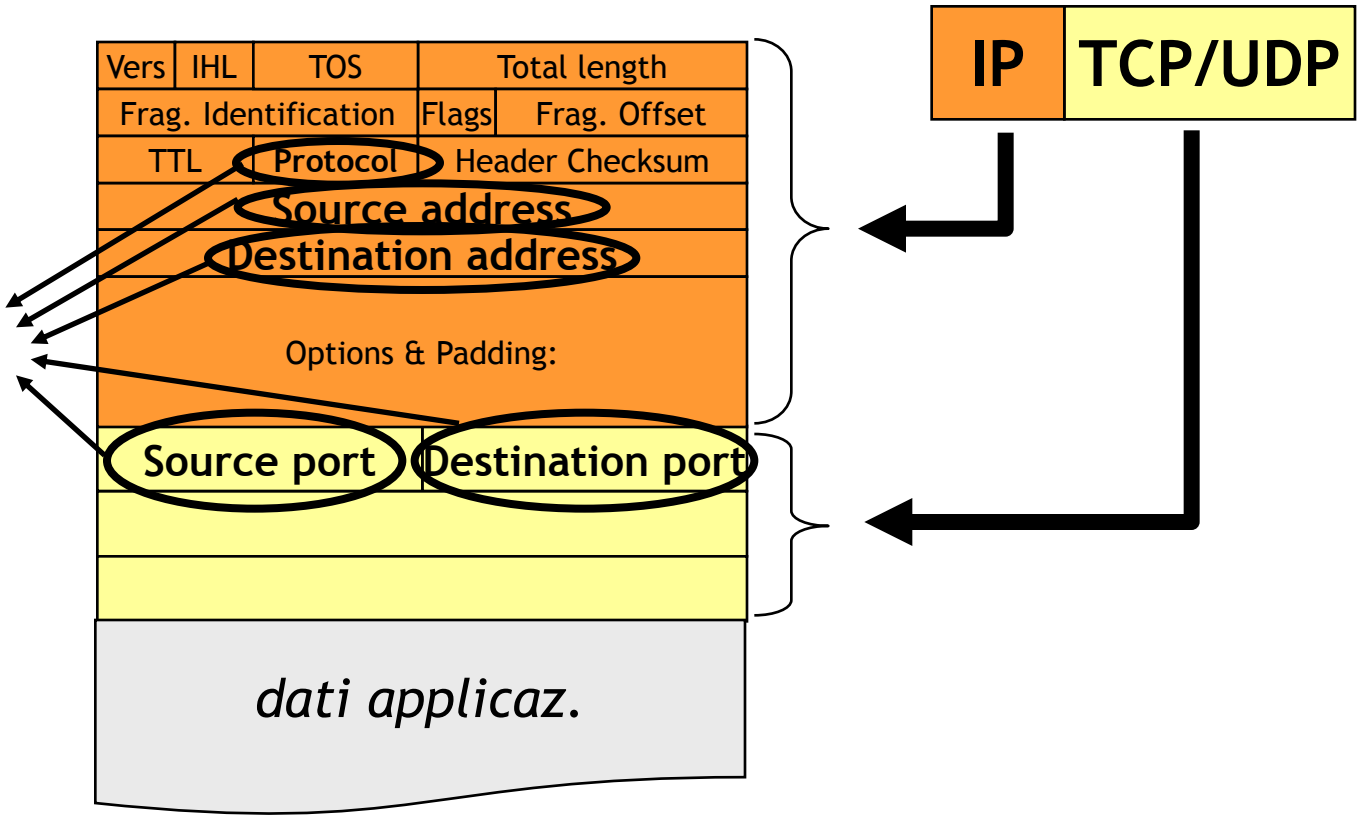
Indirizzamento: mux/demux

- Il destinatario finale dei dati non è un host ma un processo in esecuzione sull'host
- L'interfaccia tra processi applicativi e strato trasporto è rappresentata da una "porta"
  - numero intero su 16 bit
  - associazione tra porte e processi
  - processi server standard e "pubblici" sono associati a porta "ben nota", inferiore a 1024 (es: 80 per WWW, 25 per email)
  - processi client e server non standard o "nascosti" usano una porta superiore a 1024, nel caso dei client assegnata dinamicamente dal sistema operativo

Indirizzamento: mux/demux

- Il numero di porta può essere
  - statico (well known port)
    - identificativi associati ad applicazioni standard (posta elettronica, web, ftp, dns, ...)
  - dinamico (emphemeral)
    - identificativi assegnati direttamente dal sistema operativo al momento dell'apertura della connessione
- La porta sorgente e la porta destinazione non sono uguali
- L'utilizzo delle porte, insieme agli indirizzi IP sorgente e destinazione servono per il mux/demx dei dati
- Flusso o connessione
  - dati di una singola comunicazione logica
  - una applicazione può usare più flussi

Indentificare un flusso

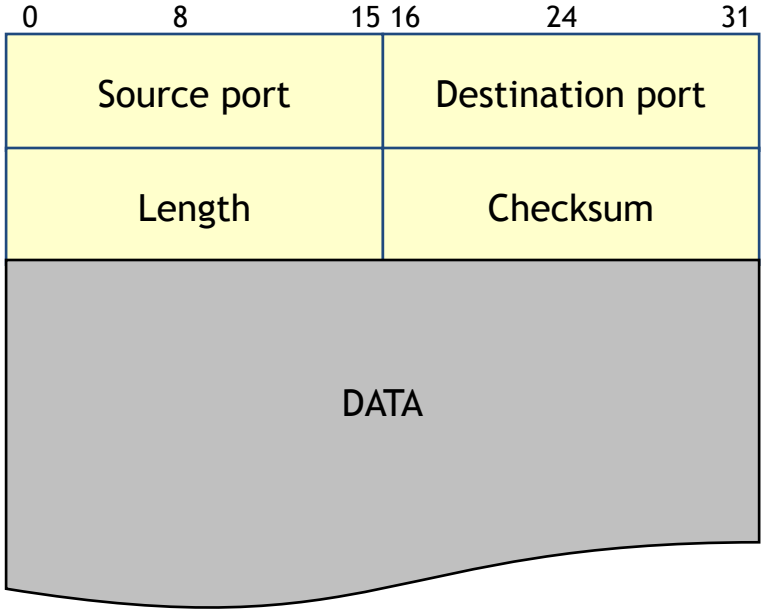


5-upla che identifica la connessione (e il conseguente flusso) tra due applicazioni

La 5-upla **{S-IP,D-IP,PROT,S-Port,D-Port}**
identifica in modo univoco un flusso in qualsiasi punto della rete ...
e quindi anche all'interfaccia tra protocollo applicativo e protocollo di trasporto in un host

Formato dei segmenti UDP

- Source Port e Destination Port [16 bit]: identificano i processi sorgente e destinazione dei dati
- Length [16 bit]: lunghezza totale (espressa in byte) del datagramma, compreso l'header UDP
- Checksum [16 bit]: campo di controllo che serve per sapere se il datagramma corrente contiene errori



User Datagram Protocol

- Protocollo di trasporto connectionless non affidabile
- Svolge solo funzione di indirizzamento delle applicazioni (porte)
- NON gestisce:
  - connessioni; controllo di flusso
  - recupero degli errori (solo rilevamento)
  - controllo della congestione; riordino dei pacchetti
- Supporto di transazioni semplici tra applicativi e applicazioni real-time che tollerano perdite
- Un'applicazione che usa UDP deve risolvere problemi di affidabilità, perdita di pacchetti, duplicazione, controllo di sequenza, controllo di flusso, controllo di congestione
- Standardizzato in RFC 768

Tecniche ARQ e protocolli a finestra

- Per ottenere affidabilità si usano tecniche di ritrasmissione automatica delle informazioni corrotte o perse
  - ARQ – Automatic Retransmission reQuest
- La decisione (automatica) se ritrasmettere l'informazione viene presa in base a protocolli che sfruttano finestre di trasmissione e ricezione
  - stop and wait (finestra pari a una unità dati – PDU)
  - finestre di dimensione fissa
  - finestre con dimensione variabile
- Servono tecniche per
  - rilevare le PDU errate
  - rilevare le PDU mancanti

Obiettivo: rilevare gli "errori" (bit alterati) nel segmento trasmesso

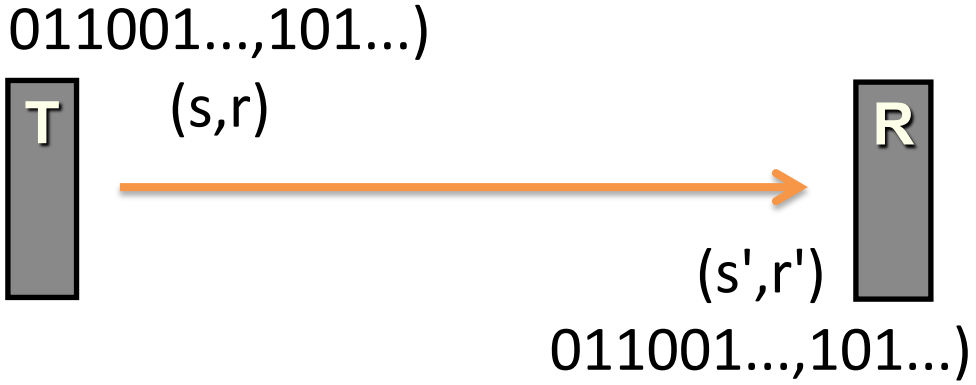
Mittente:

- Tratta il contenuto del segmento come una sequenza di interi da n bit
- checksum: somma (complemento a 1) i contenuti del segmento
- Il mittente pone il valore della checksum nel campo checksum del segmento

Ricevente:

- calcola la checksum del segmento ricevuto
- controlla se la checksum calcolata è uguale al valore del campo checksum:
  - No - errore rilevato
  - Sì - nessun errore rilevato
  - Particolari pattern di errori possono sfuggire

- In generale è una funzione della stringa s dei bit di informazione da trasmettere che calcola una stringa r di bit di ridondanza che verranno trasmessi insieme ai bit di informazione r = f(s)



- Il ricevitore calcola a sua volta r'' = f(s') sulla stringa di bit di informazione ricevuti s' e solamente se r'' = r' il pacchetto ricevuto viene accettato, altrimenti viene scartato (o corretto se possibile)

esempi banali, i protocolli moderni usano codici molto più sofisticati, in genere codici su campi finiti (Galois) oppure codici convoluzionali o a traliccio (trellis)

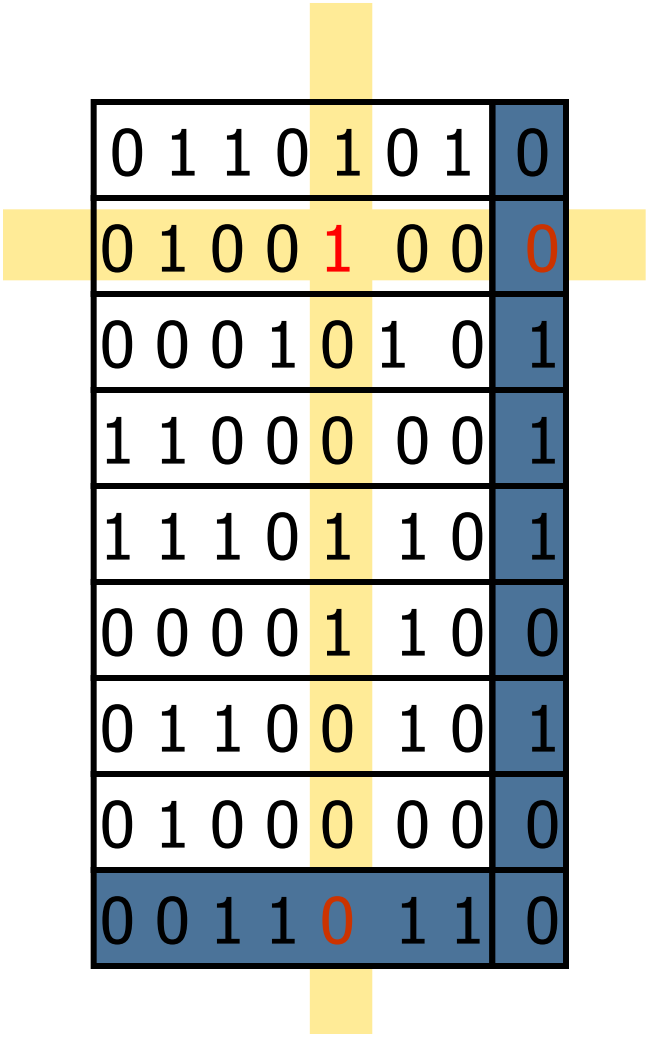
- bit di parità (riconosce errori in numero dispari)

0 1 1 0 1 0 1 0	0
0 1 0 0 1 0 1 0	1

- codice a ripetizione (decisione a maggioranza: permette di correggere errori)

0 1 1 0 1 0 1 0
0 1 1 0 1 0 1 0
0 1 1 0 1 0 1 0

- parità di riga e colonna (consente la correzione di errori singoli)



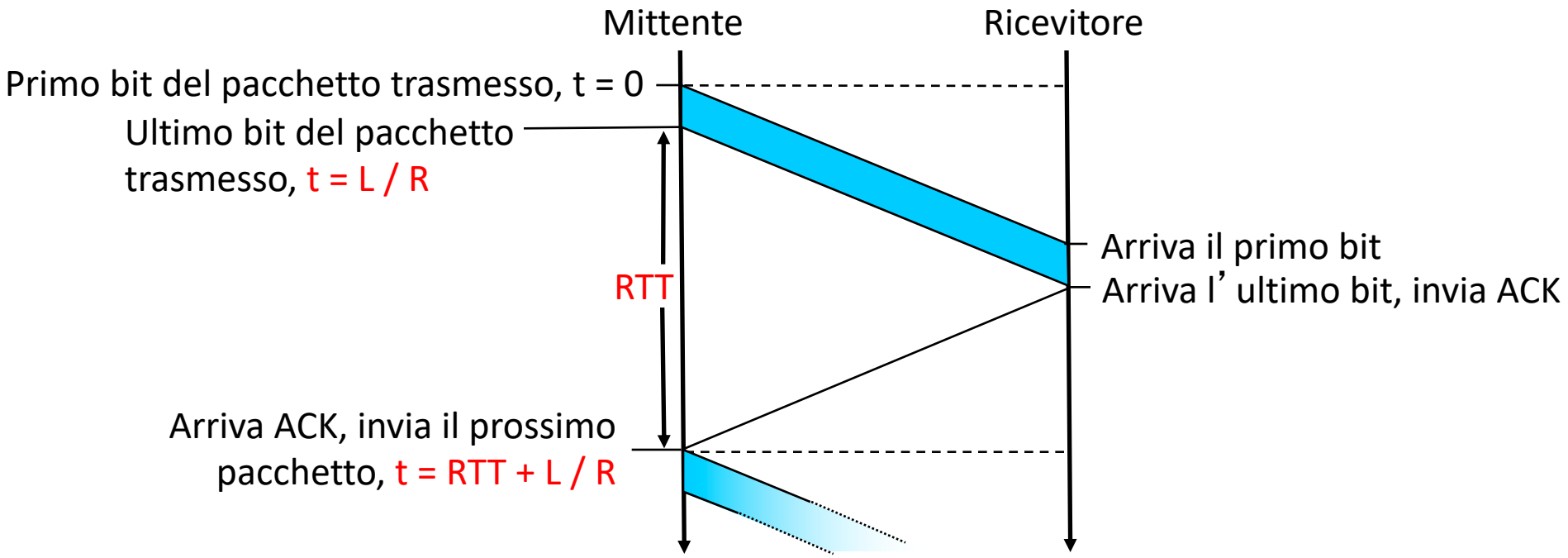
Stop and wait

- Il trasmettitore:
  - invia una PDU dopo averne fatta una copia
  - attiva un orologio (tempo di timeout)
  - si pone in attesa della conferma di ricezione (acknowledgment - ACK)
  - se scade il timeout prima dell'arrivo della conferma, ripete la trasmissione
  - quando riceve un ACK:
    - controlla la correttezza dell'ACK
    - controlla il numero di sequenza
    - se l'ACK è relativo all'ultima PDU trasmessa abilita la trasmissione della prossima PDU

Stop and wait

- Il ricevitore, quando riceve una PDU:
  - controlla la correttezza della PDU
  - controlla il numero di sequenza
  - se la PDU è corretta e in ordine invia la conferma di ricezione (ACK) e la consegna ai livelli superiori
  - se la PDU è errata o fuori sequenza scarta la PDU e non fa nulla (o invia NACK se possibile)

Stop and wait



L = lunghezza del pacchetto (bit)
R = capacità della rete (bit/s)

$$\text{efficienza} = \frac{L / R}{RTT + L / R}$$

Finestre multi-PDU

- Il protocollo Stop and Wait è poco efficiente perché dopo la trasmissione di ogni singolo pacchetto bisogna attendere la sua conferma

- Permettere la trasmissione di più di una PDU prima di fermarsi in attesa delle conferme migliora le prestazioni
  - Trasmissione e ricezione ACK avvengono in “pipeline”

- Devo poter riconoscere le PDU trasmesse

Finestra di trasmissione

- La finestra di trasmissione W_T è

"la quantità massima di PDU in sequenza che il trasmettitore è autorizzato ad inviare in rete senza averne ricevuto riscontro (ACK)"

- La dimensione della finestra è limitata dalla quantità di memoria allocata in trasmissione
- W_T rappresenta anche il massimo numero di PDU contemporaneamente presenti sul canale o in rete

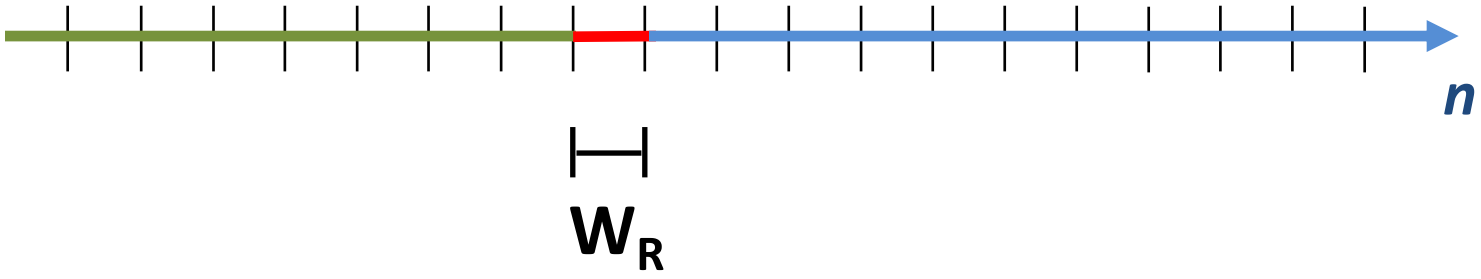
- La finestra di ricezione W_R è
  "la quantità di PDU che il ricevitore è disposto ad accettare e memorizzare"

- La dimensione della finestra è limitata dalla quantità di memoria allocata in ricezione

Finestra di trasmissione

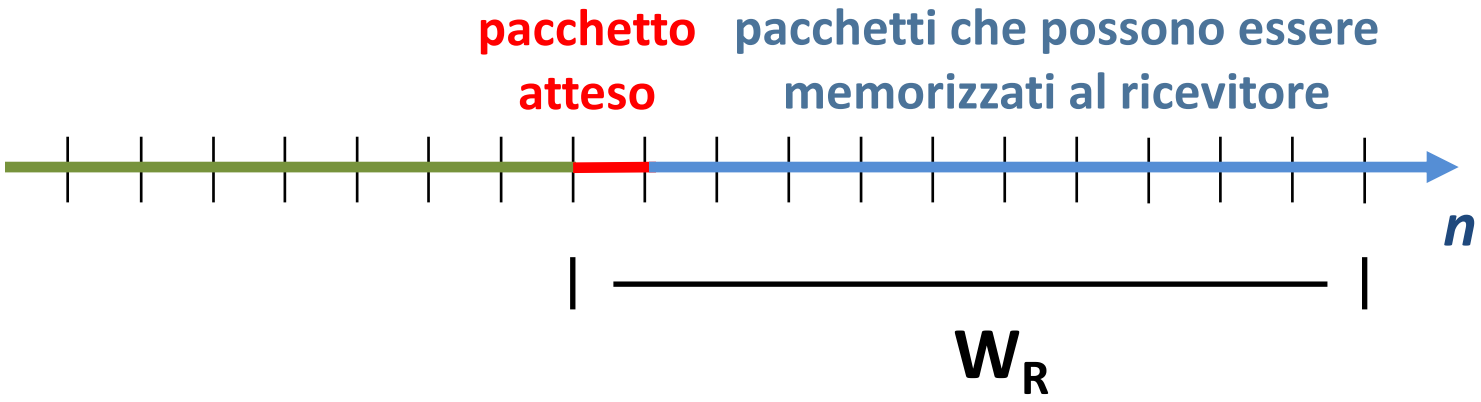
- l'asse x rappresenta il flusso di informazione da trasmettere, ad esempio un file
- La finestra di trasmissione definisce la porzione di dati che possono essere trasmessi
- La finestra si "sposta" lungo i dati (sliding window) man mano che vengono trasmessi e confermati (riscontro / ACK)

Finestra di ricezione unitaria



I pacchetti fuori sequenza non possono essere accettati
Se si perde un pacchetto si deve ritrasmettere l'intera sequenza di $N=W_T$ pacchetti inviati (Go Back N)

Finestra di ricezione maggiore di uno



pacchetti confermati

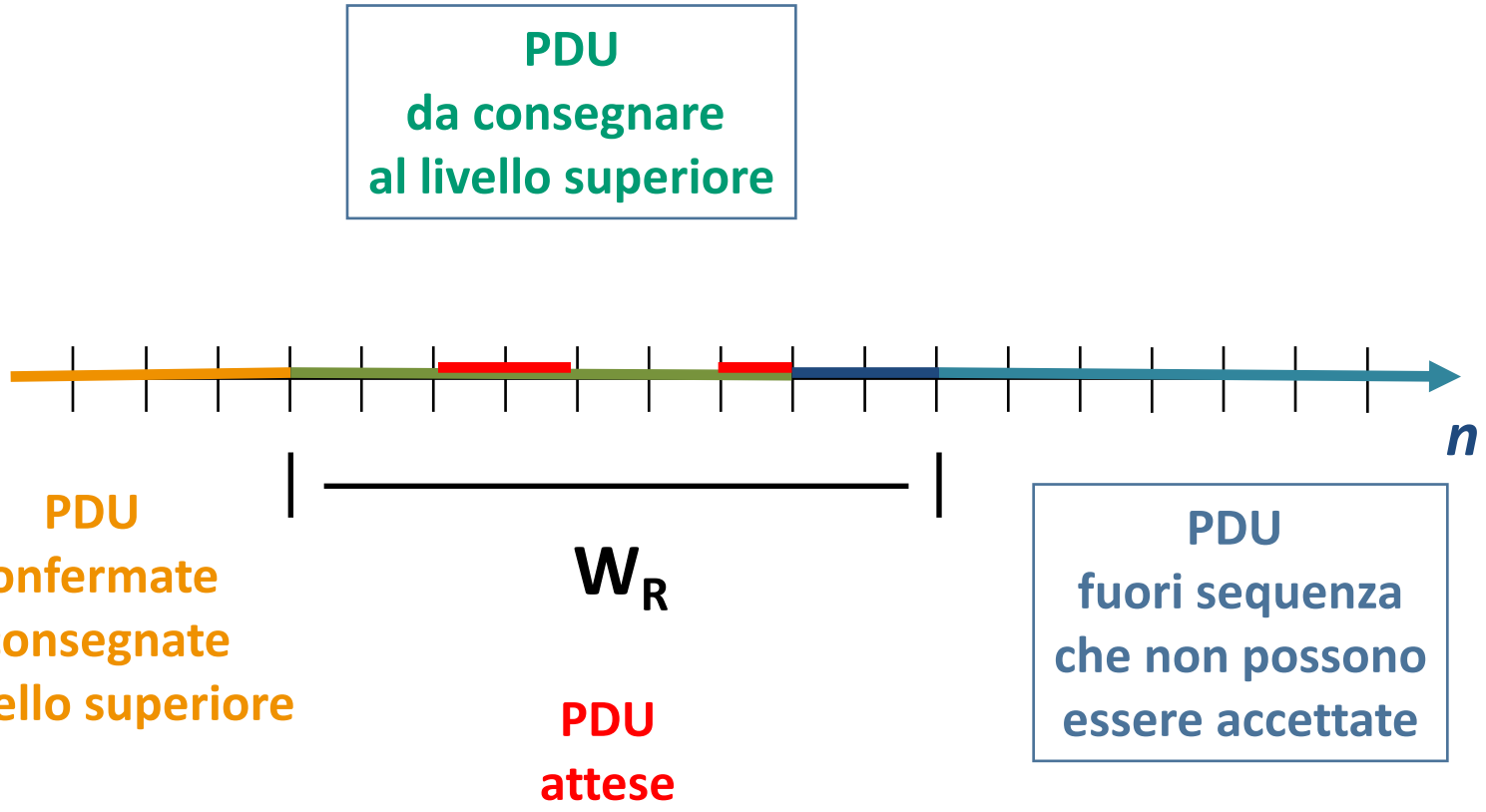
I pacchetti fuori sequenza possono essere accettati in attesa di ricevere quello atteso. Se si perde un pacchetto si può ritrasmettere solo quello perso purché $W_R >= W_T$ (Selective Repeat)

- La semantica associata al pacchetto di riscontro può essere:
  - **ACK individuale (o selettivo)**: si notifica la corretta ricezione di un pacchetto particolare.
    ACK(*n*) significa "ho ricevuto il pacchetto *n*"
  - **ACK cumulativo**: si notifica la corretta ricezione di tutti i pacchetti con numero di sequenza inferiore a quello specificato nell'ACK.
    ACK*(n*) significa *"*ho ricevuto tutto fino ad *n escluso*"
  - **ACK negativo (NAK)**: si notifica la richiesta di ritrasmissione di un singolo pacchetto.
    NAK(*n*) significa "ritrasmetti il pacchetto *n*"
- Trasmettitore e Ricevitore si devono accordare preventivamente sulla semantica degli ACK

Piggybacking

- Nel caso di flussi di informazione bidirezionali, è sovente possibile scrivere l'informazione di riscontro (ACK) nella intestazione di PDU di informazione che viaggiano nella direzione opposta
- Si risparmia la trasmissione di un pacchetto a livello rete

Finestra di ricezione > di 1

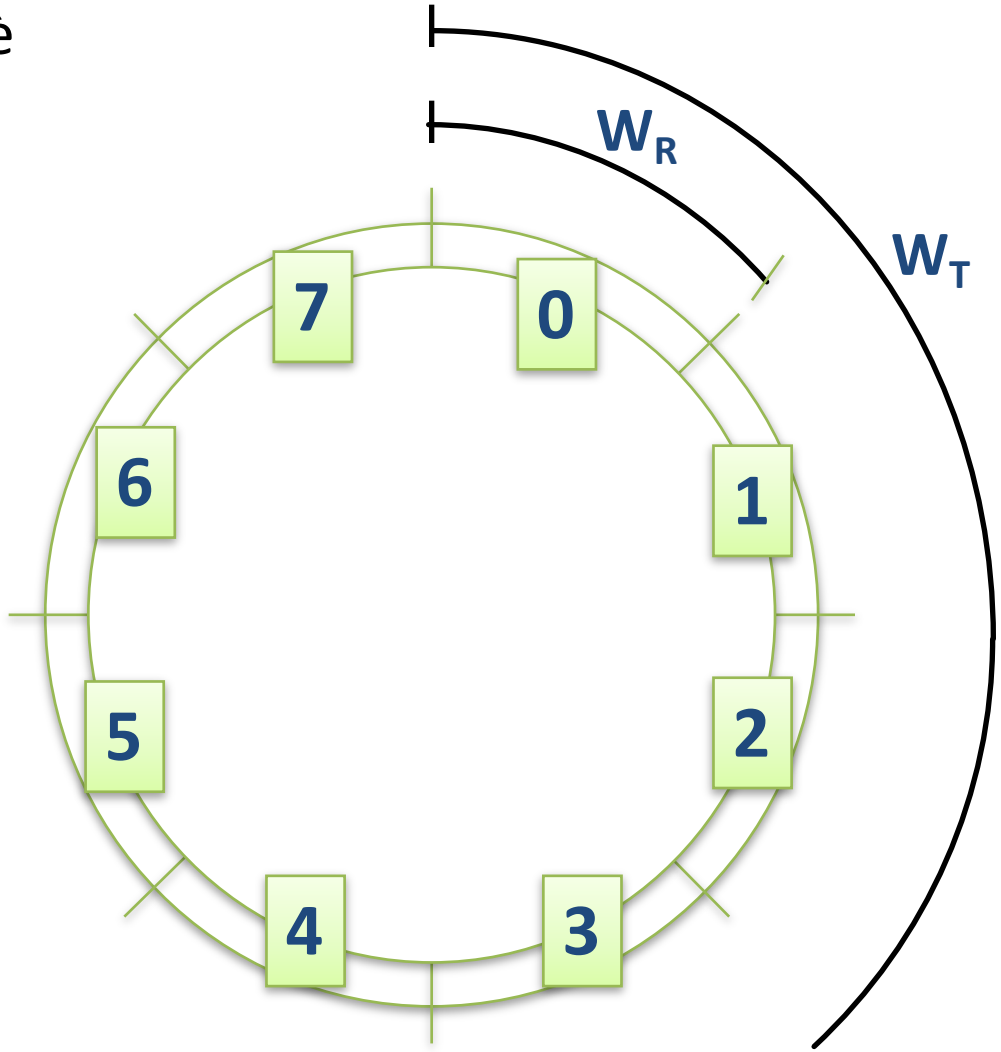


- La numerazione delle PDU è ciclica:
  - k bit di numerazione
  - numerazione modulo 2^k
  - $W_T + W_R < 2^k$

3 bit di numerazione

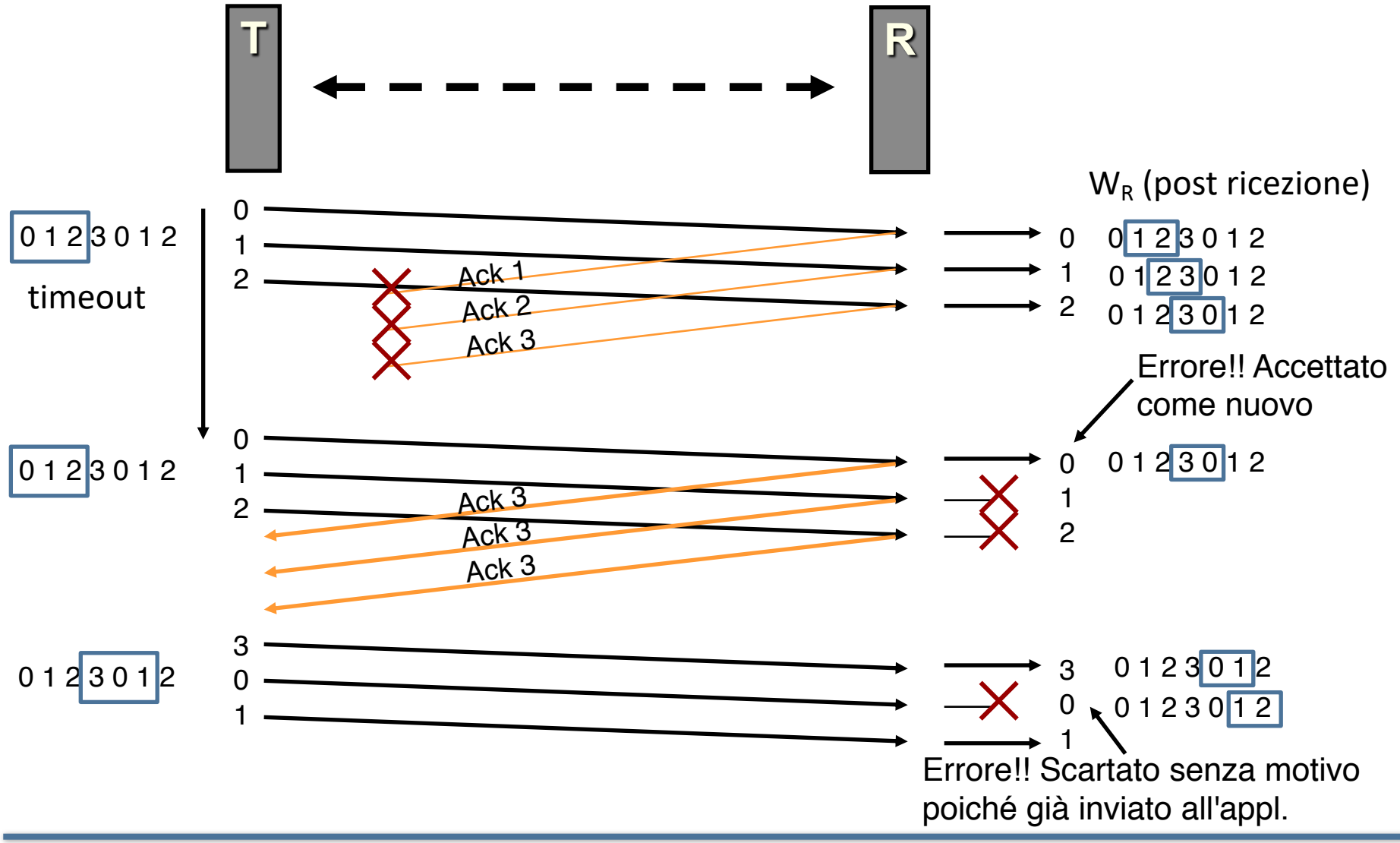
$W_R = 1$

$W_T = 3$



- Se $W_T+W_R > 2^k$ si possono creare ambiguità in caso di perdita di informazione (Dati o ACK) e il protocollo non funziona più
- Esempio con $W_T=3$, $W_R=2$, $k=2$

Esempio


T
R
W_R (post ricezione)
0 1 2 3 0 1 2
timeout
0
1
2
Ack 1
Ack 2
Ack 3
0 0 1 2 3 0 1 2
1 0 1 2 3 0 1 2
2 0 1 2 3 0 1 2
Errore!! Accettato come nuovo
0 1 2 3 0 1 2
0 1 2 3 0 1 2
Ack 3
Ack 3
Ack 3
0 1 2 3 0 1 2
3
0
1
3 0 1 2 3 0 1 2
0 0 1 2 3 0 1 2
1
Errore!! Scartato senza motivo poiché già inviato all'appl.

Selective Repeat

- Il trasmettitore:
  - trasmette fino ad N = W_T PDU senza ricevere ACK
  - per ciascuna inizializza un timeout
  - quando riceve un ACK relativo ad una PDU la toglie dalla finestra di trasmissione e trasmette una nuova PDU
  - se scade un timeout ritrasmette selettivamente la PDU relativa all'ACK (e re-inizializza il timeout)
  - quando ritrasmette una PDU può
    - fermarsi fino a quando ha ricevuto l'ACK di questa PDU
    - continuare a trasmettere PDU nuove se la dinamica della finestra e degli ACK relativi alle altre PDU lo consente

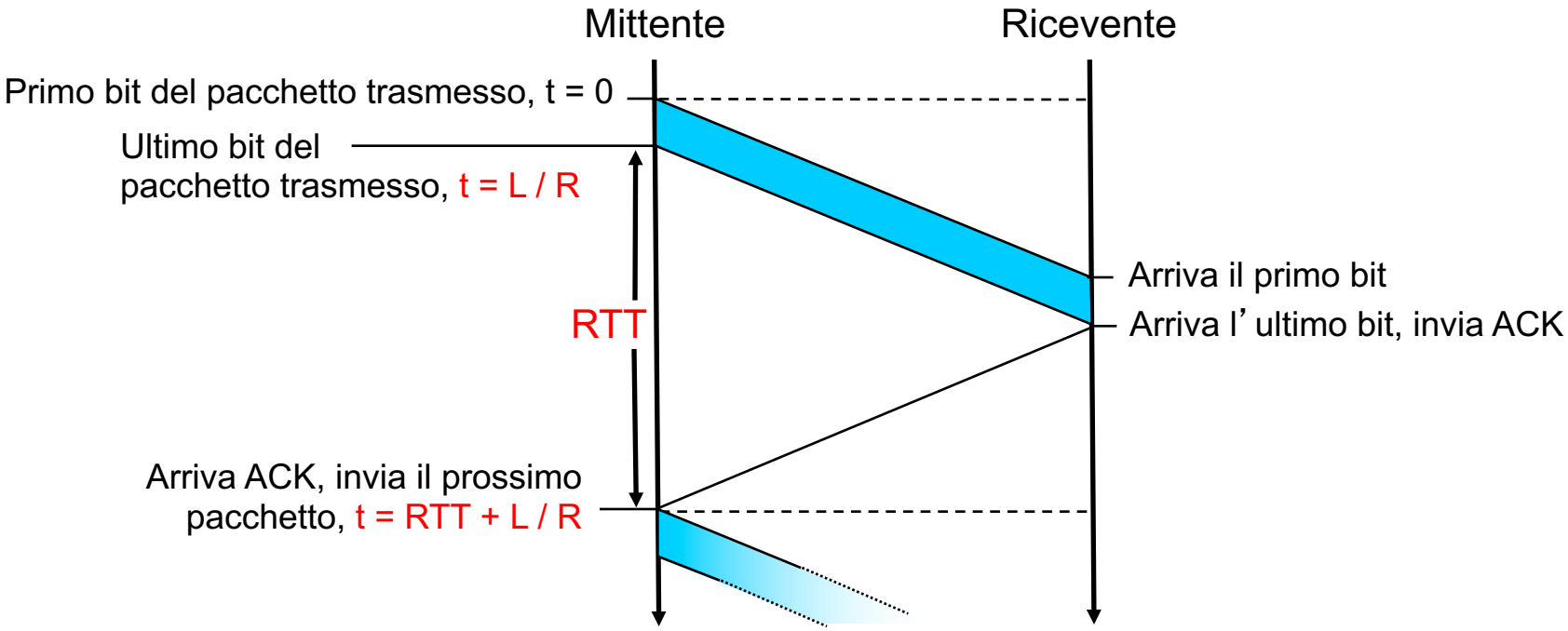
- Il ricevitore:
  - riceve una PDU
  - controlla la correttezza della PDU
  - controlla il numero di sequenza
  - se la PDU è corretta ed in sequenza la consegna al livello superiore
  - se la PDU è corretta ma non in sequenza:
    - se è entro la finestra di ricezione la memorizza
    - se è fuori dalla finestra di ricezione la scarta
  - invia un ACK
    - relativo alla PDU ricevuta (ACK selettivi)
    - relativo all'ultima PDU ricevuta in sequenza (ACK cumulativi)

esempio: collegamento da 1 Gbps, ritardo di propagazione 15 ms, pacchetti da 1 kbyte:

$$T_{trasm} = \frac{L \text{ (lunghezza del pacchetto in bit)}}{R \text{ (tasso trasmissivo, bps)}} = \frac{8 \text{ kb/pacc}}{10^9 \text{ b/sec}} = 8 \text{ microsec}$$

$$U_{mitt} = \frac{L / R}{RTT + L / R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

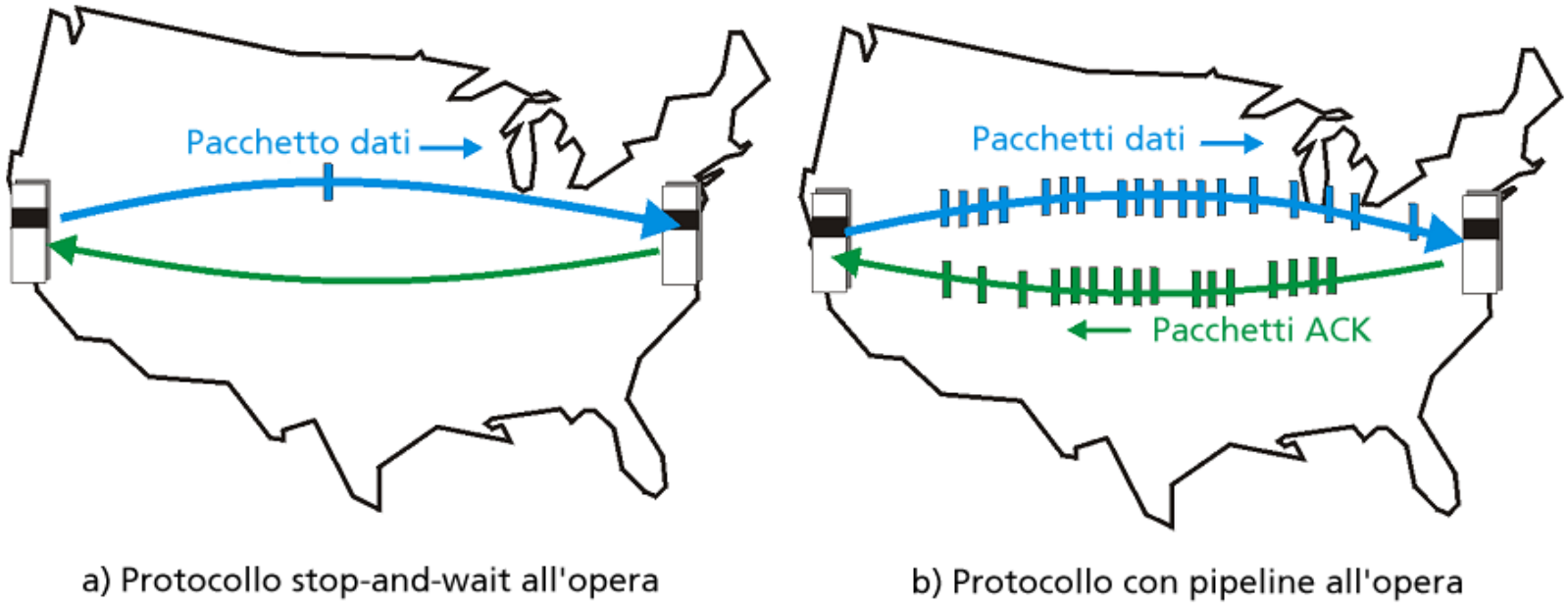
- U_{mitt} : utilizzo è la frazione di tempo in cui il mittente è occupato nell'invio di bit
- Un pacchetto da 1 KB ogni ~30 msec → throughput di ~33 kB/sec in un collegamento da 1 Gbps
- Il protocollo di trasporto limita l'uso delle risorse fisiche!



$$U_{mitt} = \frac{L/R}{RTT + L/R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

Pipelining: il mittente ammette più pacchetti in transito, ancora da notificare

- buffering dei pacchetti presso il mittente e ricevente per ritrasmissioni

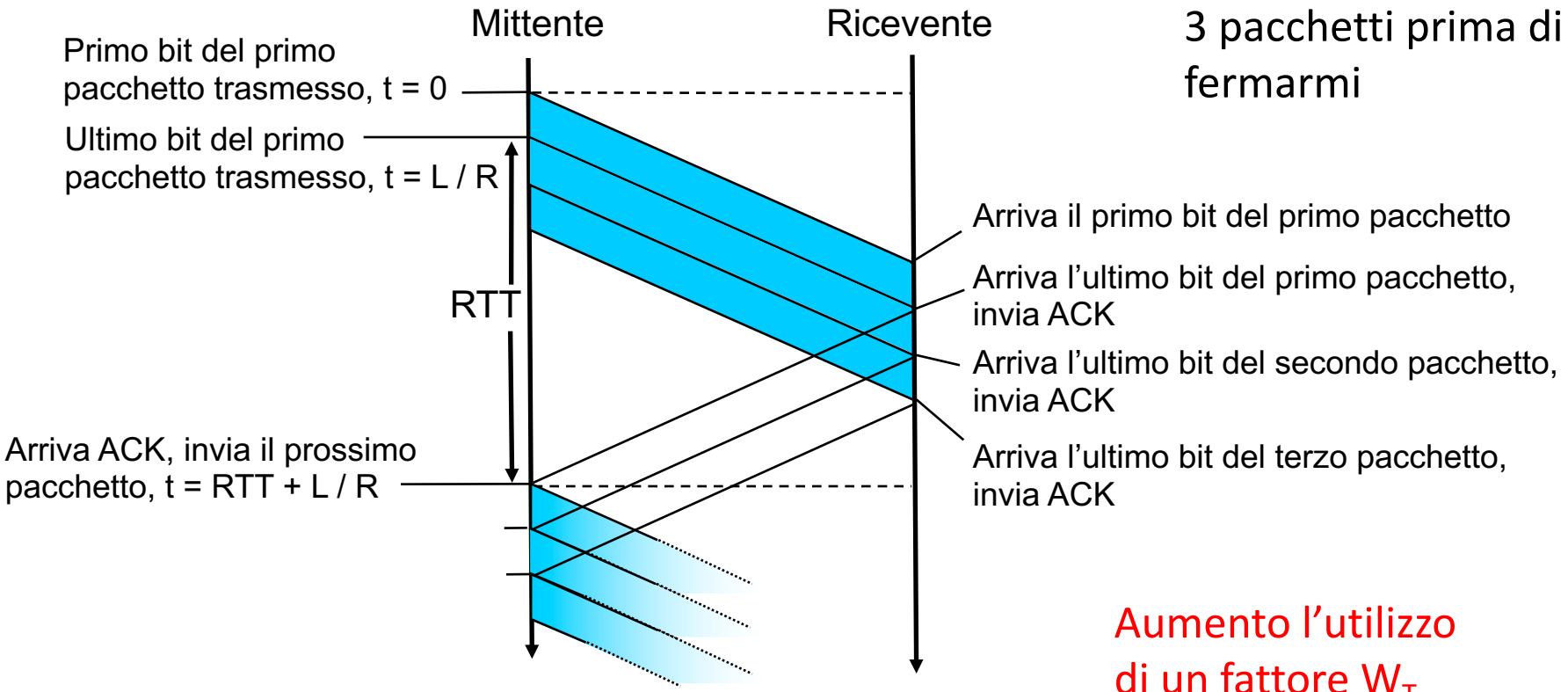


a) Protocollo stop-and-wait all'opera

b) Protocollo con pipeline all'opera

Pipelining: aumento dell'utilizzo

$W_T > 3$ trasmetto 3 pacchetti prima di fermarmi



Aumento l'utilizzo di un fattore W_T

$$W_T = 3; \quad U_{mitt} = \frac{3 * L / R}{RTT + L / R} = \frac{0{,}024}{30{,}008} = 0{,}0008$$

TCP ... finalmente

- Il “Transmission Control Protocol” è un protocollo a finestra con ACK cumulativi e ritrasmissioni selettive
- La finestra di ricezione e trasmissione possono essere variate dinamicamente durante la comunicazione
  - versioni e patch successive lo hanno reso uno dei protocolli più complessi della suite di Internet
- TCP è orientato alla connessione
- Affidabile
- Implementa controllo di flusso
- Cerca di controllare il traffico iniettato in funzione della congestione nella rete

TCP: riferimenti bibliografici

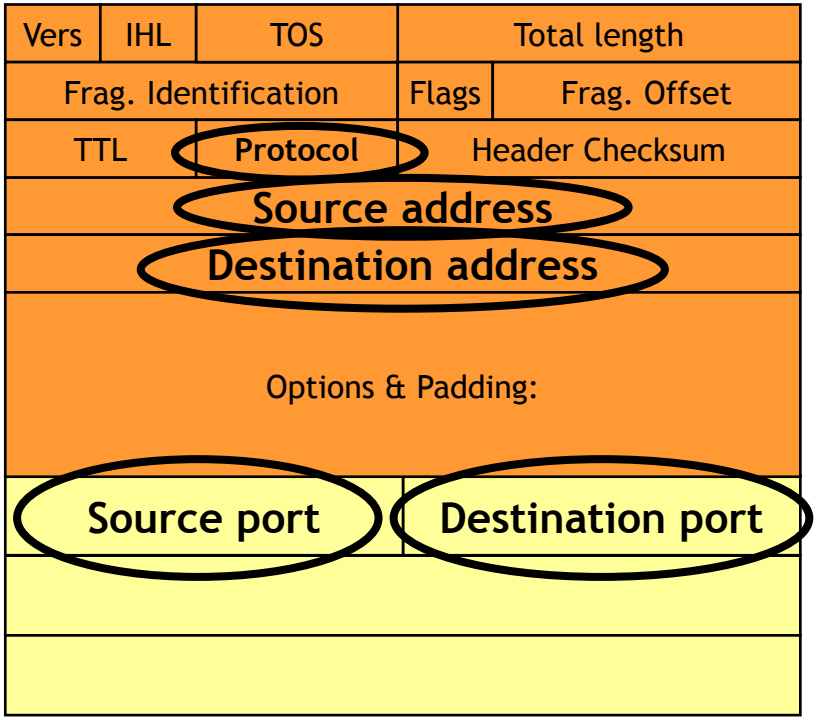
- Richard Stevens: TCP Illustrated, vol.1-2-3
- RFC 793 (1981): Transmission Control Protocol
- RFC 1122/1123 (1989): Requirements for Internet Hosts
- RFC 1323 (1992): TCP Extensions for High Performance
- RFC 2018 (1996) TCP Selective Acknowledgment Options
- RFC 2581: TCP Congestion Control
- RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm
- RFC 2883: An Extension to the Selective Acknowledgement (SACK) Option for TCP
- RFC 2988: Computing TCP's Retransmission Timer
- ...
- ...

TCP: compiti

- Fornisce porte per (de)multiplazione
- Una entità TCP di un host, quando deve comunicare con un'entità TCP di un altro host, crea una connessione fornendo un servizio simile ad un circuito virtuale
  - bidirezionale (full duplex)
  - con controllo di errore e di sequenza
- Mantiene informazioni di stato negli host per ogni connessione
- TCP segmenta e riassembla i dati secondo le sue necessità:
  - tratta stream di dati (byte) *non strutturati* dai livelli superiori
  - non garantisce nessuna relazione tra il numero di read e quello di write (buffer tra TCP e livello applicazione)

Identificazione di connessioni

- Una connessione TCP tra due processi è definita dai suoi endpoints (punti terminali), univocamente identificati da un socket:
  - Indirizzi IP host sorgente e host destinazione
  - Numeri di porta TCP host sorgente e host destinazione
- Nota: TCP o UDP usano porte **indipendenti**
- Esempio: connessione TCP tra porta 15320 host 130.192.24.5 e porta 80 host 193.45.3.10



Trasmettitore TCP

- Suddivide i dati dell'applicazione in segmenti
- Usa protocollo a finestra dinamica con $W_T \geq 1$
- Attiva timer quando invia i segmenti:
  - segmenti non confermati allo scadere del timer (Retransmission TimeOut - RTO) provocano ritrasmissioni
- Stima RTT per impostare RTO
- Calcola e trasmette checksum obbligatorio su header e dati
- Regola velocità con dimensione finestra
  - controllo di flusso e congestione

Ricevitore TCP

- Riordina segmenti fuori sequenza e scarta segmenti errati
  - consegna stream ordinato e corretto al processo applicativo
- Invia ACK **cumulativi**
- Annuncia negli ACK lo spazio libero nel buffer di ricezione per controllare velocità trasmettitore (controllo di flusso)

Ricevitore TCP

- Segmento corretto ed in sequenza
  - Memorizza (ed eventualmente consegna al livello superiore) ed invia ACK cumulativo
- Segmento duplicato
  - Scarta ed invia ACK relativo all'ultimo segmento ricevuto in sequenza
- Segmento con checksum errato
  - Scarta senza inviare ACK
- Segmento fuori sequenza
  - Memorizza ed invia ACK relativo all'ultimo segmento ricevuto in sequenza (ACK duplicato)

TCP: generazione ACK

Eventi	**Azioni ricevitore TCP**
arrivo segmento in ordine, inviato ACK correttamente per tutti segmenti precedenti	**invia ACK**
arrivo segmento fuori sequenza con numero maggiore di quello atteso: vuoto rilevato	**invia ACK duplicato, indicando come numero di sequenza il prossimo byte che si attende di ricevere**
arrivo di segmento che riempie vuoti parzialmente o completamente	**ACK immediato, indicando come numero di sequenza il prossimo byte che si attende di ricevere**

Throughput ottenibile

- Generico protocollo a finestra: la velocità di trasmissione in assenza di errori è

$$\frac{\text{Finestra di trasmissione}}{\text{Round trip time}}$$

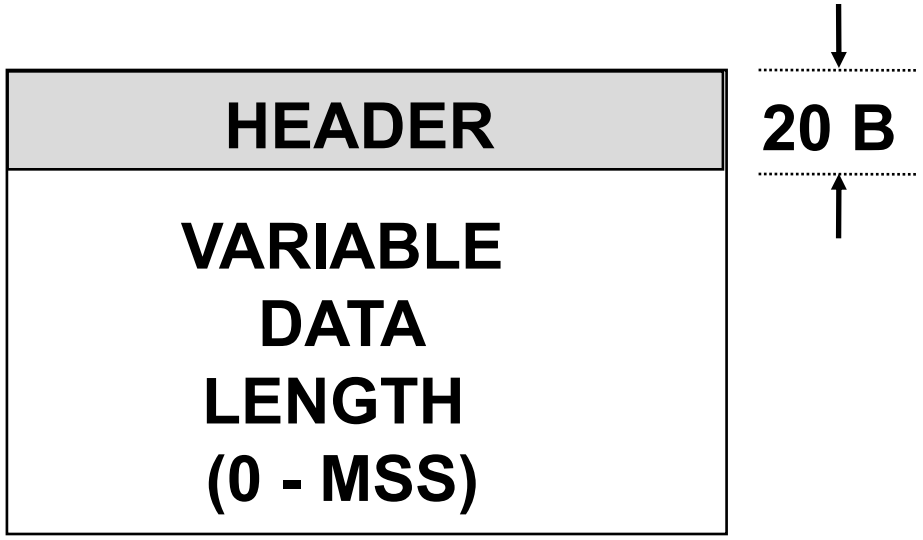
  - Connessioni “corte” ottengono banda maggiore a parità di finestra
- Per regolare velocità di trasmissione posso agire su
  - round trip time
  - dimensione finestra

- TCP impiega un controllo end-to-end basato su controllo della dimensione della finestra del trasmettitore:
  - evita che un host veloce saturi un ricevitore lento
  - il ricevitore impone la dimensione massima della finestra del trasmettitore, indicando negli ACK la **finestra di ricezione disponibile**
  - valore dinamico che dipende dall'allocazione del sistema operativo e dalla quantità di dati ricevuti e non ancora letti dall'applicazione

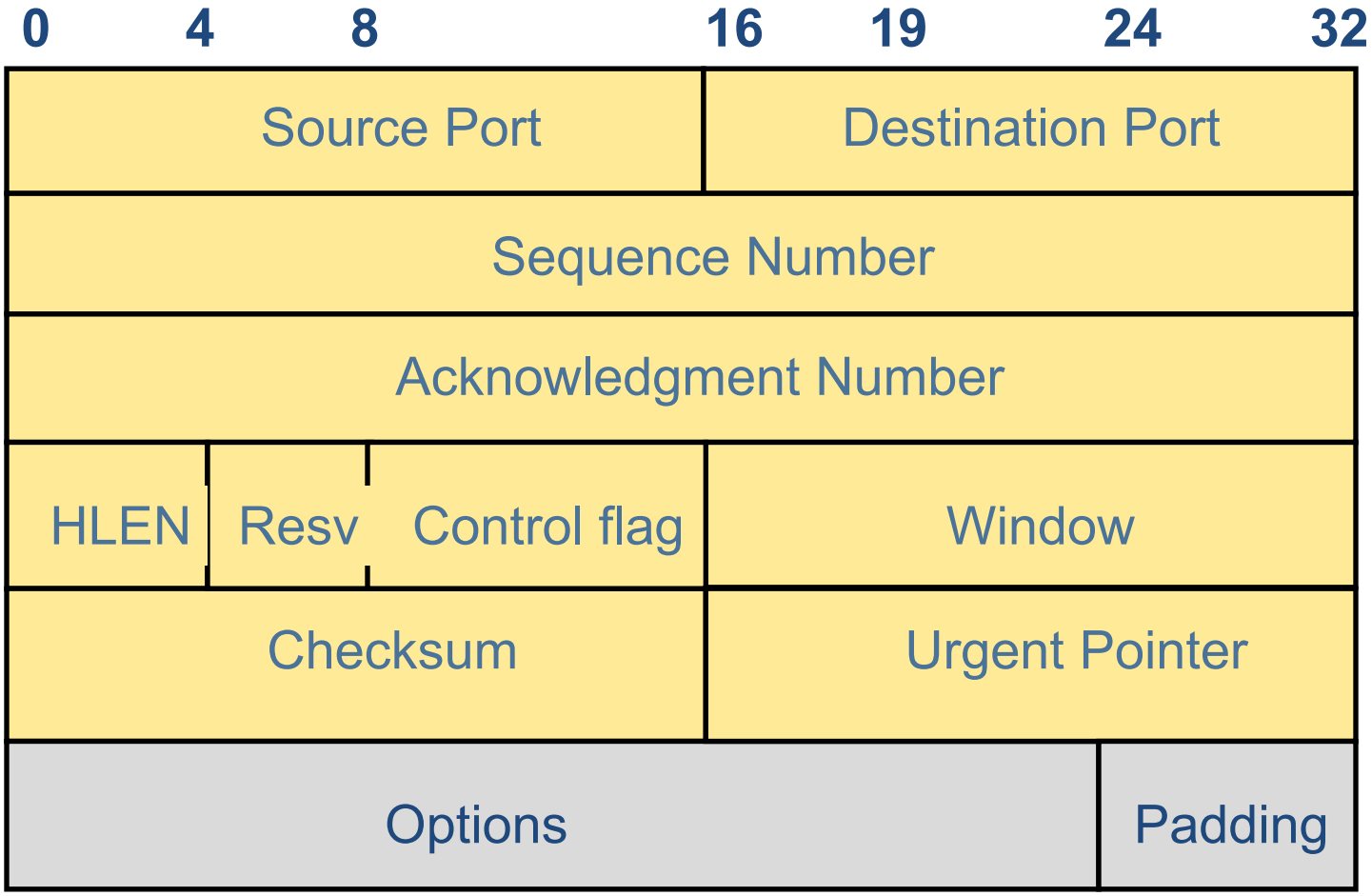
- Deve evitare che gli host nel loro insieme siano troppo aggressivi saturando la rete
- Non posso trasmettere sempre alla velocità massima consentita dal ricevitore
- Se la finestra di trasmissione è maggiore del valore per cui il bit rate in trasmissione supera capacità del collo di bottiglia
  - si memorizzano dati nei buffer lungo il percorso, e cresce round trip time
  - si perdono dati nei buffer dei nodi intermedi

La PDU TCP

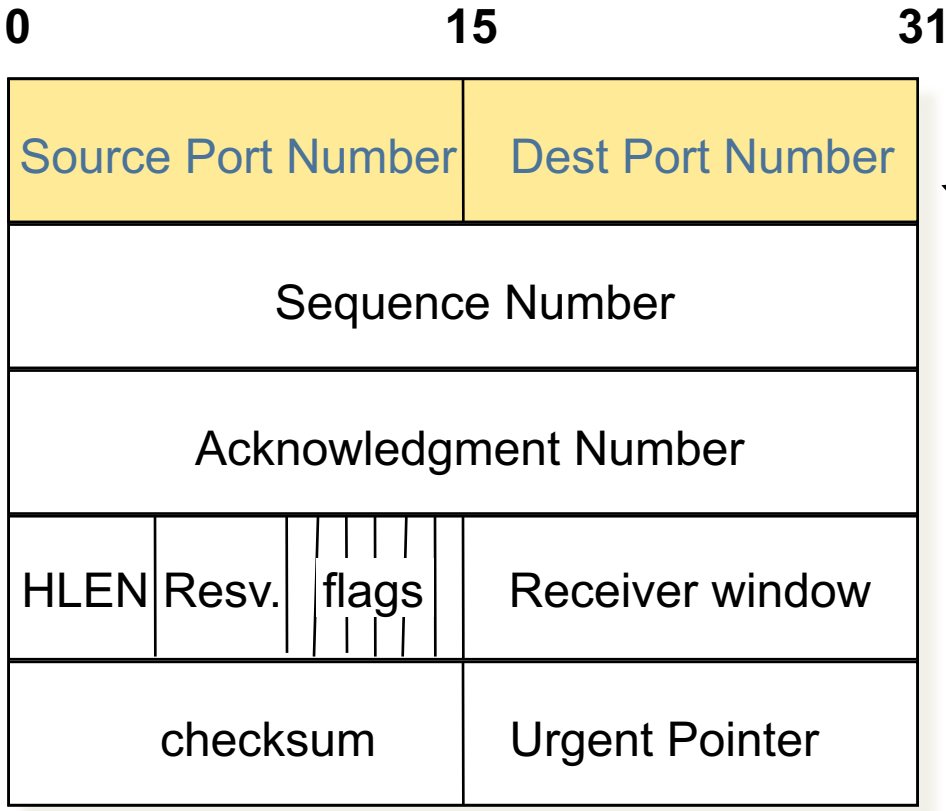
- La PDU di TCP è detta *segmento*
- La dimensione dei segmenti può variare dal solo header (ACK, 20 byte) fino ad un valore massimo MSS dipendente dalla MTU IP
- La dimensione del singolo segmento dipende dallo stream dei livelli superiori



TCP header

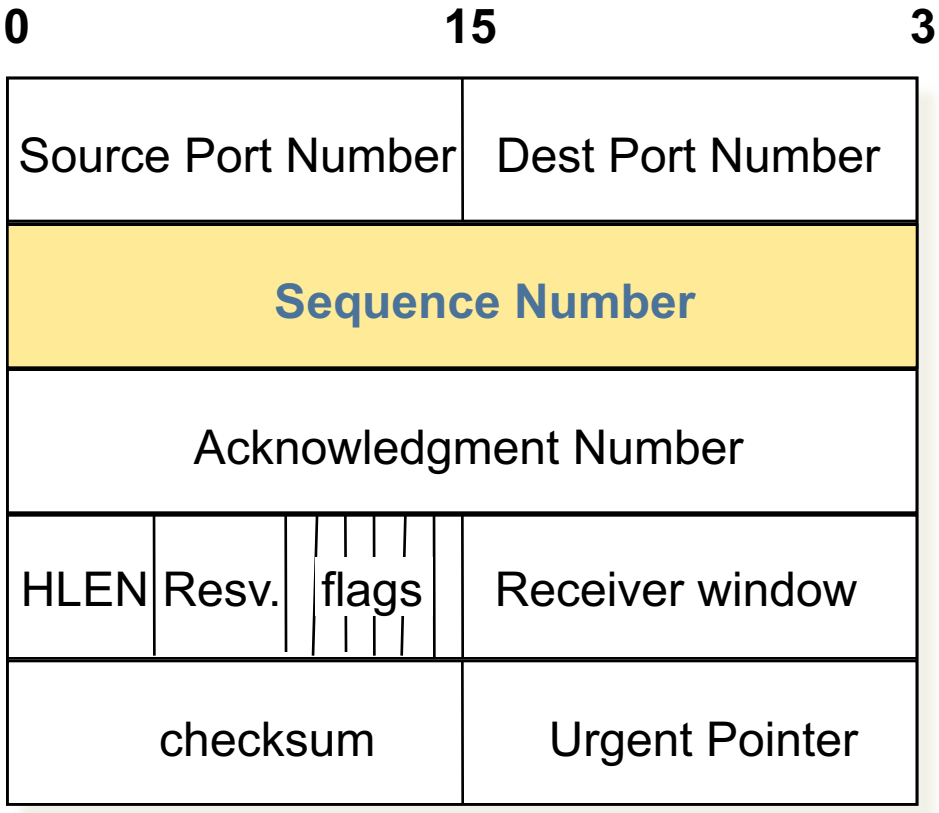

0
4
8
16
19
24
32
Source Port
Destination Port
Sequence Number
Acknowledgment Number
HLEN
Resv
Control flag
Window
Checksum
Urgent Pointer
Options
Padding

TCP header



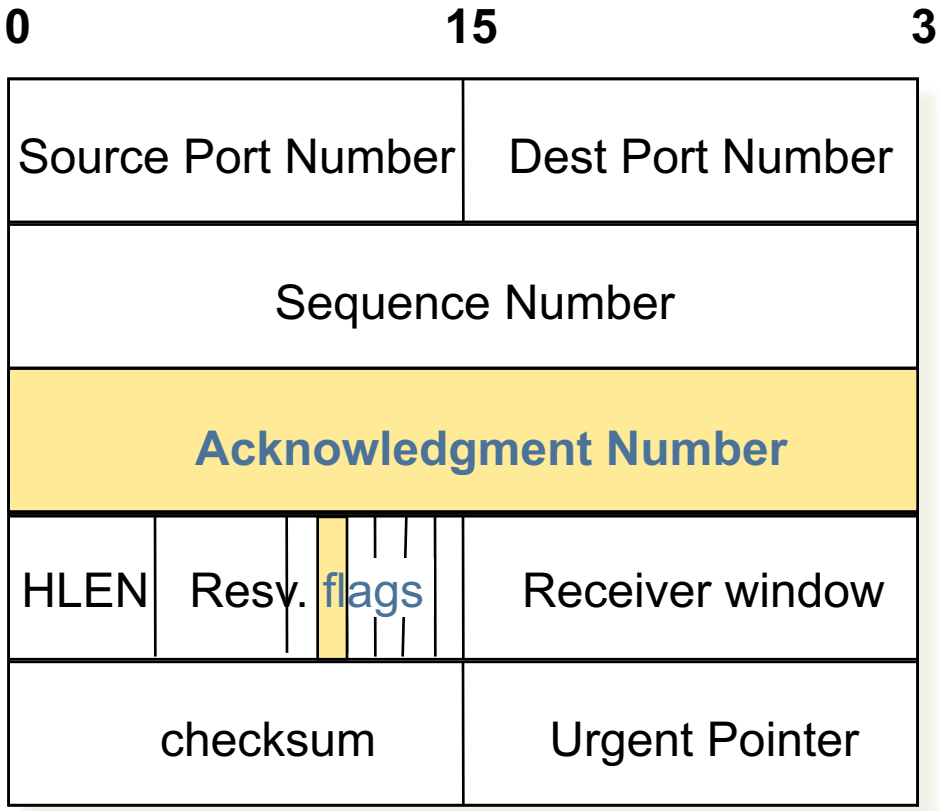
Identificano l'applicazione che sta inviando e ricevendo dati. Combinati con i rispettivi indirizzi IP, identificano in modo univoco una connessione

TCP header



- Identifica, nello stream di dati, la posizione del primo byte del payload del segmento
- **Numerazione ciclica su 32 bit**
- Ogni direzione della connessione procede con numeri di sequenza diversi e indipendenti

TCP header



Numero di sequenza più 1 dell'ultimo byte di dati ricevuto correttamente e in sequenza

Ovvero il prossimo byte che il ricevitore si aspetta di ricevere

Valido solo con ACK flag settato

Numeri di sequenza e di ack

Host A

Host B

Utente preme tasto 'C'

Seq=42, ACK=79, data = 'C'

host conferma ricezione di 'C', ed invia echo

Seq=79, ACK=43, data = 'C'

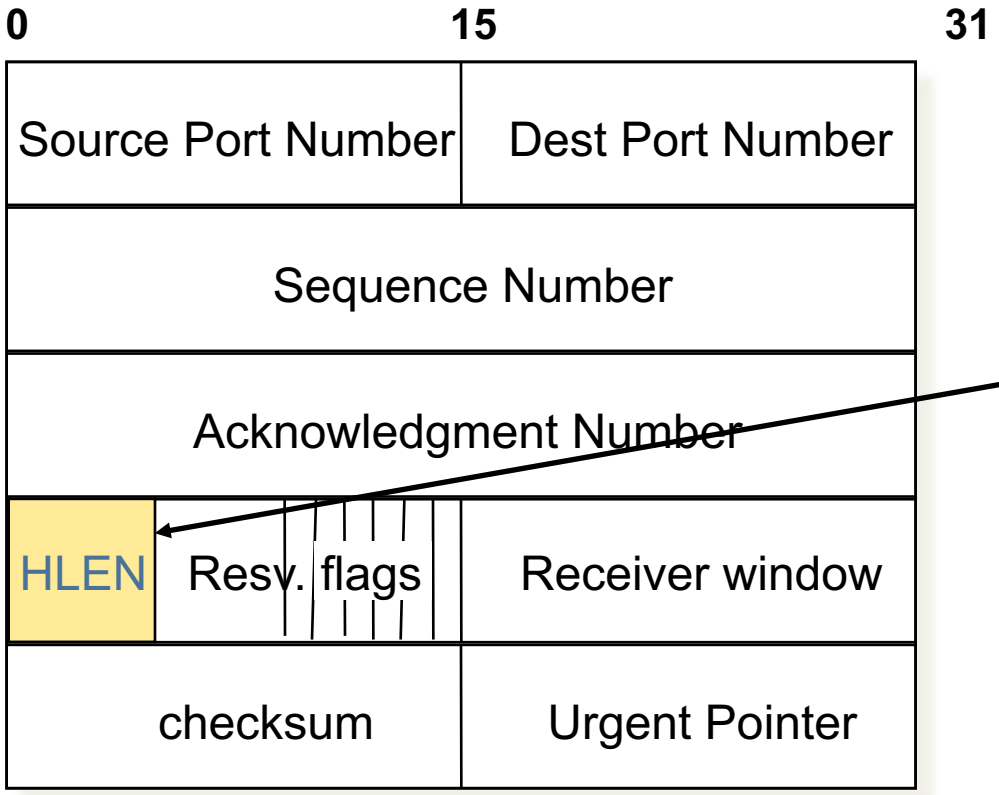
host conferma ricezione dell'eco 'C'

Seq=43, ACK=80

tempo

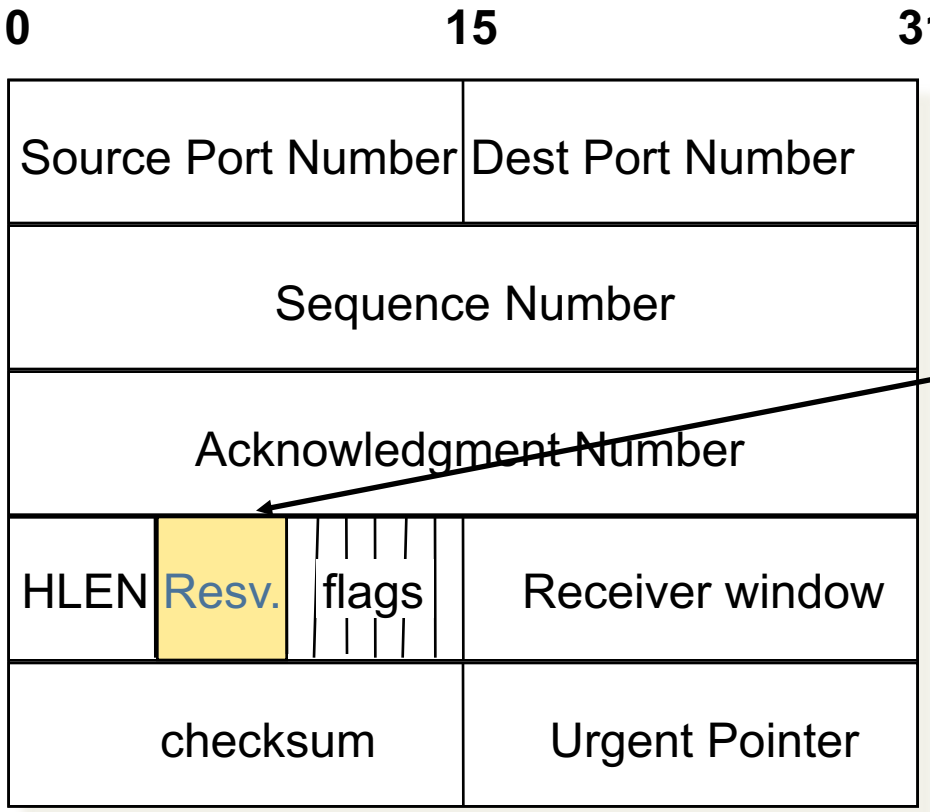
Esempio di sessione telnet

TCP header



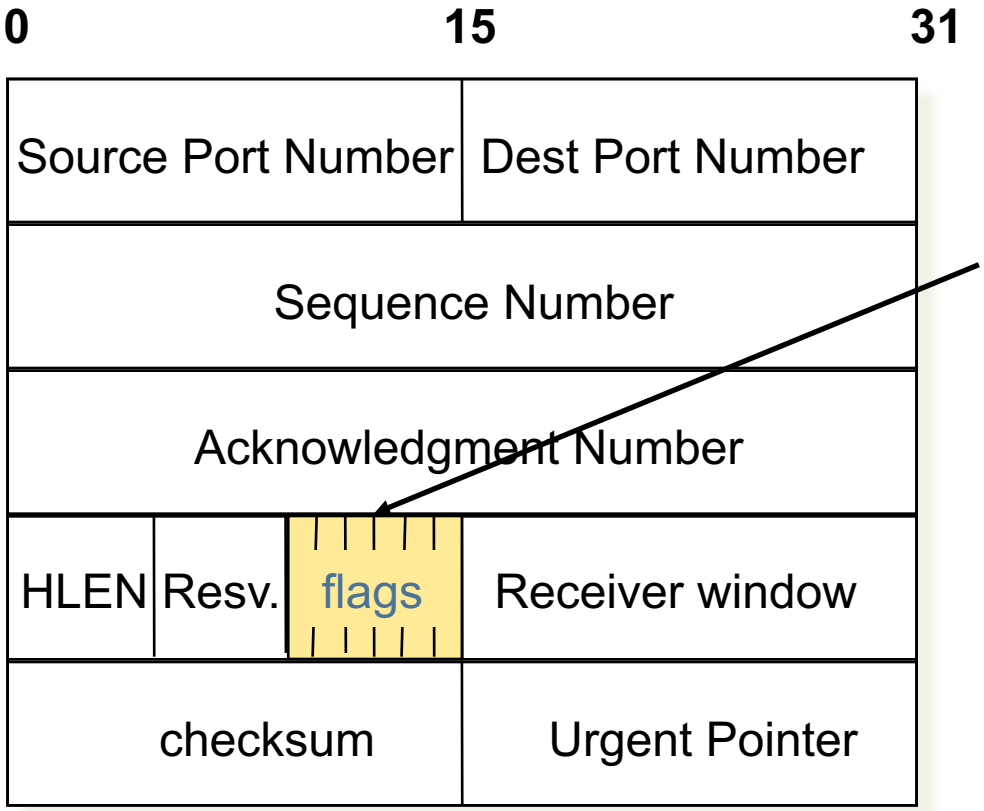
Lunghezza dell'header in parole di 32 bit

TCP header



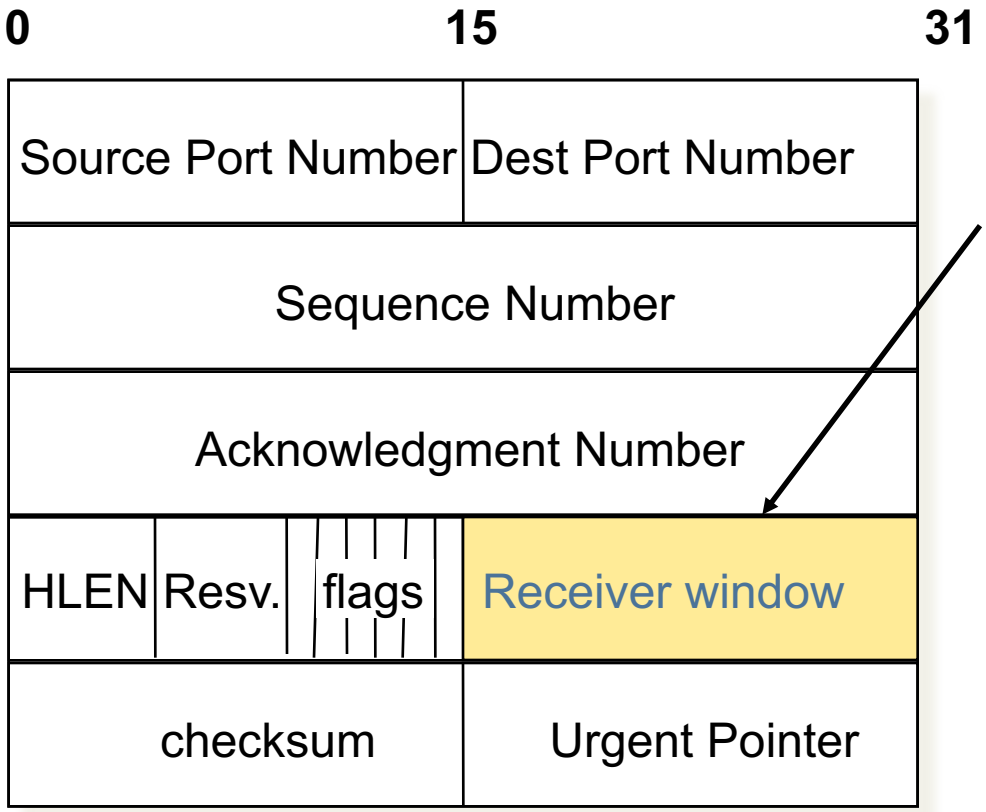
RiservatI per usi futuri (es. ECN – Explicit Congestion Notification)

Gestione connessione



- Sei bit di flag, uno o più possono essere settati insieme:
  - URG: urgent pointer valido
  - ACK: numero di ack valido
  - PSH: forza passaggio dati applicazione
  - RST: reset connessione
  - SYN: synchronize seq. No. Apertura connessione
  - FIN: chiusura connessione

TCP header



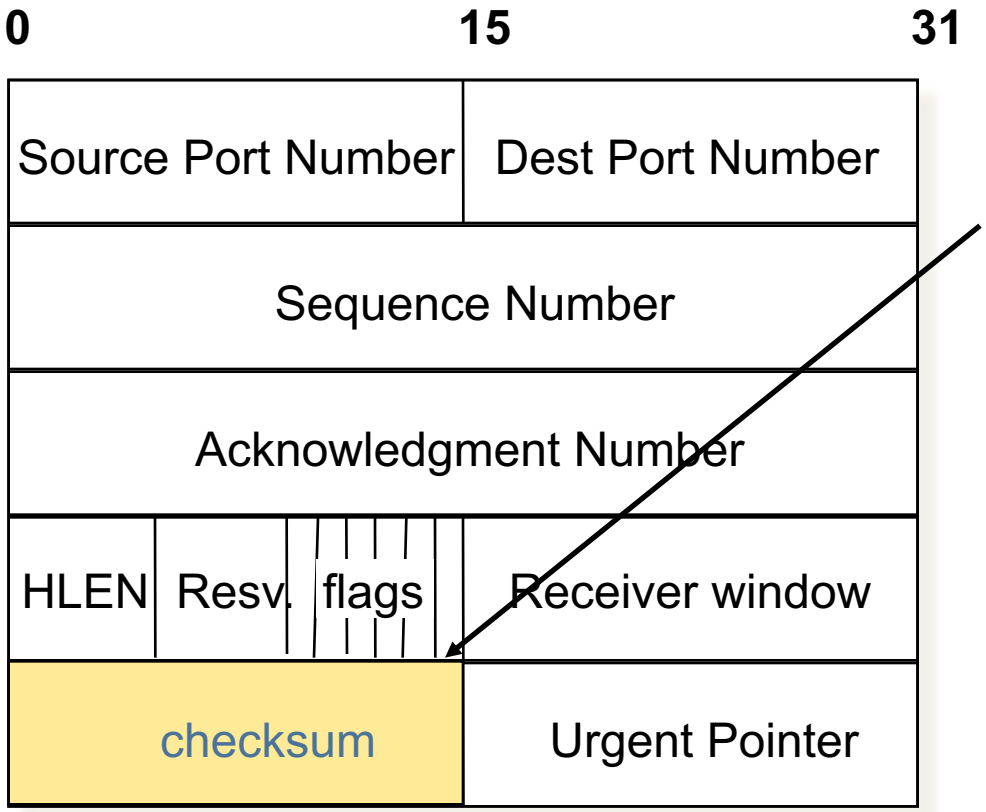
Numero di byte, a partire da quello nel campo di ACK, che il ricevitore è disposto ad accettare per controllo di flusso
Valore massimo rwnd 65535 byte se non si usa l'opzione "window scale"

- Massima quantità di dati in transito per RTT:
  - 16-bit rwnd = 64KB max
- RTT=100ms

Prodotto banda x ritardo dato

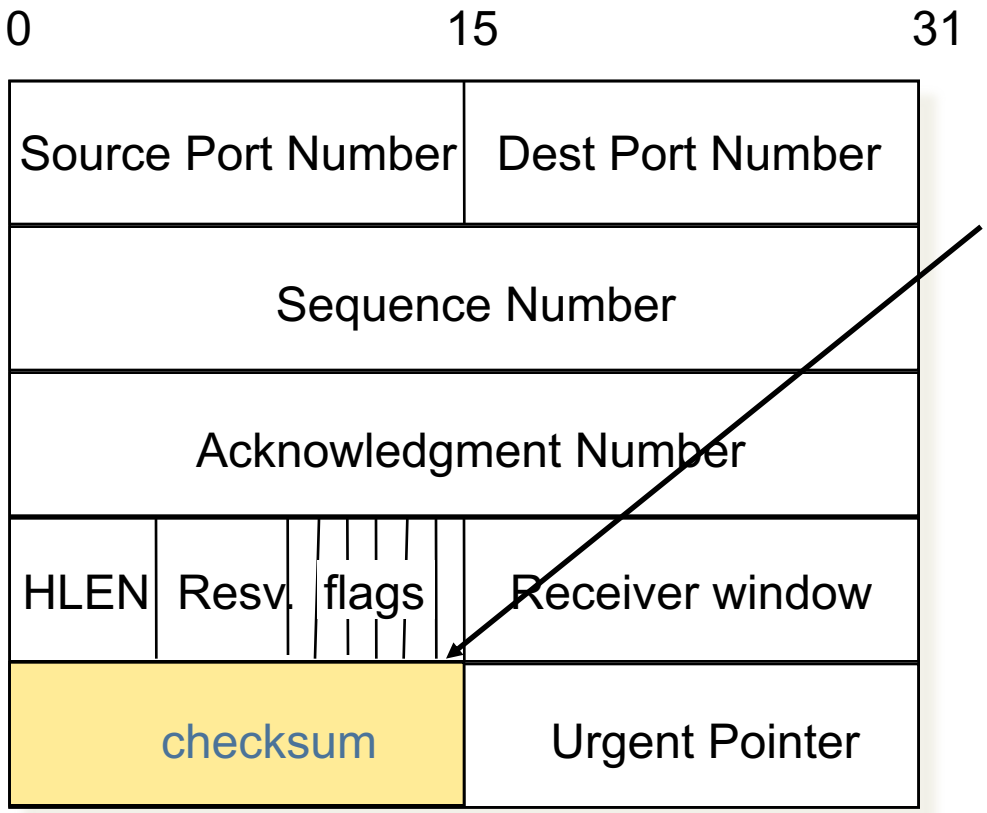
Banda		banda x ritardo
T1	(1.5Mbps)	18KB
Ethernet	(10Mbps)	**122KB**
T3	(45Mbps)	**549KB**
FastEthernet	(100Mbps)	**1.2MB**
STS-3	(155Mbps)	**1.8MB**
STS-12	(622Mbps)	**7.4MB**
STS-48	(2.5Gbps)	**29.6MB**

TCP header



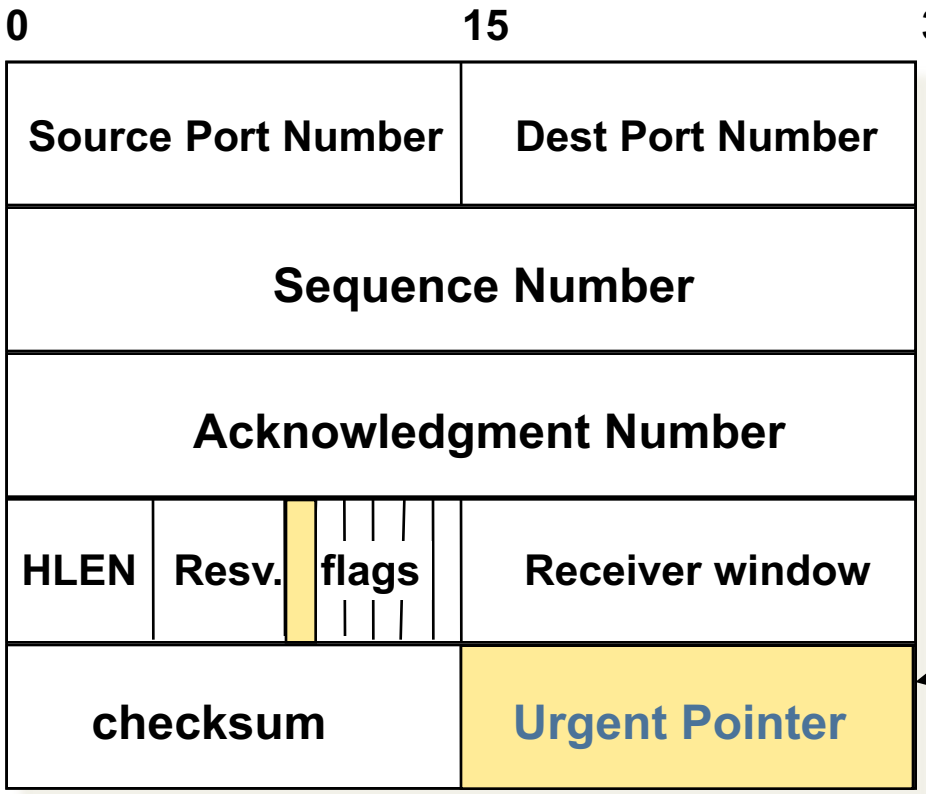
Checksum obbligatorio su header e dati, più pseudo-header che include indirizzi IP e tipo di protocollo (violazione del principio di stratificazione OSI)

TCP header



- Algoritmo di checksum
  - allineamento di header, dati e pseudo-header su 16 bit
  - somma in complemento a 1 di ogni riga
  - si ottiene numero a 32 bit, che si divide in due parti di 16 bit
  - somma in complemento a 1 delle due parti, incluso il riporto
  - inserisco nell'header i 16 bit risultanti

TCP header

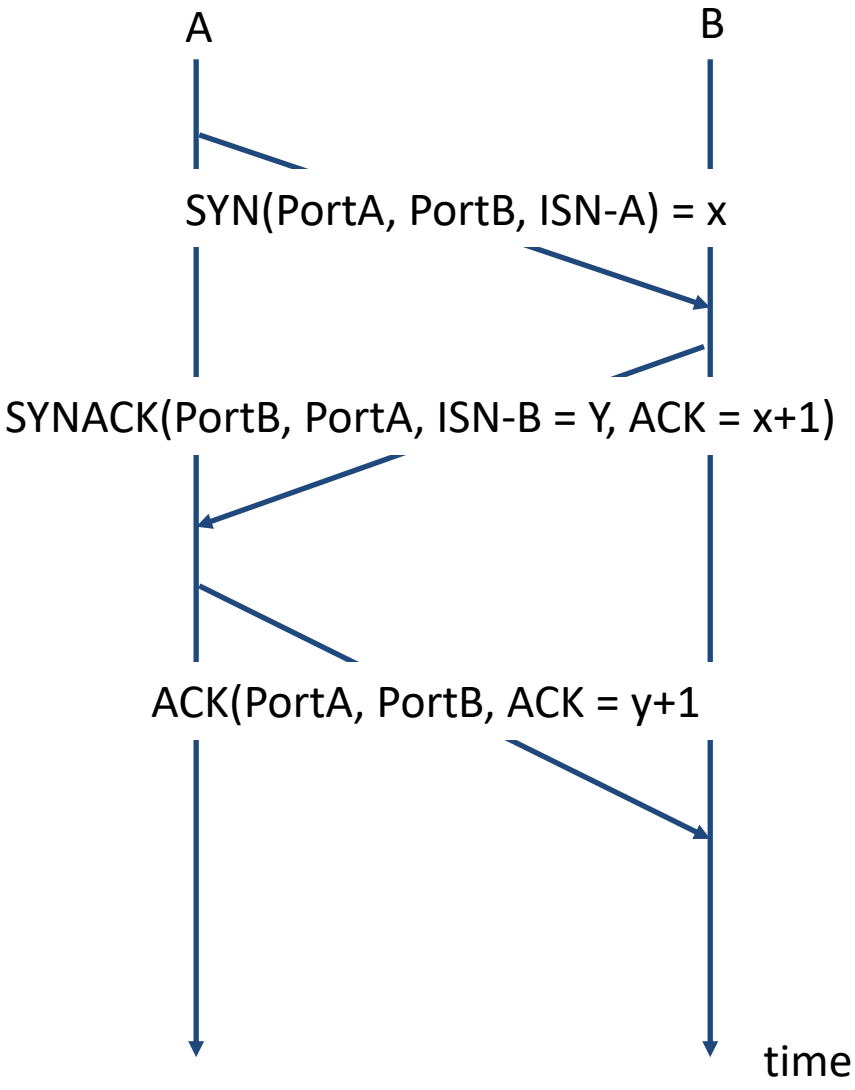


Puntatore a “dati urgenti” nel campo dati (es. ctrl-C in una sessione telnet).
Offset rispetto al num. di seq.

Valido solo se flag URG è settato

Instraurazione della connessione

- L'instaurazione della connessione avviene secondo la procedura detta di "three-way handshake"
- la stazione che richiede la connessione (A) invia un segmento di SYN
- parametri specificati: numero di porta dell'applicazione cui si intende accedere e Initial Sequence Number (ISN-A)
- la stazione che riceve la richiesta (B) risponde con un segmento SYN
- parametri specificati: ISN-B e riscontro (ACK) ISN-A
- la stazione A riscontra il segmento SYN della stazione B (ISN-B)



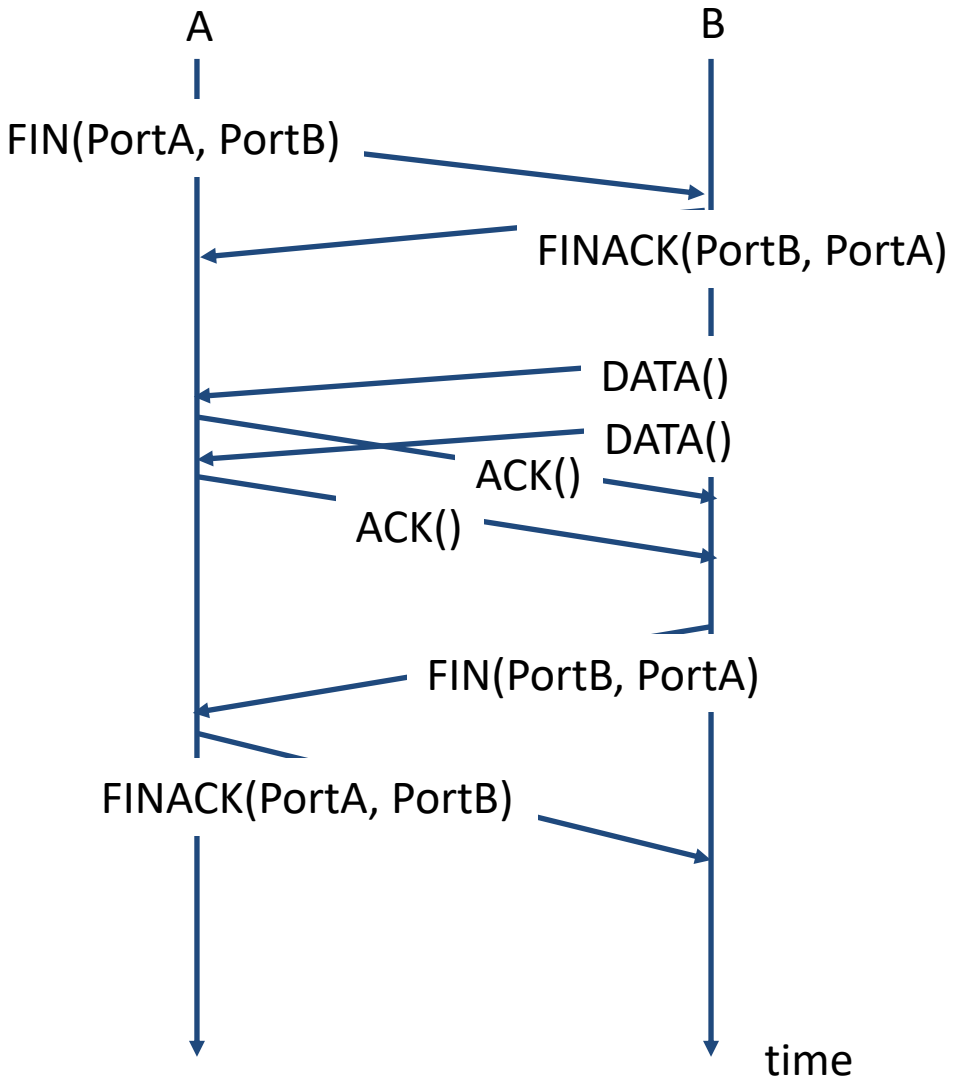
Come si calcola MSS

- MSS dipende dalla MTU (Maximum Transfer Unit) del livello IP, che a sua volta dipende dal livello Data-Link del collegamento con MTU più piccola lungo il percorso
- Purtroppo non esistono meccanismi di segnalazione per comunicare MSS
- TCP può cercare di stimare la MTU minima con un meccanismo di "trial and error"
- Prova segmenti sempre più grossi fino a quando uno viene scartato per dimensione eccessiva e il dispositivo che l'ha scartato manda un messaggio ICMP (v. livello rete) al mittente
- ... purtroppo non tutti i dispositivi rispondono ...
- Default: MSS = 1460 (1500 trama ethernet – 40 bytes di headers)
- Default "minimo": MSS = 536

Terminazione della Connessione

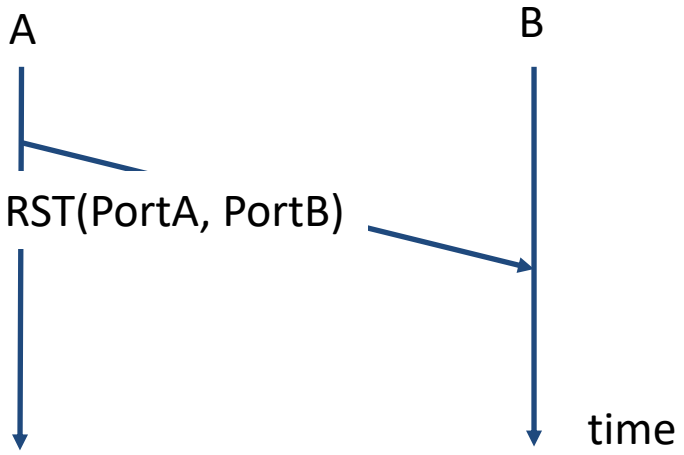
- Poiché la connessione è bidirezionale, la terminazione deve avvenire in entrambe le direzioni
- Procedura di terminazione “gentle”
  - la stazione che non ha più dati da trasmettere e decide di chiudere la connessione invia un segmento FIN (segmento con il campo FIN posto a 1 e il campo dati vuoto)
  - la stazione che riceve il segmento FIN invia un ACK e indica all’applicazione che la comunicazione è stata chiusa nella direzione entrante
- Se questa procedura avviene solo in una direzione (half close), nell’altra il trasferimento dati può continuare (gli ACK non sono considerati come traffico originato, ma come risposta al traffico)
  - per chiudere completamente la connessione, la procedura di half close deve avvenire anche nell’altra direzione

Terminazione della Connessione



Terminazione con RST

- La terminazione “gentle” è molto lenta e complessa, e si presta a attacchi DoS
- La maggior parte delle applicazioni moderne (soprattutto i grandi web server) terminano la connessione con un singolo RST
- RST nasce come “disaster recovery” per liberare le risorse logiche in caso di impossibilità di comunicazione
- Se l’altro end-point della connessione riceve RST libera anche lui le risorse logiche



Gestione del timeout

- Il timeout (RTO → Retransimission Time Out) indica il tempo entro il quale la sorgente si aspetta di ricevere il riscontro (ack)
  - nel caso in cui il riscontro non arrivi, la sorgente procede alla ritrasmissione
- RTO non può essere un valore statico predefinito
  - il tempo di percorrenza sperimentato dai segmenti è variabile e dipende
    - dalla distanza tra sorgente e destinazione
    - dalle condizioni della rete
    - dalla disponibilità della destinazione
  - esempio: fattorino che deve consegnare un pacco in città
- RTO deve dunque essere calcolato dinamicamente di volta in volta
  - durante la fase di instaurazione della connessione
  - durante la trasmissione dei dati
- Il calcolo di RTO si basa sulla misura del RTT (Round Trip Time)
  - RTT: intervallo di tempo tra l'invio di un segmento e la ricezione del riscontro di quel segmento

$$SRTT = (1- \alpha)\, SRTT + \alpha\, RTT$$

RTT = valore del campione attuale di RTT
SRTT = stima smoothed di RTT

- Poiché RTT può variare anche molto in base alle condizioni della rete, il valore di RTT (SRTT, Smoothed RTT) utilizzato per il calcolo di RTO risulta una stima del valor medio di RTT sperimentato dai diversi segmenti
- Il parametro α è regolabile e, a seconda dei valori assunti, rende il peso della misura di RTT istantaneo più o meno incisivo
  - se α → 0 il SRTT stimato risulta abbastanza stabile e non viene influenzato da singoli segmenti che sperimentano RTT molto diversi
  - se α → 1 il SRTT stimato dipende fortemente dalla misura puntuale dei singoli RTT istantanei
  - tipicamente α = 0.125 = (1/8)

RTTVAR = (1- b) RTTVAR + b |SRTT – RTT|

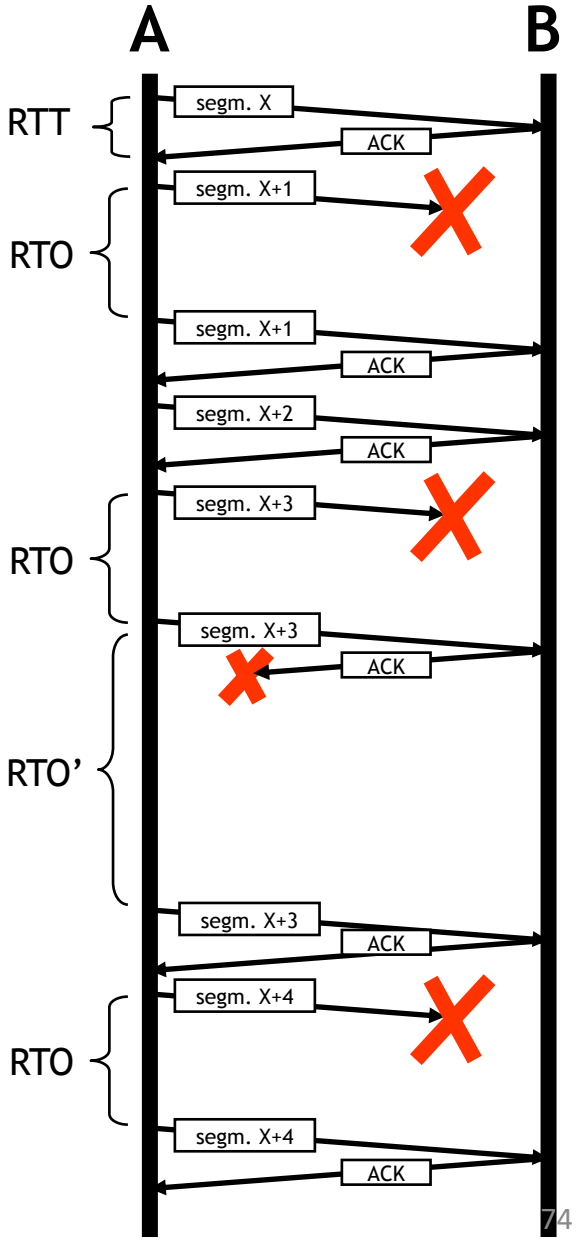
RTTVAR = stima smoothed della varianza di RTT

- Tipicamente b = 0.25 = (1/4)
- La RTTVAR può anche essere molto maggiore di SRTT
- La sua stima usa un filtro meno stretto, per cui la stima è più “reattiva”
- Conoscere media e varianza di RTT serve a impostare correttamente il timeout di ritrasmissione

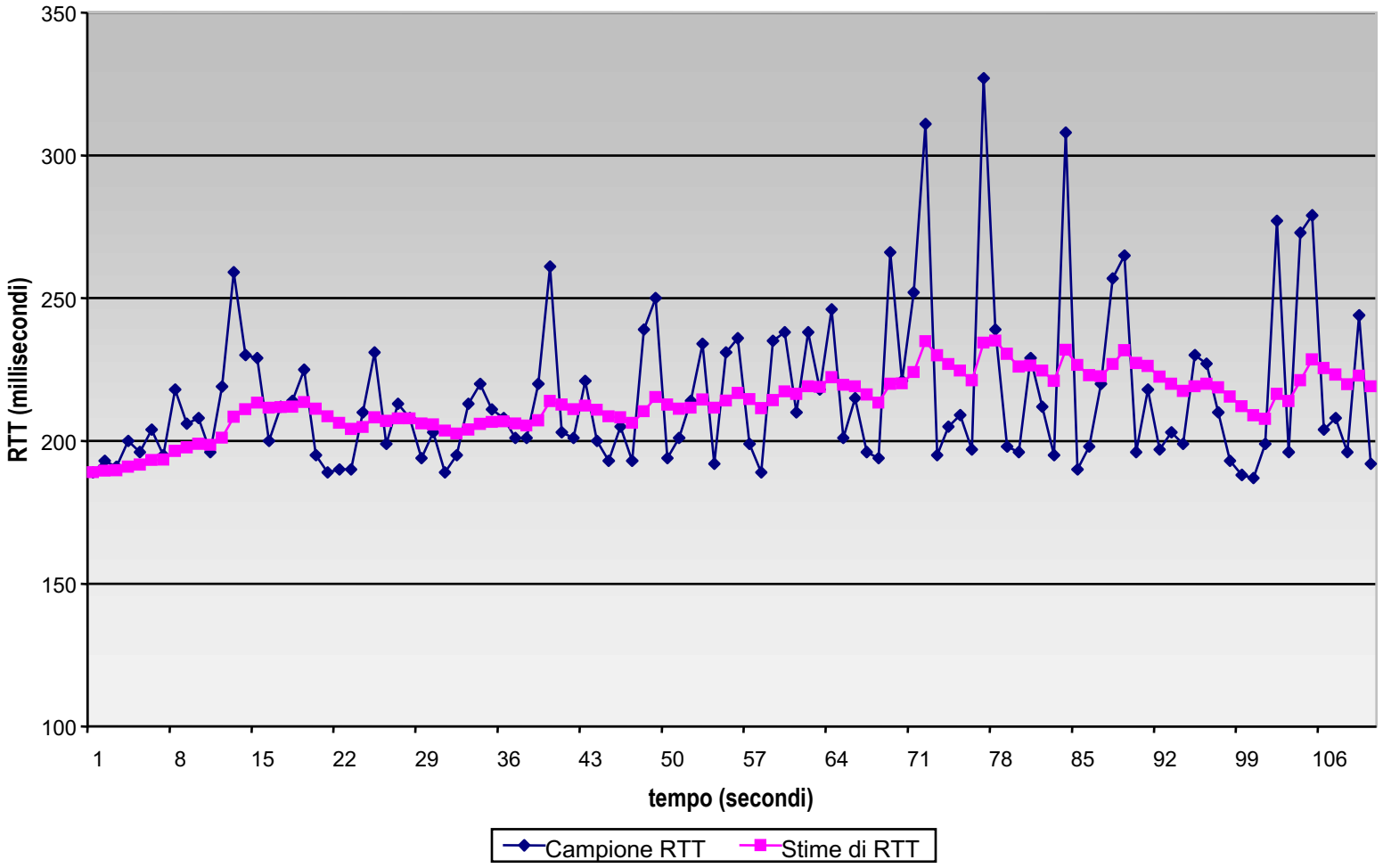
Stima di RTO

- **RTO = SRTT + 4 RTTVAR**

- La sorgente attende il RTT medio (SRTT) più quattro volte la sua varianza (RTTVAR) prima di considerare il segmento perso e ritrasmetterlo
- In caso di ritrasmissione, il RTO per quel segmento viene ricalcolato in base ad un processo di exponential backoff
  - se è scaduto il RTO, probabilmente c'è congestione, quindi meglio aumentare il RTO per quel segmento
  - RTO-retransmission = 2*RTO
- RTO viene riportato al suo valore "calcolato" senza backoff dopo la trasmissione corretta di un segmento nuovo



Esempio di stima di RTT:



Stima di RTT e RTO

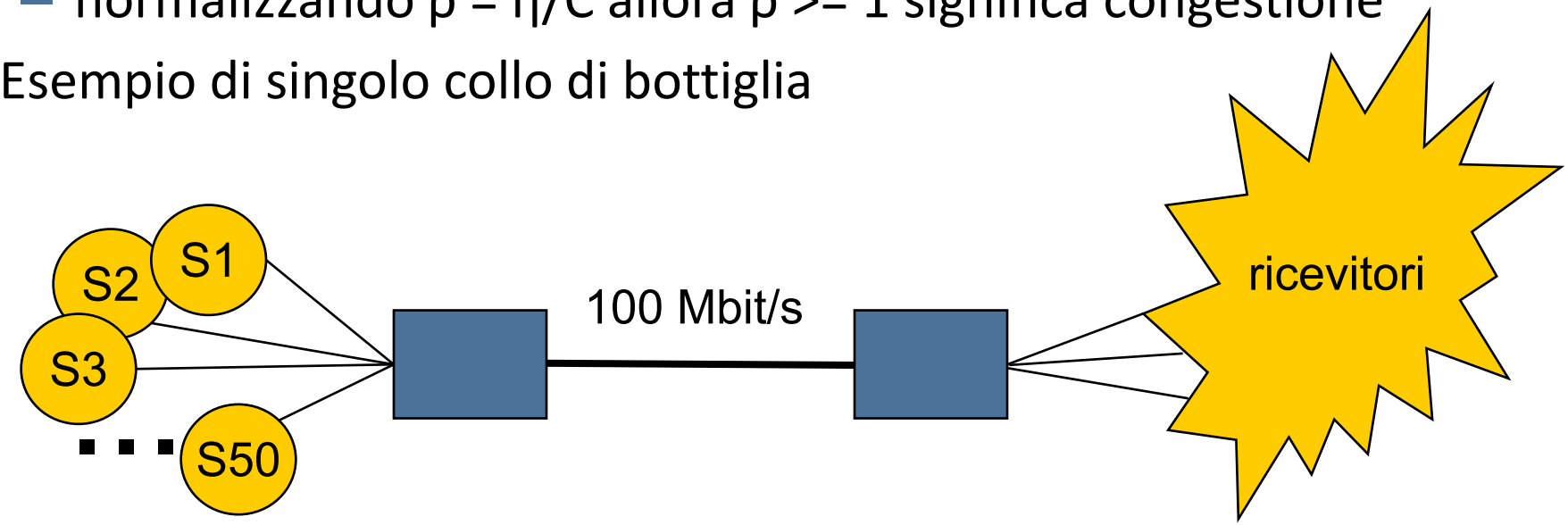
- RTO ha un valore minimo (normalmente intorno a 200ms) e uno massimo (normalmente 60s e oltre)
- Implementazioni moderne usano l'opzione "timestamp" per il calcolo di RTT, ma i principi base di adattamento alle condizioni della rete rimane lo stesso:
  - viene stimato un valore medio e una deviazione media di RTT
  - RTO viene calcolato in base a questi valori medi

- Nelle implementazioni, inoltre, si tiene conto di altri fattori che possono influenzare il calcolo di RTT e di RTO:
  - esempio: il RTT dei segmenti ritrasmessi dovrebbe influenzare il SRTT e quindi il RTO?

Congestione: generalità

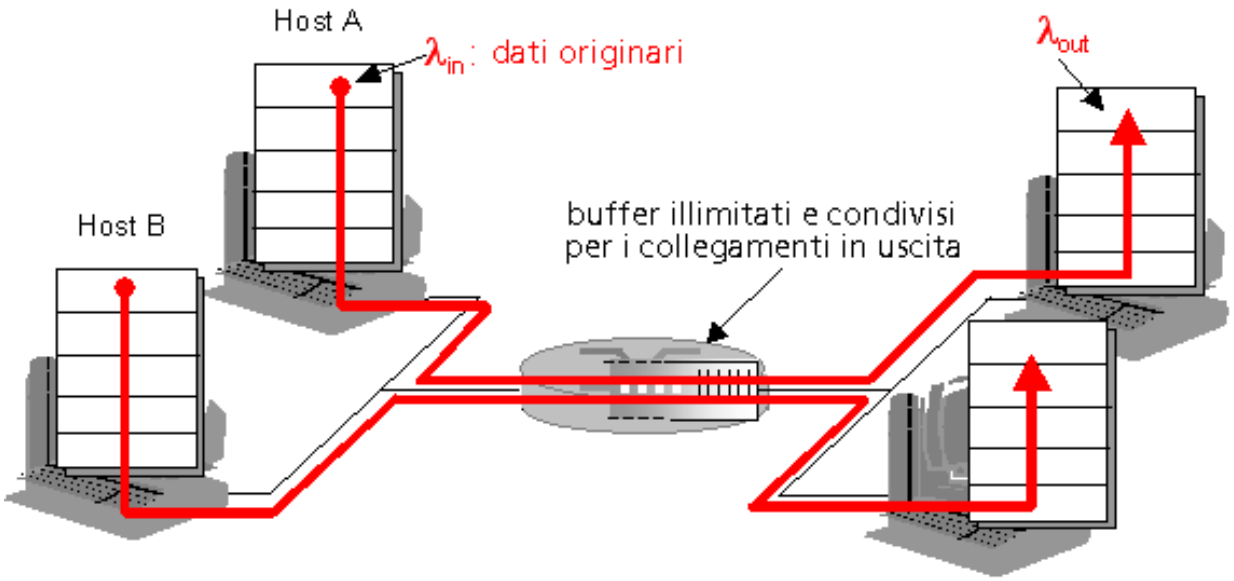
- In caso di congestione della rete, a causa dei buffer limitati degli apparati di rete, alcuni segmenti potrebbero venire persi
- La perdita dei segmenti e il relativo scadere del timeout di ritrasmissione è considerato un sintomo di congestione
  - TCP non ha altri mezzi per conoscere lo stato della rete
- La sorgente dovrebbe essere in grado di reagire diminuendo il tasso di immissione dei nuovi segmenti
- Questa reazione viene detta “controllo della congestione”
  - si differenzia dal controllo di flusso che serve a evitare il sovraccarico del ricevitore

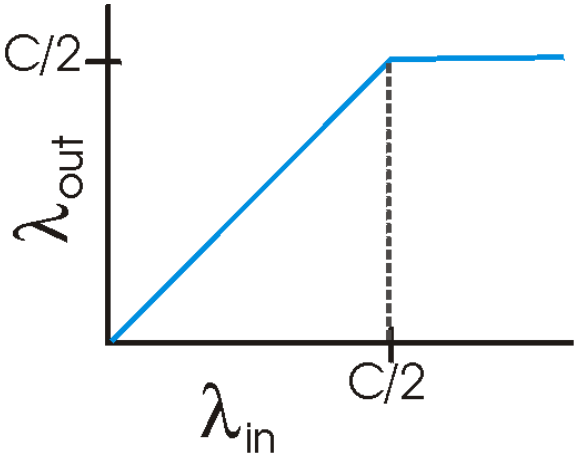
- Le ritrasmissioni rischiano di essere inutili nel caso in cui i pacchetti vengono buttati via per mancanza di risorse
- Cos'è la congestione e come si genera?
- **Congestione**: lo stato di una rete in cui il traffico offerto η è maggiore della capacità delal rete C
  - normalizzando ρ = η/C allora ρ >= 1 significa congestione
- Esempio di singolo collo di bottiglia

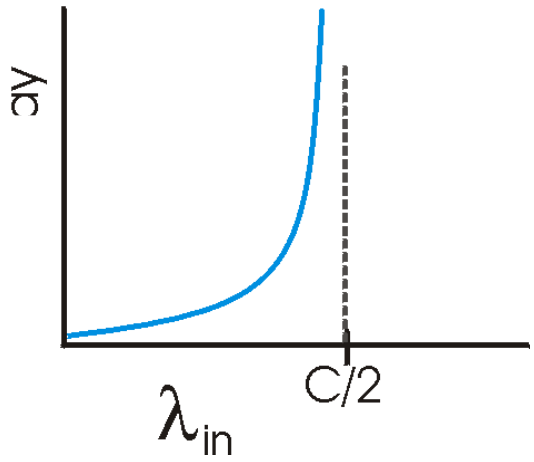


Cause/costi della congestione:1

- due mittenti, due destinatari
- un router con buffer illimitati
- nessuna ritrasmissione

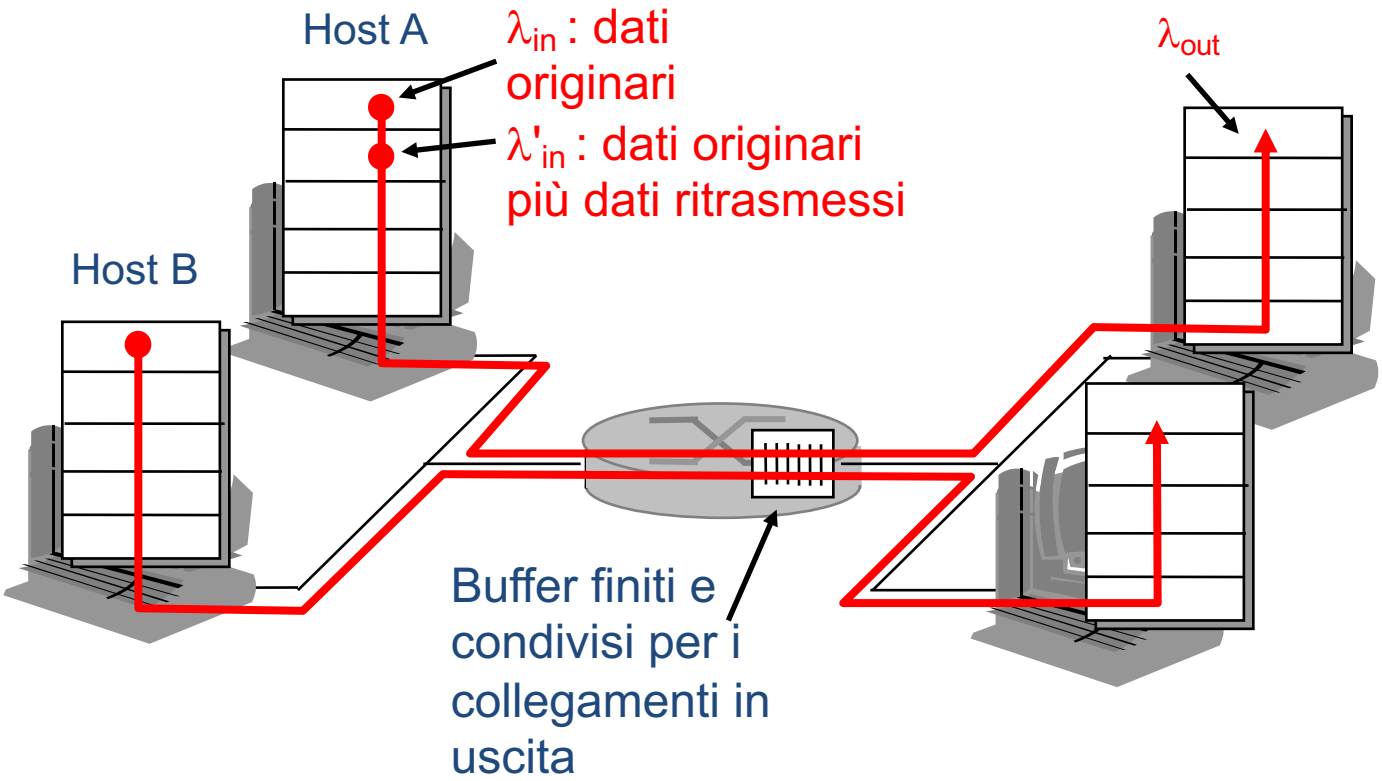






- throughput massimo
- ritardo “infinito”

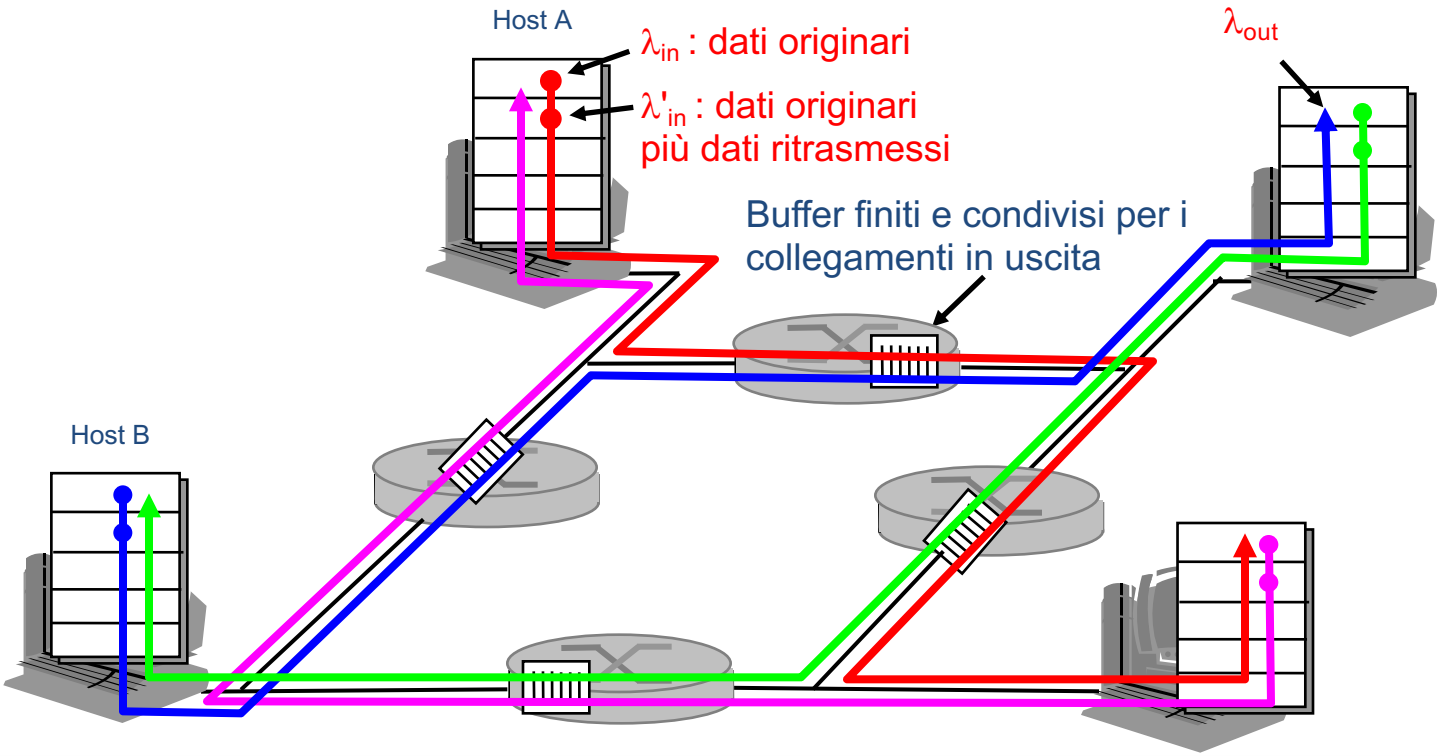
- un router, buffer *finiti*
- il mittente ritrasmette il pacchetto perduto

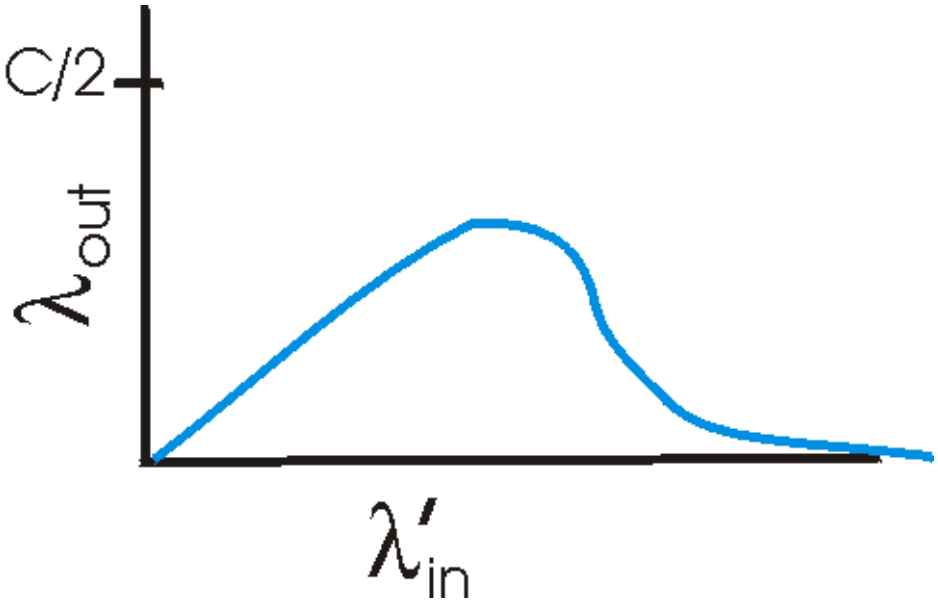


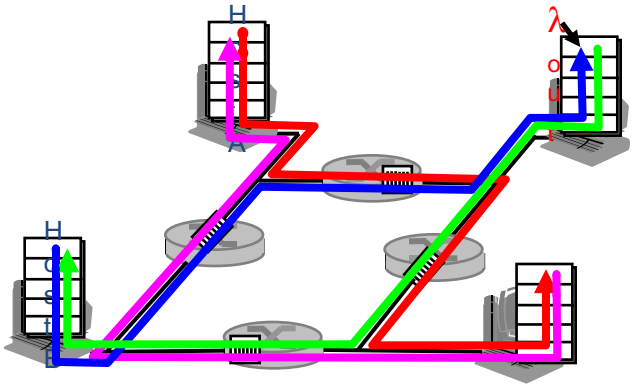
Cause/costi della congestione: 3

- Quattro mittenti
- Percorsi multihop
- timeout/ritrasmissione

D: Che cosa accade quando λ_{in} e λ'_{in} aumentano?







Un altro "costo" della congestione:

- Quando il pacchetto viene scartato, la capacità trasmissiva utilizzata sui collegamenti per instradare il pacchetto risulta sprecata!

Approcci al controllo della congestione

Controllo di congestione end-to-end:

- nessun supporto esplicito dalla rete
- la congestione è dedotta osservando le perdite e i ritardi nei sistemi terminali
- metodo adottato da TCP

Controllo di congestione assistito dalla rete:

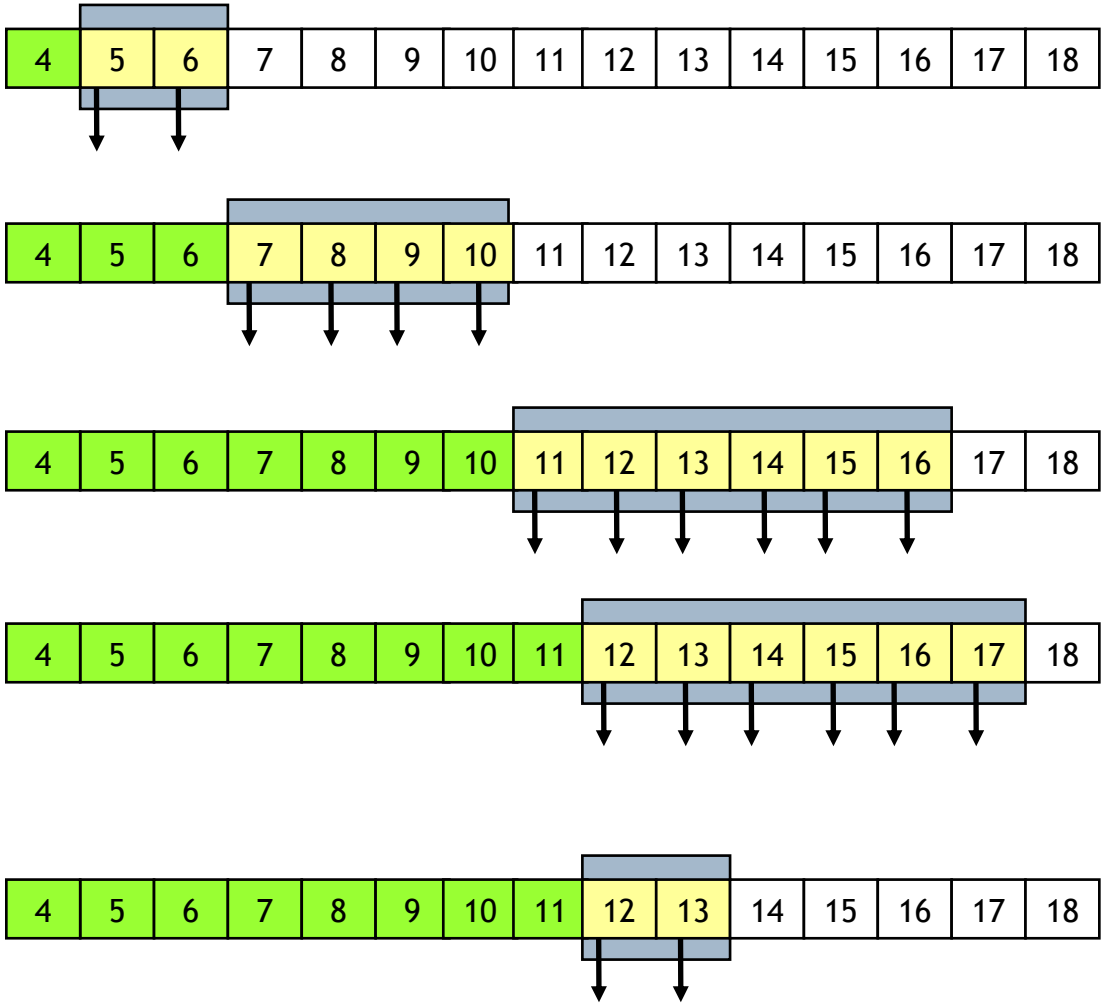
- i router forniscono un feedback ai sistemi terminali
  - un singolo bit per indicare la congestione (SNA, DECbit, TCP/IP ECN, ATM)
  - comunicare in modo esplicito al mittente la frequenza trasmissiva

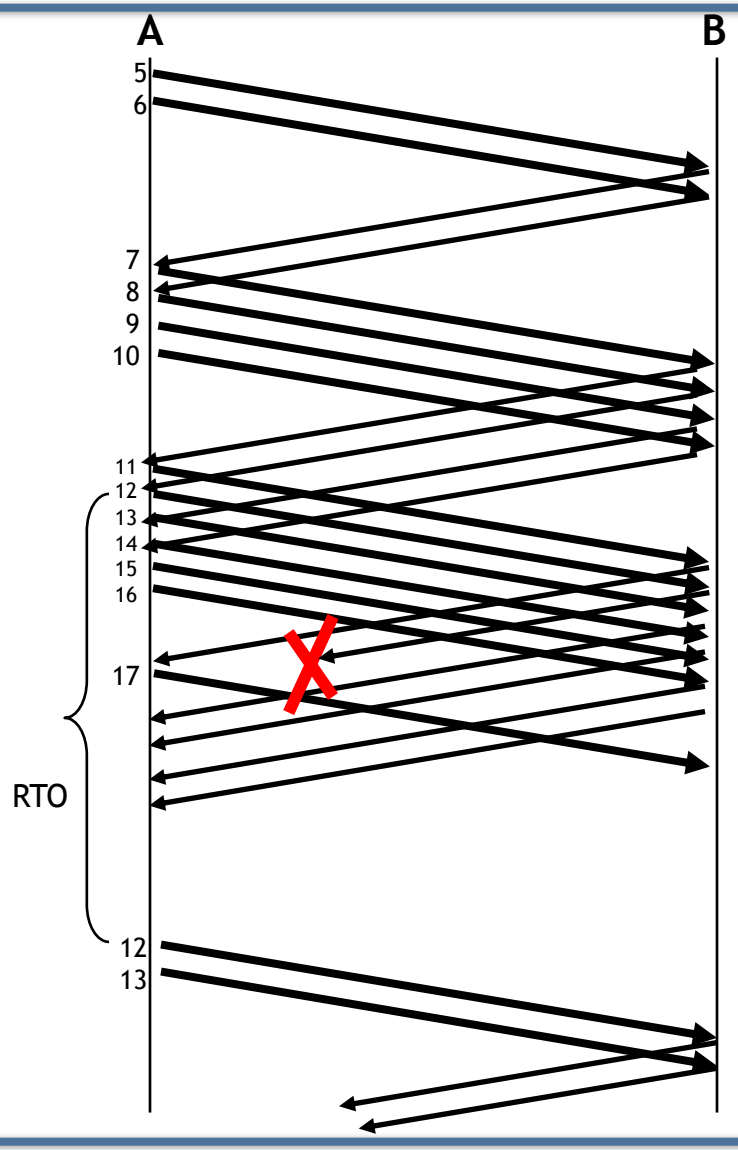
Congestion Window

- In caso di congestione il controllo di flusso rallenta la sorgente
  - se la rete è congestionata, arriveranno meno riscontri e quindi il tasso di immissione diminuisce automaticamente
- La dimensione della finestra è ottimale? No!

- Soluzione per il controllo della congestione → variare dinamicamente la dimensione della finestra di trasmissione
  - la dimensione della finestra non è decisa a priori e statica, ma si adatta alle situazioni (stimate) di carico
  - tale finestra scorrevole e variabile viene denominata "congestion window" (CWND)

Esempio


4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18


A
B
5
6
7
8
9
10
11
12
13
14
15
16
17
RTO
12
13

- Esistono diversi algoritmi che regolano la crescita della finestra (CWND) in assenza di perdite
- I due algoritmi base sono:
  - Slow Start
  - Congestion Avoidance
- Esistono due algoritmi (e una opzione) per aumentare l'efficienza di TCP in caso di perdita (al posto del RTO)
  - Fast Retransmit
  - Fast Recovery
  - SACK (opzione)

- Slow Start
  - per ciascun riscontro ricevuto la CWND aumenta di MSS →quando si riceve un riscontro, si trasmettono due nuovi segmenti (se CWND rimane fissa se ne trasmette 1 solo)
  - l'evoluzione della CWND ha un andamento esponenziale con base RTT
    - al primo RTT la CWND=1, ricevuto il riscontro CWND = 2, ricevuti i due riscontri CWND = 4, ...
- Congestion avoidance
  - per ciascun riscontro ricevuto, la finestra aumenta di MSS/CWND
  - questo implica che ad ogni RTT, in cui si ricevono un numero di riscontri pari alla CWND, la CWND aumenta di un segmento
  - l'evoluzione della CWND ha un andamento lineare nel tempo

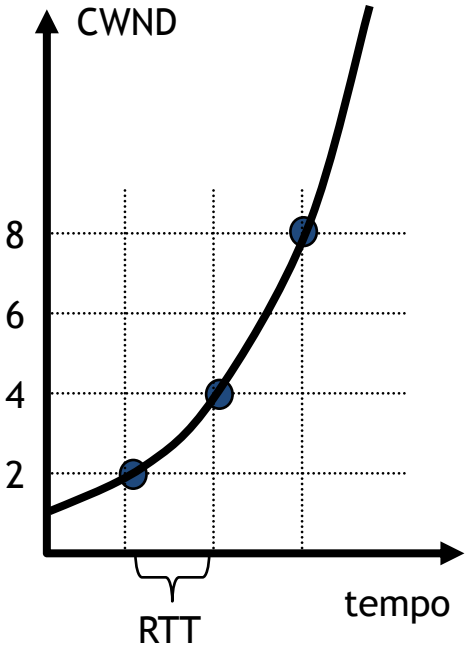
- Durante la fase SS alla ricezione di ogni ACK valido e non duplicato

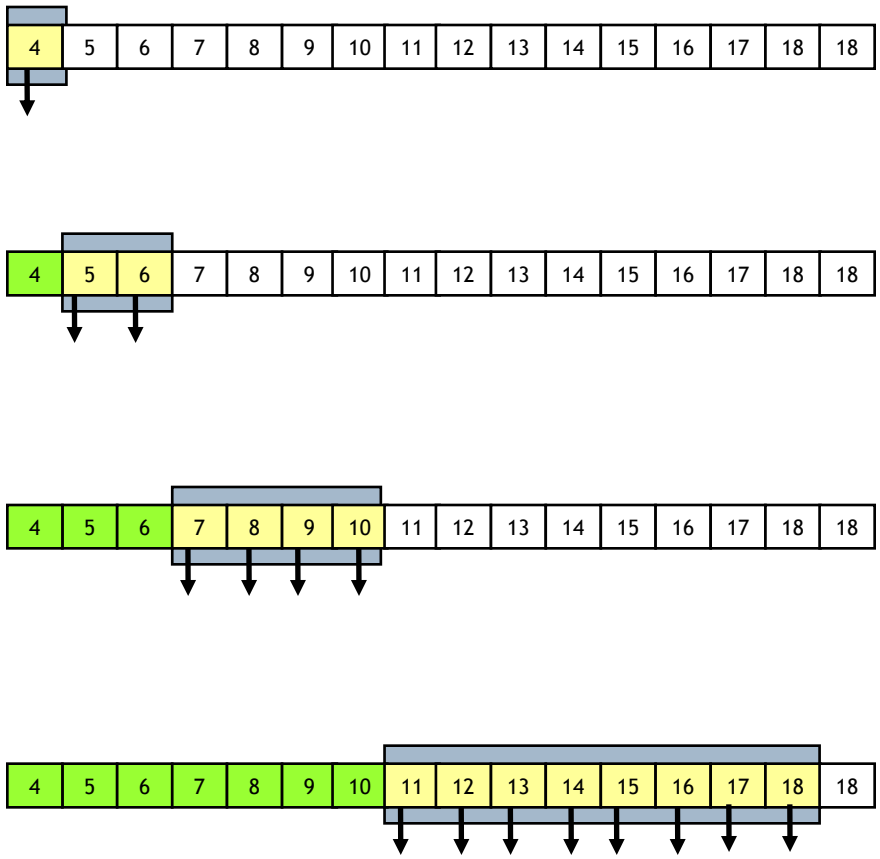
$$CWND = CWND + MSS$$

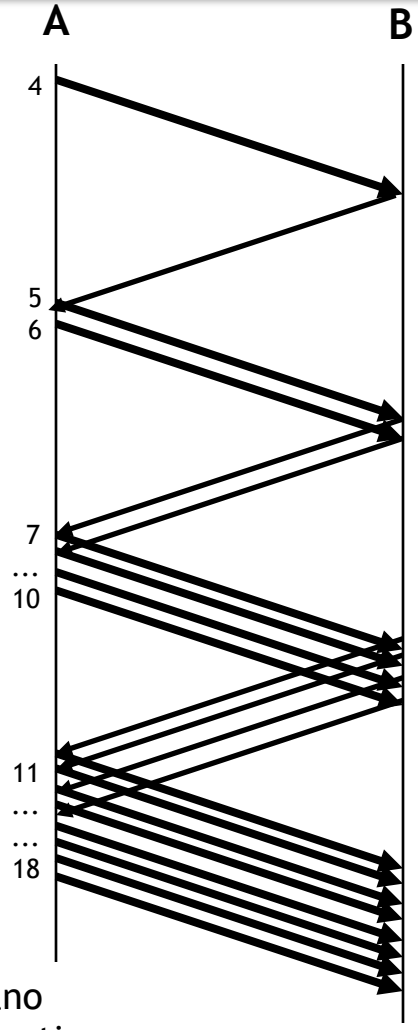
- Durante la fase CA alla ricezione di ogni ACK valido e non duplicato

$$CWND = CWND + MSS*(MSS/CWND)$$

Esempio: Slow Start

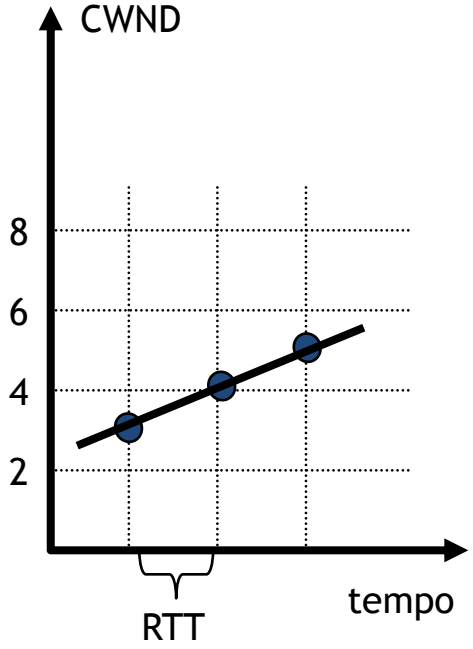


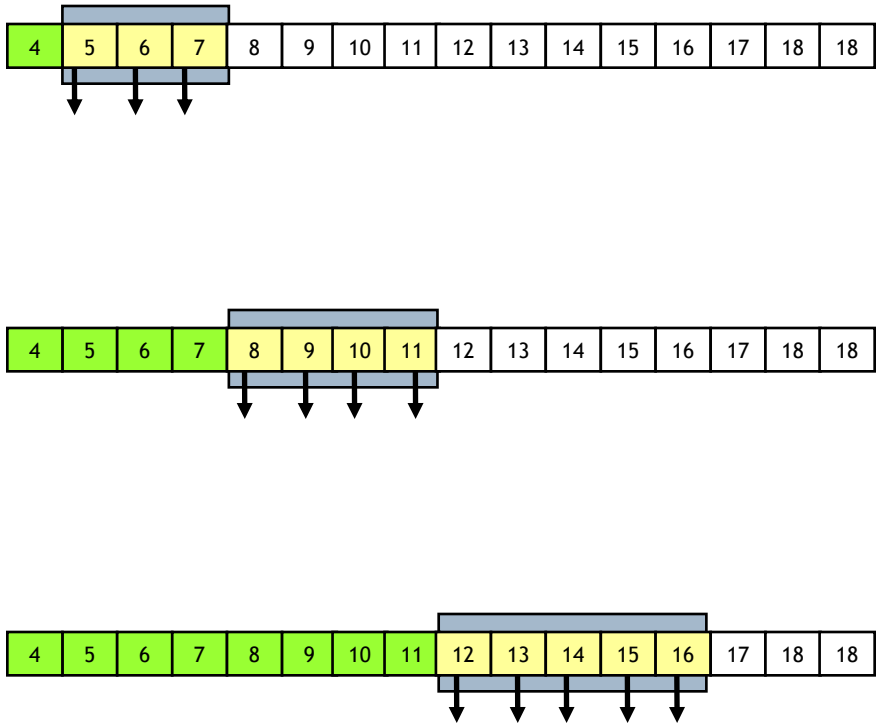


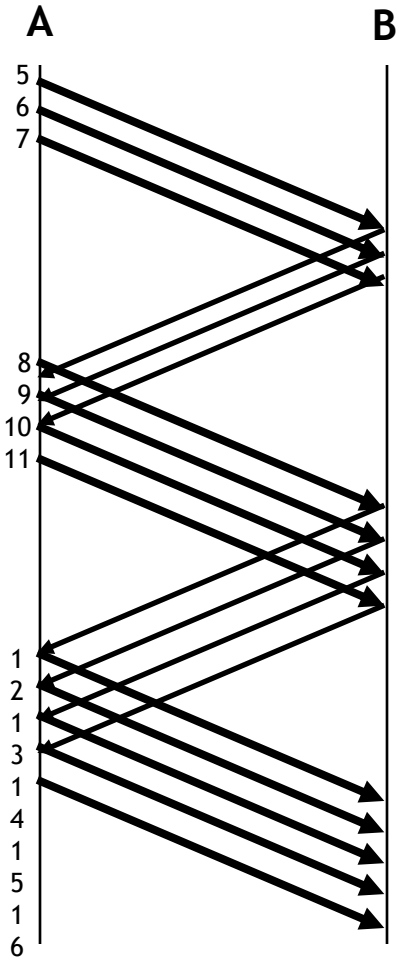


NOTA: per semplificare la rappresentazione grafica, si assume che i segmenti vengano generati e trasmessi tutti nello stesso istante e i corrispettivi riscontri vengano ricevuti di conseguenza tutti insieme dopo un tempo pari a RTT (supposto costante)

Esempio: Congestion Avoidance







Evoluzione della CWND

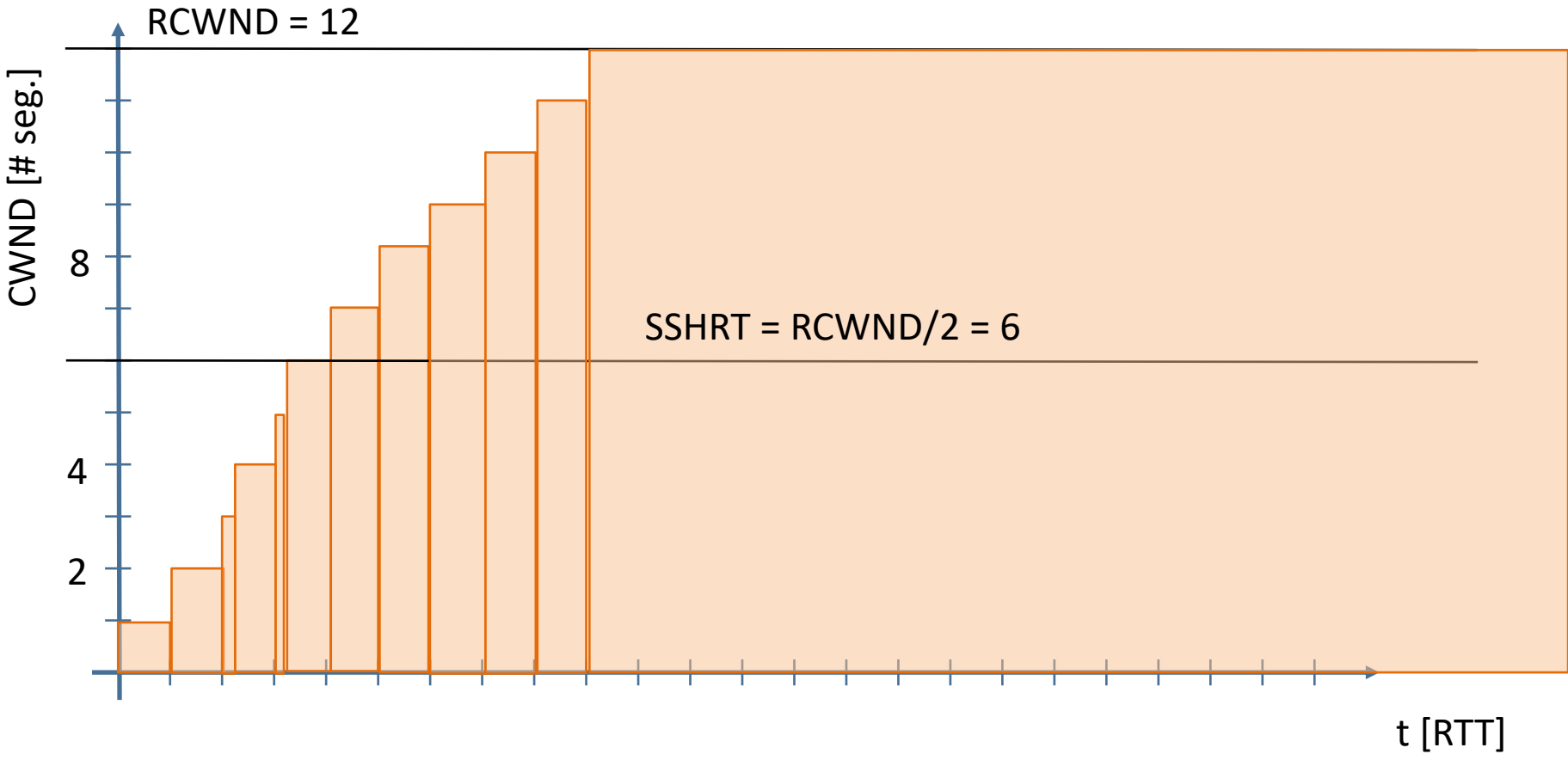
- Congestion Window (CWND)
  - dimensione della finestra (espressa in byte o in numero di segmenti) di trasmissione
- Receiver Window (RECWND)
  - dimensione della finestra di ricezione (espressa in byte o in numero di segmenti) annunciata dalla destinazione; è il limite massimo che la CWND può assumere
- Slow Start Threshold (SSTHRESH)
  - dimensione della finestra (espressa in byte o in numero di segmenti) raggiunta la quale, invece di seguire l'algoritmo di Slow Start, si segue l'algoritmo di Congestion Avoidance
- RTT: tempo trascorso tra l'invio di un segmento e la ricezione del riscontro; in condizioni di stabilità della rete e del carico, RTT rimane pressoché costante
- RTO: tempo che la sorgente aspetta prima di ritrasmettere un segmento non riscontrato

Evoluzione della CWND

- L'algoritmo che regola la dimensione della CWND è il seguente:
  - all'inizio della trasmissione si pone
    - CWND = 1 segmento (ovvero un numero di byte pari a MSS)
    - SSTHRESH = RCVWND oppure SSTHRESH = RCVWND / 2 (dipende dalle implementazioni)
  - la CWND evolve secondo l'algoritmo di Slow Start fino al raggiungimento della SSTHRESH
  - raggiunta la soglia SSTHRESH, la dimensione di CWND è regolata dall'algoritmo di Congestion Avoidance
  - la finestra cresce fino al raggiungimento di RCVWND

Evoluzione CWND senza perdite

- Slow Start + Congestion Avoidance + RCWND



Throughput

- Abbiamo visto che il throughput di un protocollo a finestra (fissa) è dato dalla formula

$$\text{Thr [bit/s]} = \text{DF [byte]} * 8 / \text{T [s]}$$

- dove T è la durata della trasmissione e DF la dimensione della finestra

- E per un protocollo a finestra variabile?
- Dipenderà dalla finestra istante per istante e quindi nel lungo periodo dall'integrale della finestra nel tempo

$$\text{Thr [bit/s]} = 1/T \int_T DF(t)\, dt$$

Throughput

- Per avere buone prestazioni è fondamentale che TCP mantenga la giusta dimensione della finestra
- Se RCVWND è troppo grande i pacchetti si accumulano nei buffer della rete aumentando RTT ... e quindi la durata della connessione per trasferire una data quantità di dati
- Questa è una forma di congestion control
- Funziona molto bene quando il collo di bottiglia è il link al trasmettitore: i pacchetti si accumulano nel SO dell'host che rappresenta un buffer molto grande

- Cosa accade in caso di congestione e perdite?

Recupero delle perdite

- TCP recupera le perdite con due meccanismi
- Scadenza RTO
- Ricezione di 3 ACK duplicati (4 ACK uguali) --> Fast Retransmit
- In ogni caso TCP ritrasmette il pacchetto perso (il primo della finestra per costruzione) e gli ACK cumulativi garantiscono che il meccanismo sia di tipo selective-repeat.
- Se scade RTO TCP “ricomincia da capo” (ma non si comporta mai come Go Back N, perché RCWND >= TXWND
- Se si usa Fast Retransmit TCP lo abbina a un meccanismo che si chiama Fast Recovery il cui scopo è dimezzare la finestra al momento della perdita e entrare in congestion avoidance
- Vediamo i diversi casi in dettaglio ...

Recupero con RTO

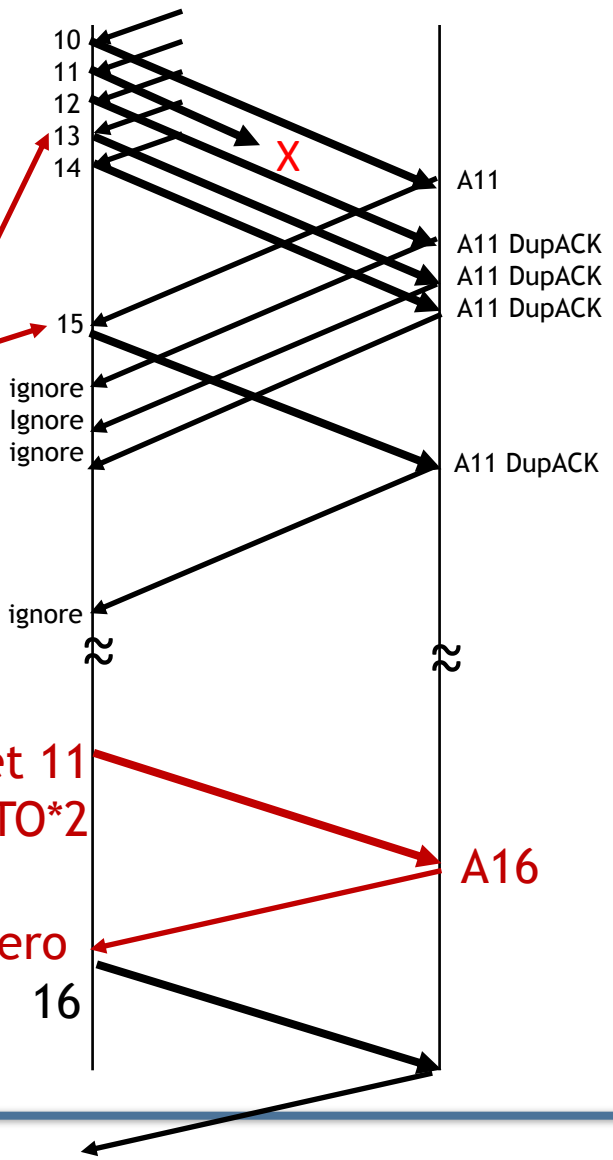
Alla scadenza di RTO TCP:

1. RTO = RTO*2 ; SSHTHR = CNGWND/2
2. CNGWND = MSS (1 segmento)
3. Ritrasmette il primo segmento della finestra
4. Attende di ricevere l'ACK
5. Quando (e se) riceve ACK: WLow = (ACK Number)
6. Se (ACK Number) > WUp
   ricomincia funzionamento normale
   Altrimenti
   ritrasmette un segmento ricominciando dal punto 3
7. Se scade RTO ricomincia dal punto 1. per 10 volte; e per ulteriori 6 volte senza incrementare RTO ... poi si arrende

Recupero con RTO

Ragioniamo a segmenti
WLow = 10; CNGWND = 5; WUp = 14
Fase Congestion Avoidance
Segmento perso No. 11

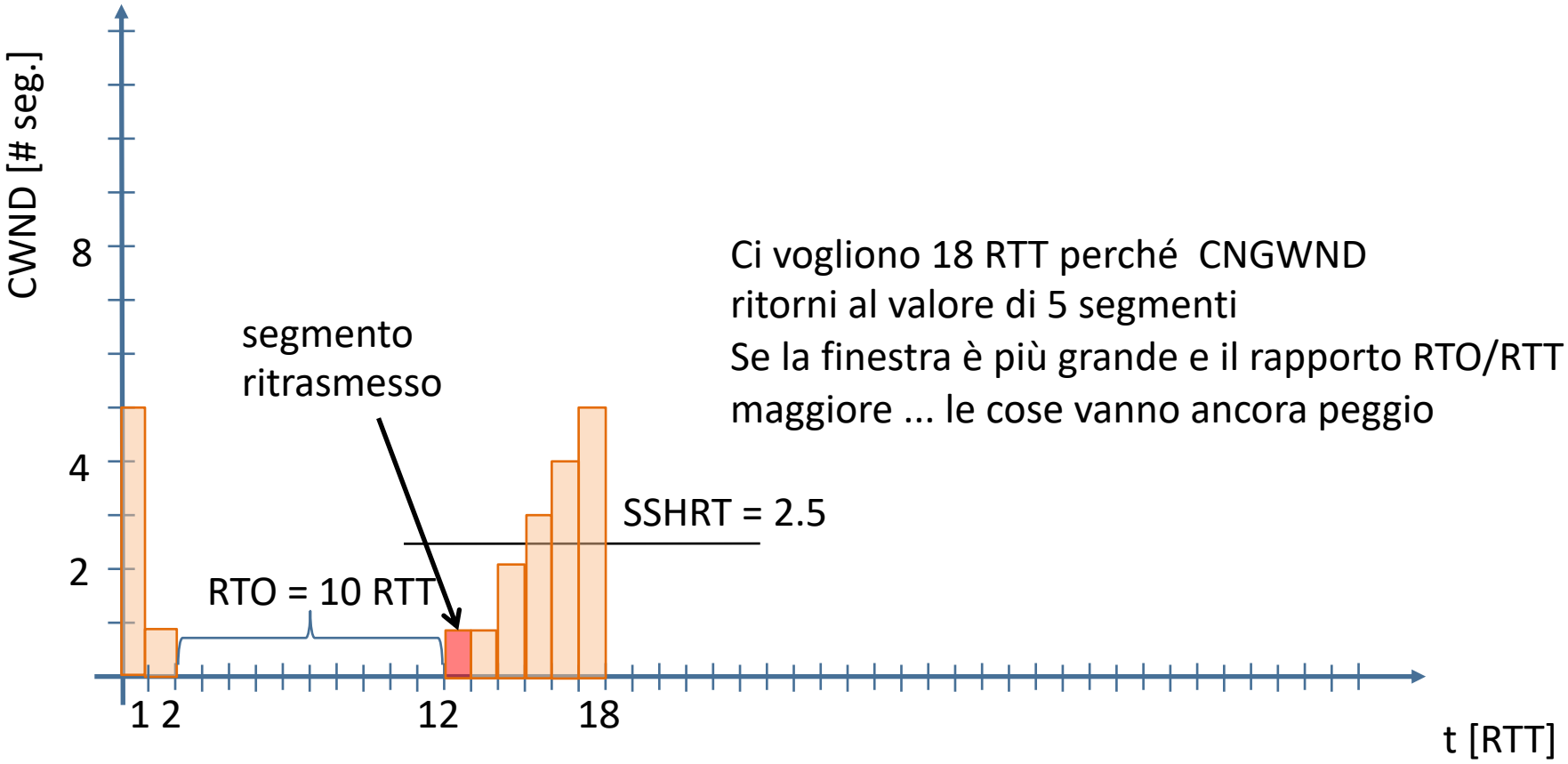
La finestra NON cresce perché
non sono stati ricevuti CNGWND ACK



WLow = 16; CNGWND = 1; WUp = 16
SSTHR = 5*MSS/2
Riparte in slow start

Dinamica della finestra: RTT = 20ms; RTO = 200ms; RCWND = 40 segmenti (~64K)

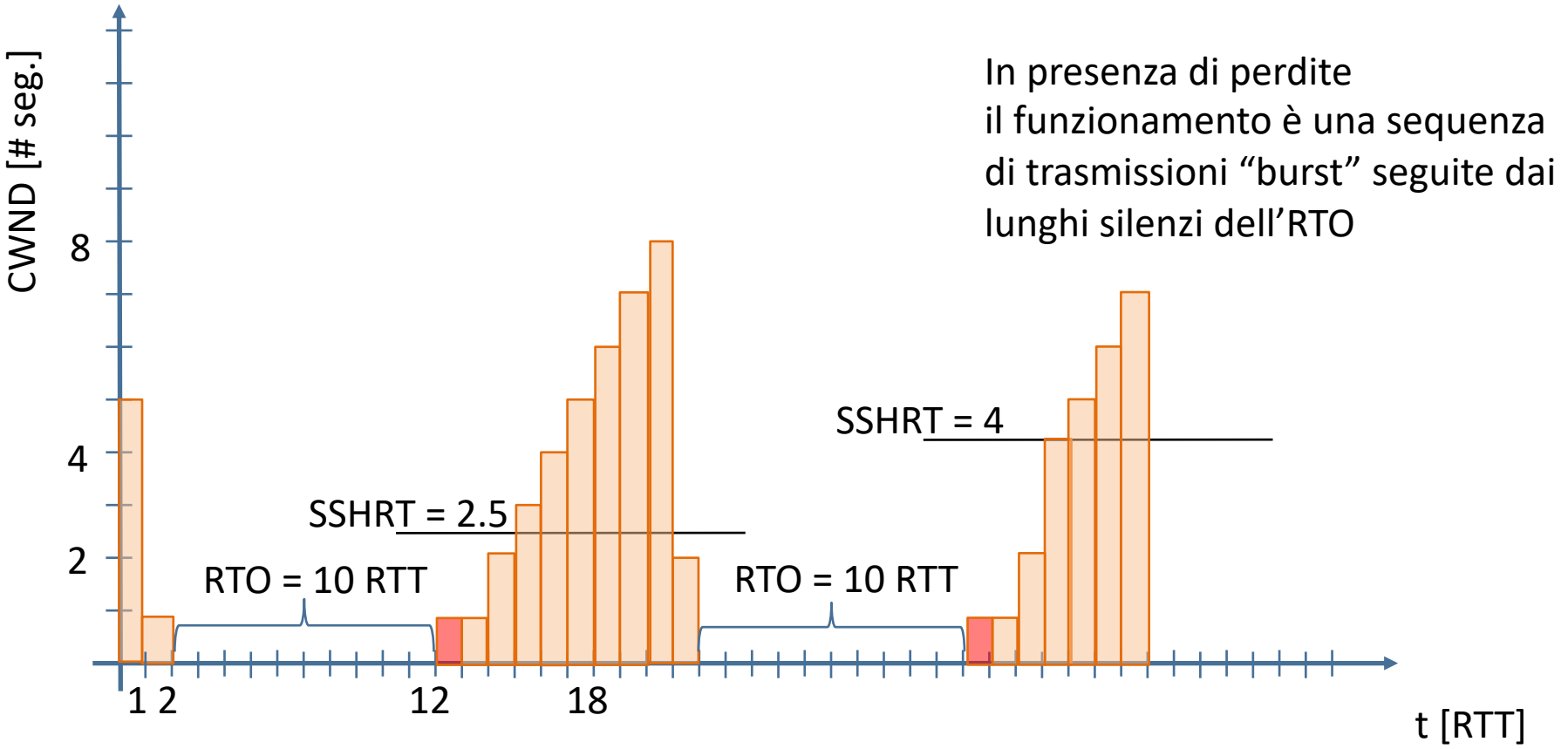
La finestra in questa slide rappresenta il numero effettivo di segmenti inviati nel RTT



Ci vogliono 18 RTT perché CNGWND ritorni al valore di 5 segmenti
Se la finestra è più grande e il rapporto RTO/RTT maggiore ... le cose vanno ancora peggio

Recupero con RTO perdite multiple

Dinamica della finestra: RTT = 20ms; RTO = 200ms; RCWND = 40 segmenti (~64K)
Seconda perdita quanto CNGWND ha raggiunto 8 segmenti
La finestra in questa slide rappresenta il numero effettivo di segmenti inviati nel RTT



In presenza di perdite
il funzionamento è una sequenza
di trasmissioni “burst” seguite dai
lunghi silenzi dell’RTO

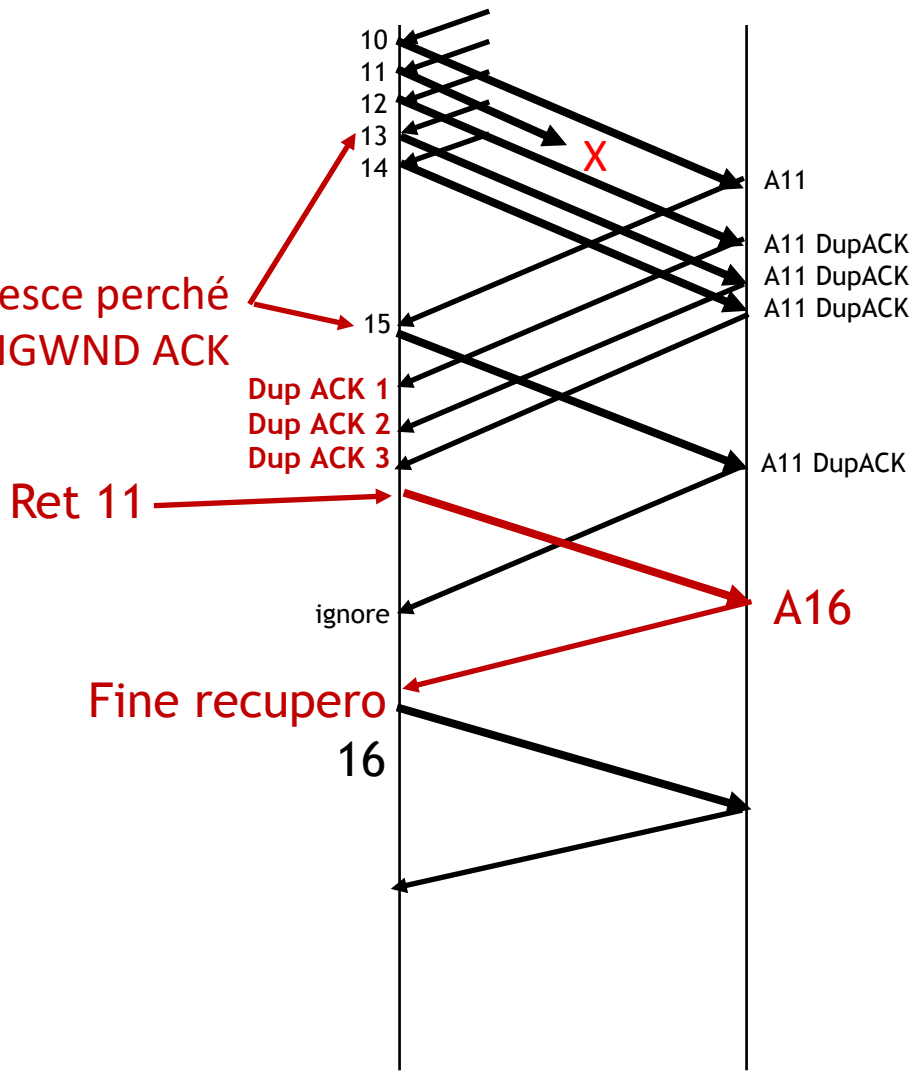
Quando vengono ricevuti 3 ACK duplicati:

1. SSHTHR = CNGWND/2; WUpLoss=WUp
2. CNGWND = MSS (1 segmento)
3. Ritrasmette il primo segmento della finestra
4. Attende e scarta (ignora) tutti i DupACK successivi
5. Quando (e se) riceve ACK non duplicato: WLow = (ACK Number); WUp = WLow + CNGWND
6. Se (ACK Number) > WUpLoss
   ricomincia funzionamento normale (in Slow Start)
   Altrimenti
   ritrasmette un segmento ricominciando dal punto 3
7. Se scade RTO (es. perdita segmento ritrasmesso) si comporta come con recupero con RTO

Recupero con Fast Retransmit

Ragioniamo a segmenti
WLow = 10; CNGWND = 5; WUp = 14
Fase Congestion Avoidance
Segmento perso No. 11

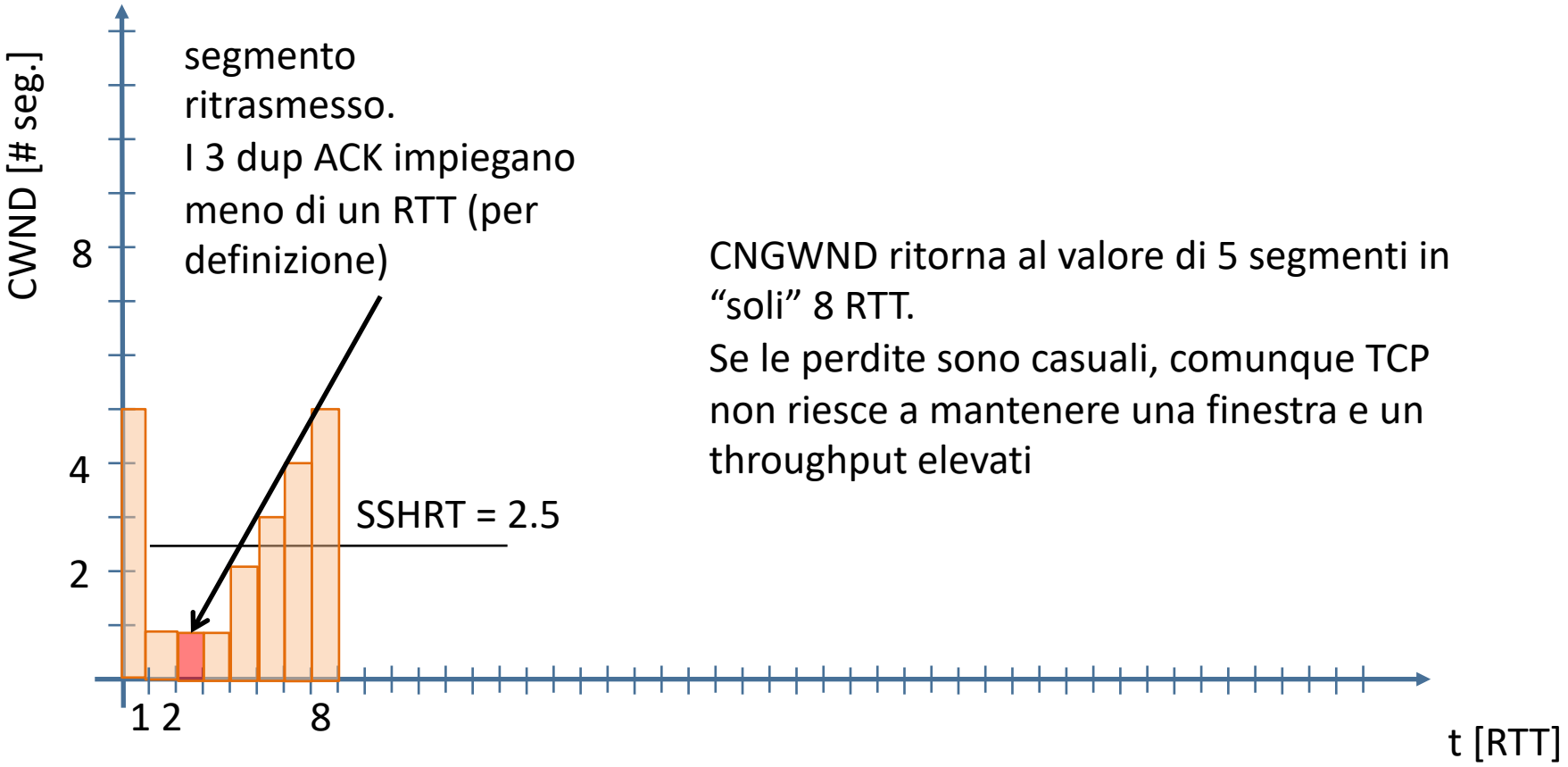
La finestra NON cresce perché
non sono stati ricevuti CNGWND ACK



WLow = 16; CNGWND = 1; WUp = 16
SSTHR = 5*MSS/2
Riparte in slow start

Dinamica della finestra: RTT = 20ms; RTO = 200ms; RCWND = 40 segmenti (~64K)

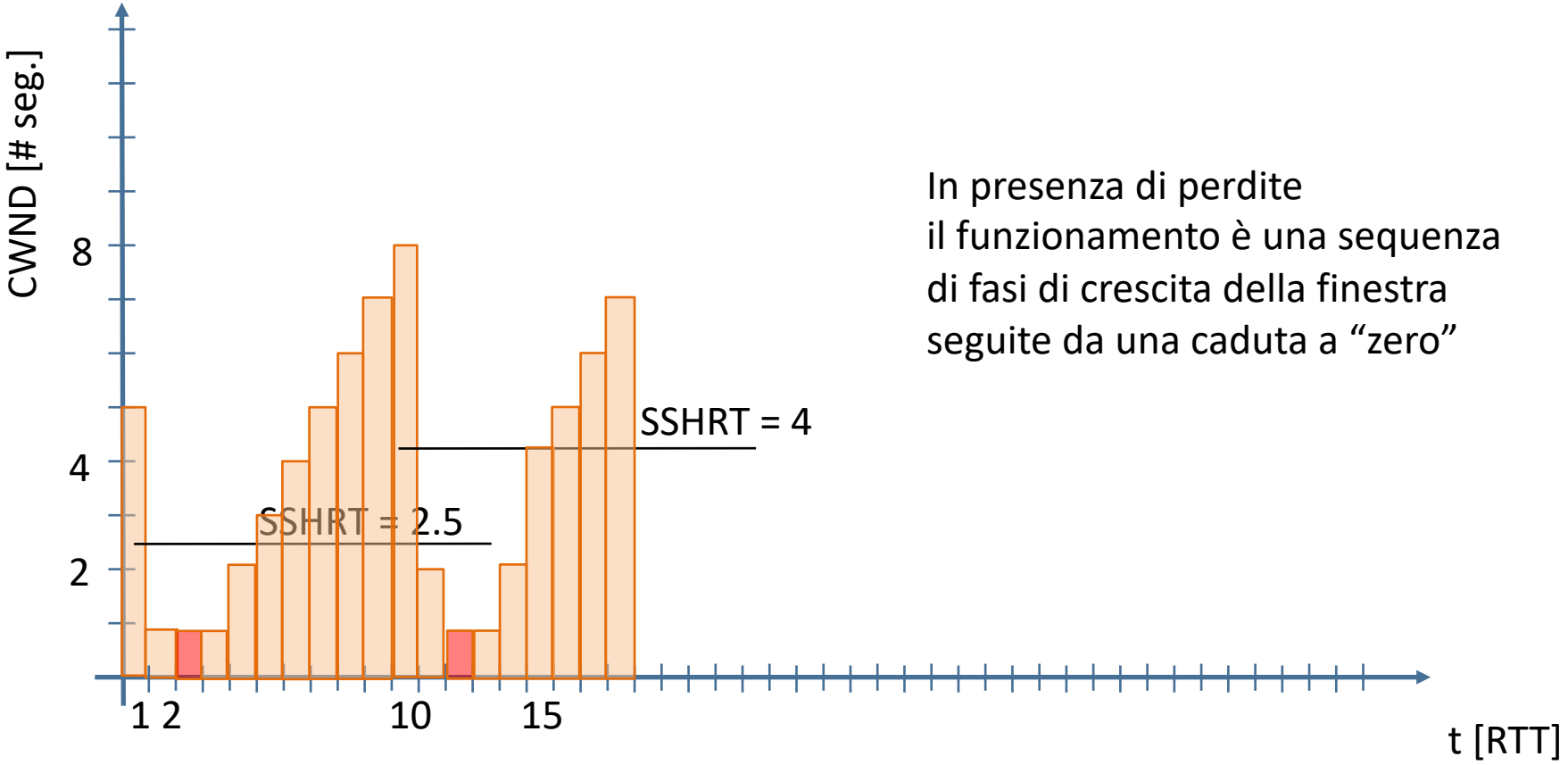
La finestra in questa slide rappresenta il numero effettivo di segmenti inviati nel RTT



segmento ritrasmesso.
I 3 dup ACK impiegano meno di un RTT (per definizione)

CNGWND ritorna al valore di 5 segmenti in "soli" 8 RTT.
Se le perdite sono casuali, comunque TCP non riesce a mantenere una finestra e un throughput elevati

Dinamica della finestra: RTT = 20ms; RTO = 200ms; RCWND = 40 segmenti (~64K)
Seconda perdita quanto CNGWND ha raggiunto 8 segmenti
La finestra in questa slide rappresenta il numero effettivo di segmenti inviati nel RTT

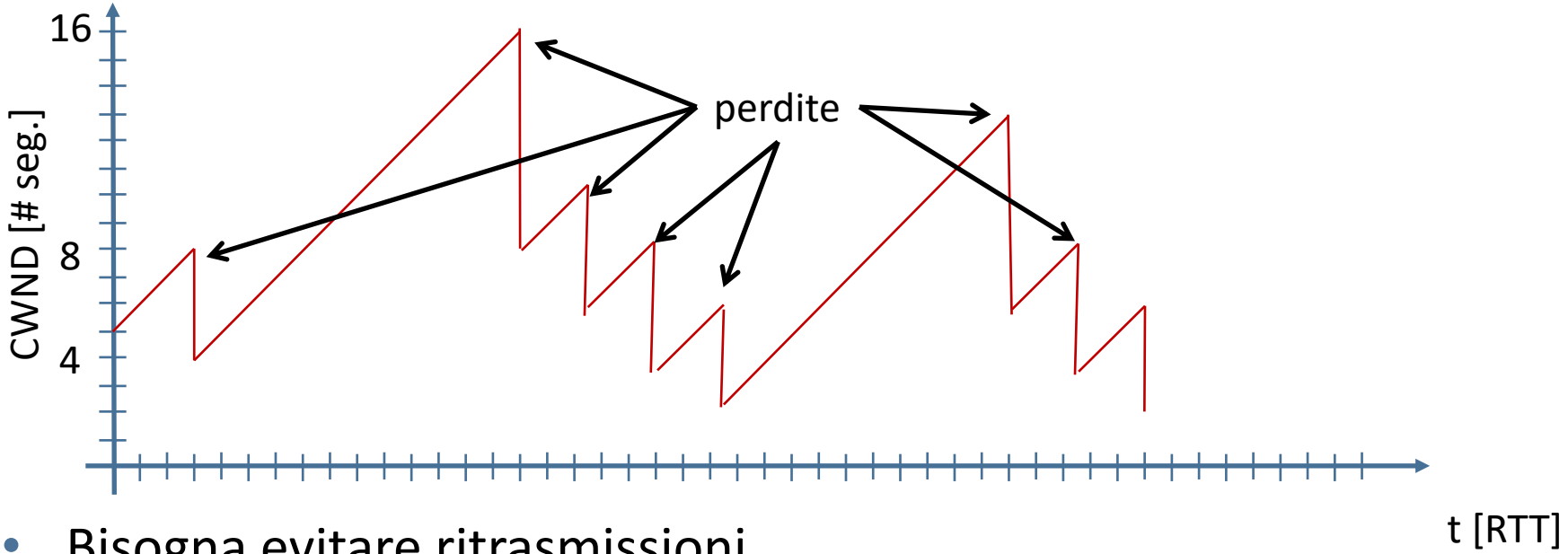


In presenza di perdite
il funzionamento è una sequenza
di fasi di crescita della finestra
seguite da una caduta a “zero”

Recupero con Fast Retransmit / RTO

- Usando i semplici meccanismi di RTO e Fast Retransmit TCP implementa ritrasmissione selettiva
- Può recuperare un segmento perso ciascun RTT
- RTO è molto inefficiente
- Fast Retransmit migliora efficienza, ma il funzionamento continua ad essere “a singhiozzo”
- Idea: anziché “ricominciare da capo” si può entrare direttamente in congestion avoidance mettendo CNGWND = SSTHR
- Però senza altri accorgimenti si rischia di ritrasmettere segmenti già ricevuti

- L'idea di base è quella di ottenere una "oscillazione" di CNGWND "a dente di sega" dimezzando la finestra in caso di perdite e incrementandola linearmente quando non ci sono perdite



- Bisogna evitare ritrasmissioni
- Bisogna evitare timeout
- Sarebbe meglio evitare l'attesa di un RTT se si continuano a ricevere ACK duplicati

Quando vengono ricevuti 3 ACK duplicati:

1. SSHTHR = CNGWND/2; WUpLoss
2. Ritrasmette il primo segmento della finestra
3. CNGWND = SSHTHR + 3MSS (3 Dup ACK ricevuti); se la finestra lo consente invia **nuovi segmenti**
4. Per ogni nuovo Dup ACK CNGWND = CNGWND + MSS e invia un nuovo segmento se la finestra lo consente
5. Quando (e se) riceve ACK non duplicato: WLow = (ACK Number)

1. Se (ACK Number) > WUpLoss
   CNGWND = SSHTHR
   ricomincia funzionamento normale in congestion avoidance
   Altrimenti
   ritrasmette un segmento rimane in Fast Recovery
   riprendendo dal punto 4, a questo punto gli ACK duplicati si riferiscono al secondo segmento ritrasmesso
2. Se scade RTO (es. perdita segmento ritrasmesso) si comporta come con recupero con RTO

Recupero con Fast Recovery

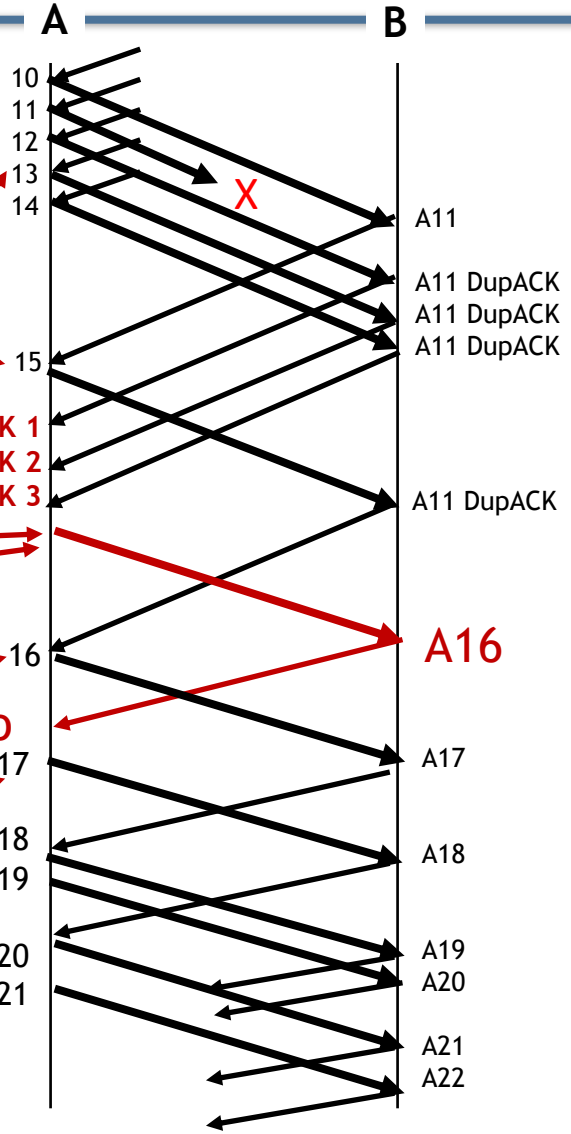
Ragioniamo a segmenti
WLow = 10; CNGWND = 5; WUp = 14
Fase Congestion Avoidance
Segmento perso No. 11

La finestra NON cresce perché
non sono stati ricevuti CNGWND ACK

WLow = 10; WUp = 16
SSTHR = 5*MSS/2
CNGWND = SSHTR+3
Non può trasmettere
nuovi segmenti

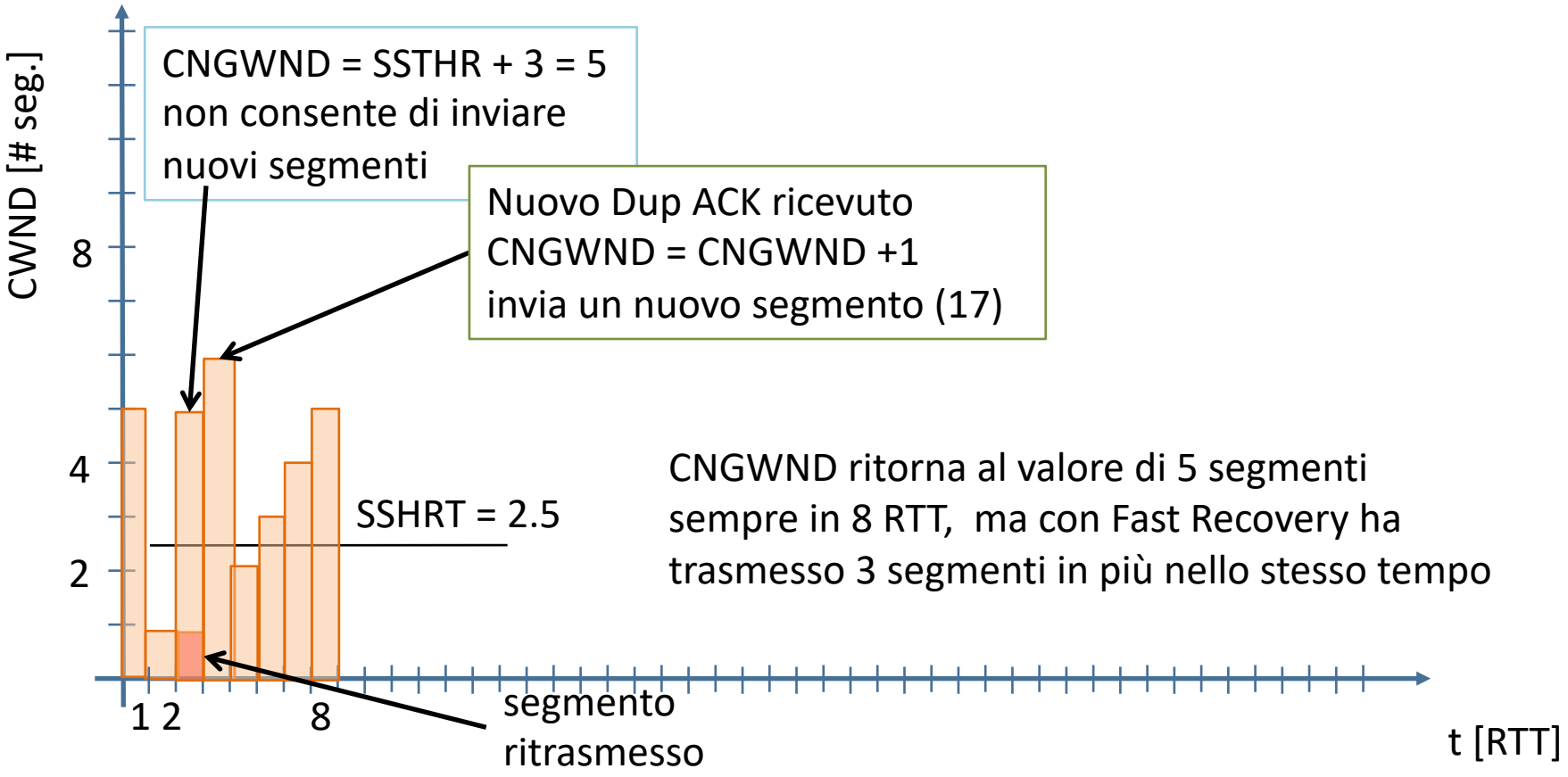
CNGWND = CNGWND + 1
trasmette nuovo segmento

CNGWND = SSTHR
trasmette nuovo segmento
CNGWND = CNGWND + MSS/2 = 3
trasmette 2 nuovi segmenti



Dinamica della finestra: RTT = 20ms; RTO = 200ms; RCWND = 40 segmenti (~64K)

La finestra in questa slide rappresenta il numero effettivo di segmenti inviati nel RTT



CNGWND ritorna al valore di 5 segmenti sempre in 8 RTT, ma con Fast Recovery ha trasmesso 3 segmenti in più nello stesso tempo

Recupero perdite: riassunto

- RTO implementa ritrasmissioni selettive, funziona sempre, ma spreca moltissime risorse (tempo)
- Fast Retransmit abbatte il tempo di recupero, consente di recuperare un segmento per RTT (anziché RTO), ma è ancora piuttosto inefficiente
- Fast Recovery è molto complesso e ci sono casi in cui ... si inceppa → RTO
- Quando funziona correttamente Fast Recovery sostiene la trasmissione di nuovi pacchetti mentre recupera quelli persi.

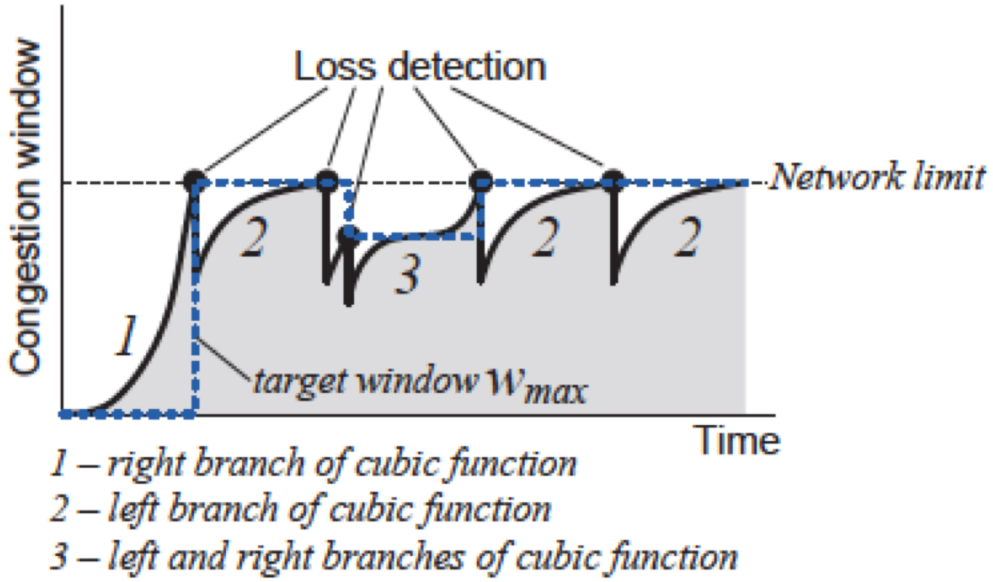
Crescita finestra

- Lo standard prevede che la finestra in CA cresca linearmente con RTT
- Diverse implementazioni non rispettano questa indicazione
- Linux → TCP Cubic
- Windows → TCP Compound
- Il risultato è ... un po' di caos in rete (ancora?!?!)

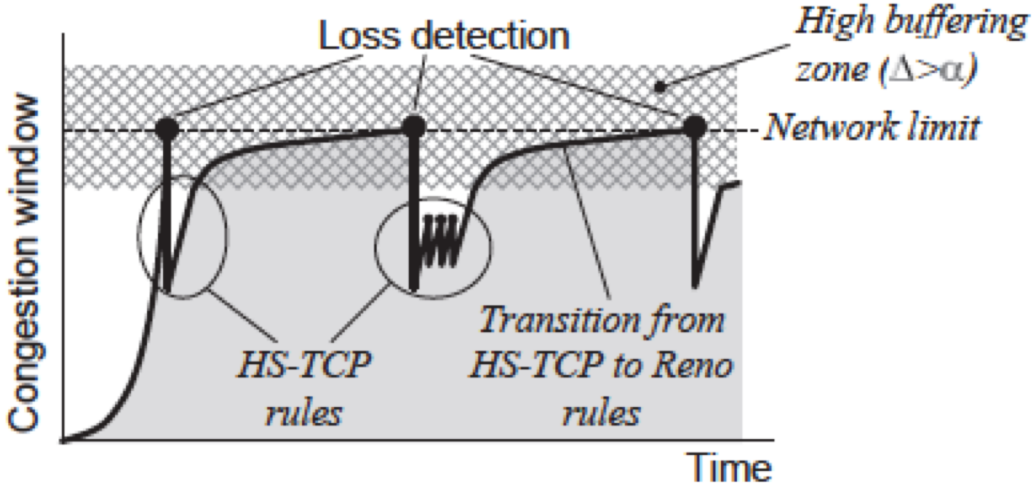
- TCP dovrebbe controllare la congestione ed essere anche equo
- Abbiamo già visto che in realtà dipende da RTT
- Con diverse "logiche di crescita della finestra è ancora peggio

Implementazioni non standard

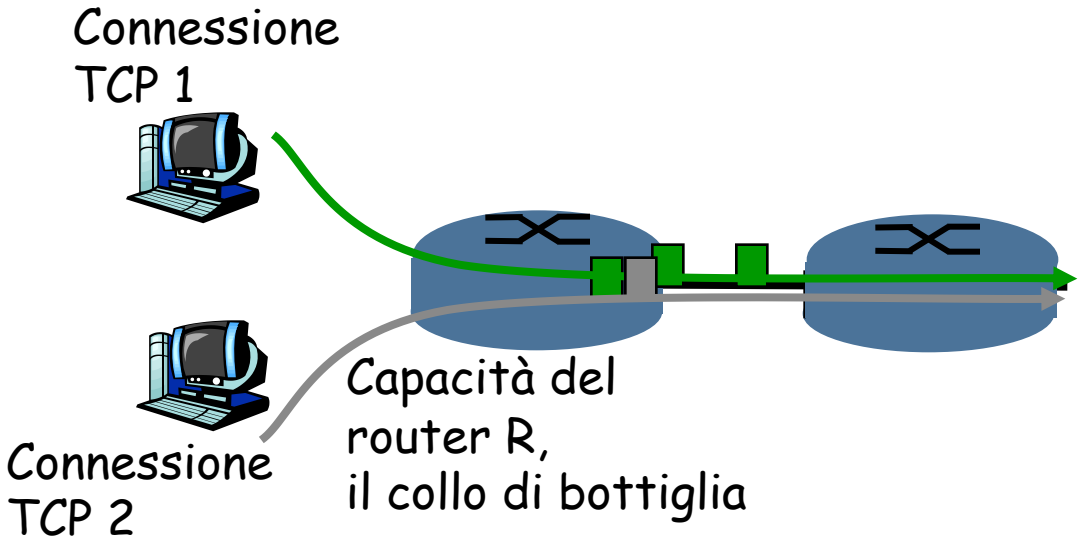
- TCP CUBIC (Linux)



- TCP Compound (MS Windows)



Equità: se K sessioni TCP condividono lo stesso collegamento con ampiezza di banda R, che è un collo di bottiglia per il sistema, ogni sessione dovrà avere una frequenza trasmissiva media pari a R/K.



TCP è equo?

Due connessioni in concorrenza tra loro:

- L'incremento additivo determina una pendenza pari a 1, all'aumentare del throughout
- Il decremento moltiplicativo riduce il throughput in modo proporzionale