Marco Listanti

Protocolli di trasporto

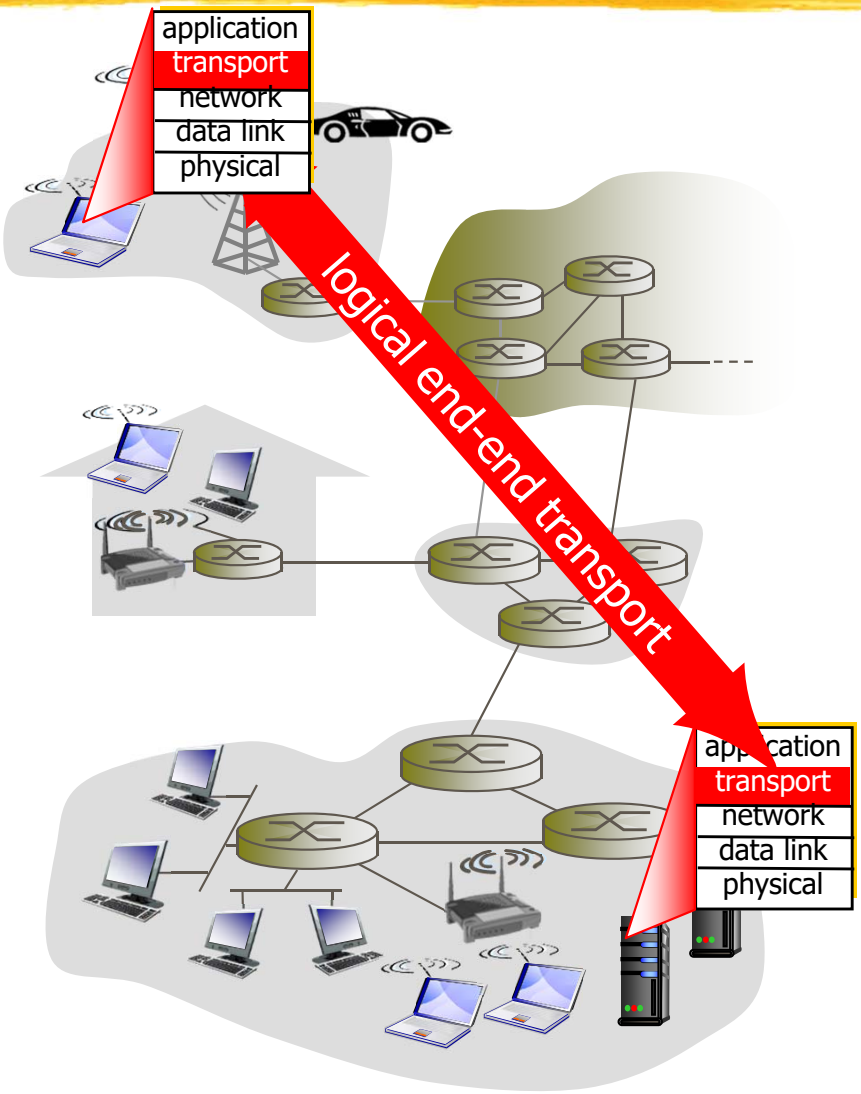
Testo di riferimento:

J. Kurose, K. Ross: "*Reti di calcolatori e Internet* (4a edizione)". Pearson Addison Wesley

INFOCOM DIET DEPT Networking Group

Servizio e protocolli di trasporto

- Fornisce un collegamento logico tra processi in host remoti
- Le funzioni di trasporto sono eseguite nei sistemi terminali
  - Lato emittente: divide i messaggi (A-PDU) in segmenti (T-PDU) e li inoltra allo strato di rete
  - Lato ricevente: esegue la ricostruzione dei messaggi e li inoltra allo stato di applicazione
- Protocolli di trasporto in Internet: TCP and UDP



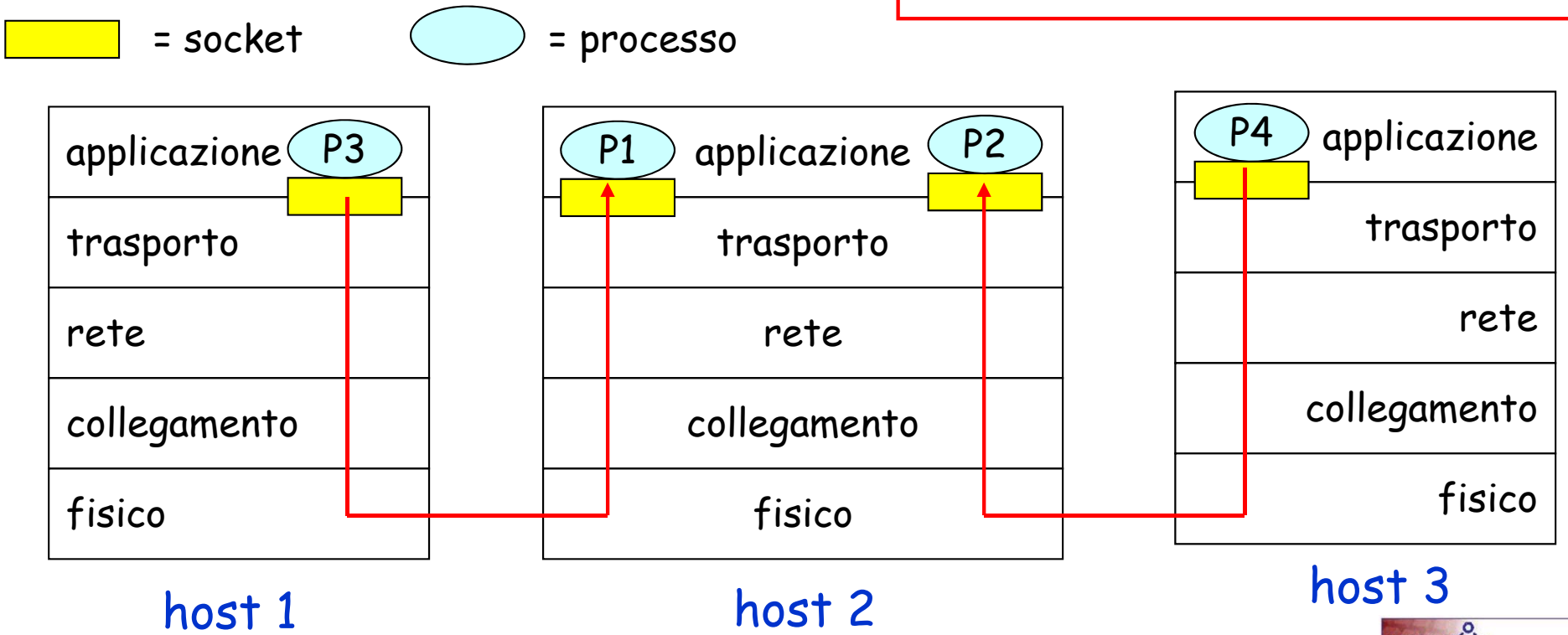
Multiplexing/demultiplexing

Multiplexing nell'host mittente:

Raccolta dei dati da varie socket, incapsulamento con l'intestazione (utilizzata poi per il demultiplexing)

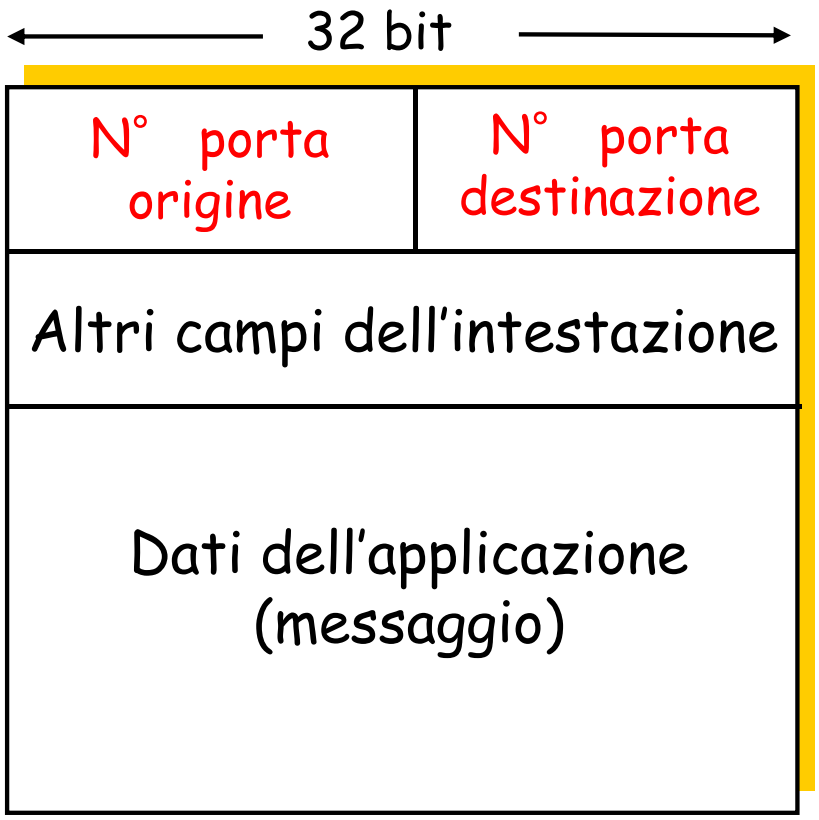
Demultiplexing nell'host ricevente:

Consegna dei segmenti ricevuti alla socket appropriata



Demultiplexing

- **L'host riceve i pacchetti IP**
  - ogni pacchetto ha un indirizzo IP di origine e un indirizzo IP di destinazione
  - ogni pacchetto trasporta 1 segmento a livello di trasporto
  - ogni segmento ha un numero di porta di origine e un numero di porta di destinazione
- **L'host usa gli indirizzi IP e i numeri di porta per inviare il segmento alla socket appropriata**



Struttura del segmento TCP/UDP

Demultiplexing senza connessione

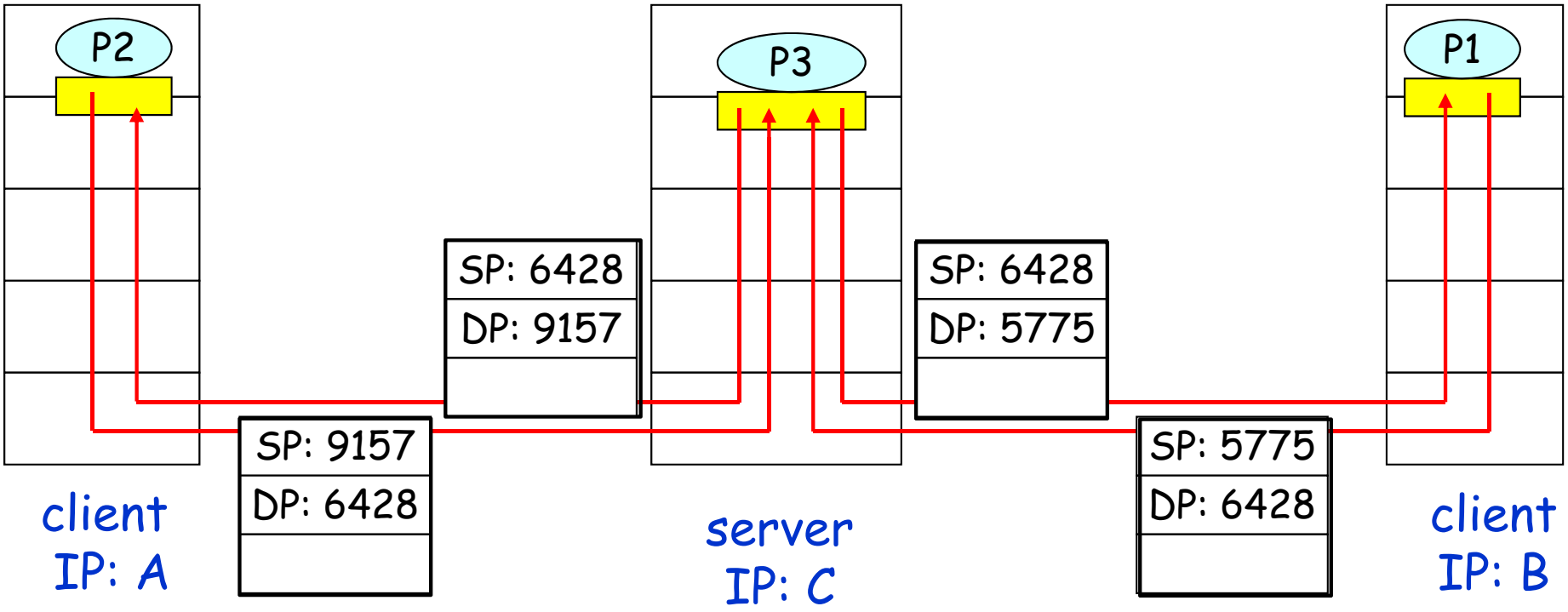
- Un Host crea le socket con i numeri di porta
- Una socket UDP è identificata da 2 parametri

(indirizzo IP di destinazione, numero di porta di destinazione)

- Quando l'host riceve il segmento UDP
  - controlla il numero della porta di destinazione nel segmento
  - invia il segmento UDP alla socket con quel numero di porta
- Pacchetti IP con la stesso numero di porta di destinazione, ma con indirizzi IP di sorgente, e/o numeri di porta di sorgente diversi vengono inviati alla stessa socket

Demultiplexing senza connesssione

- Il server C crea per il processo P3 una socket con il numero di porta 6428

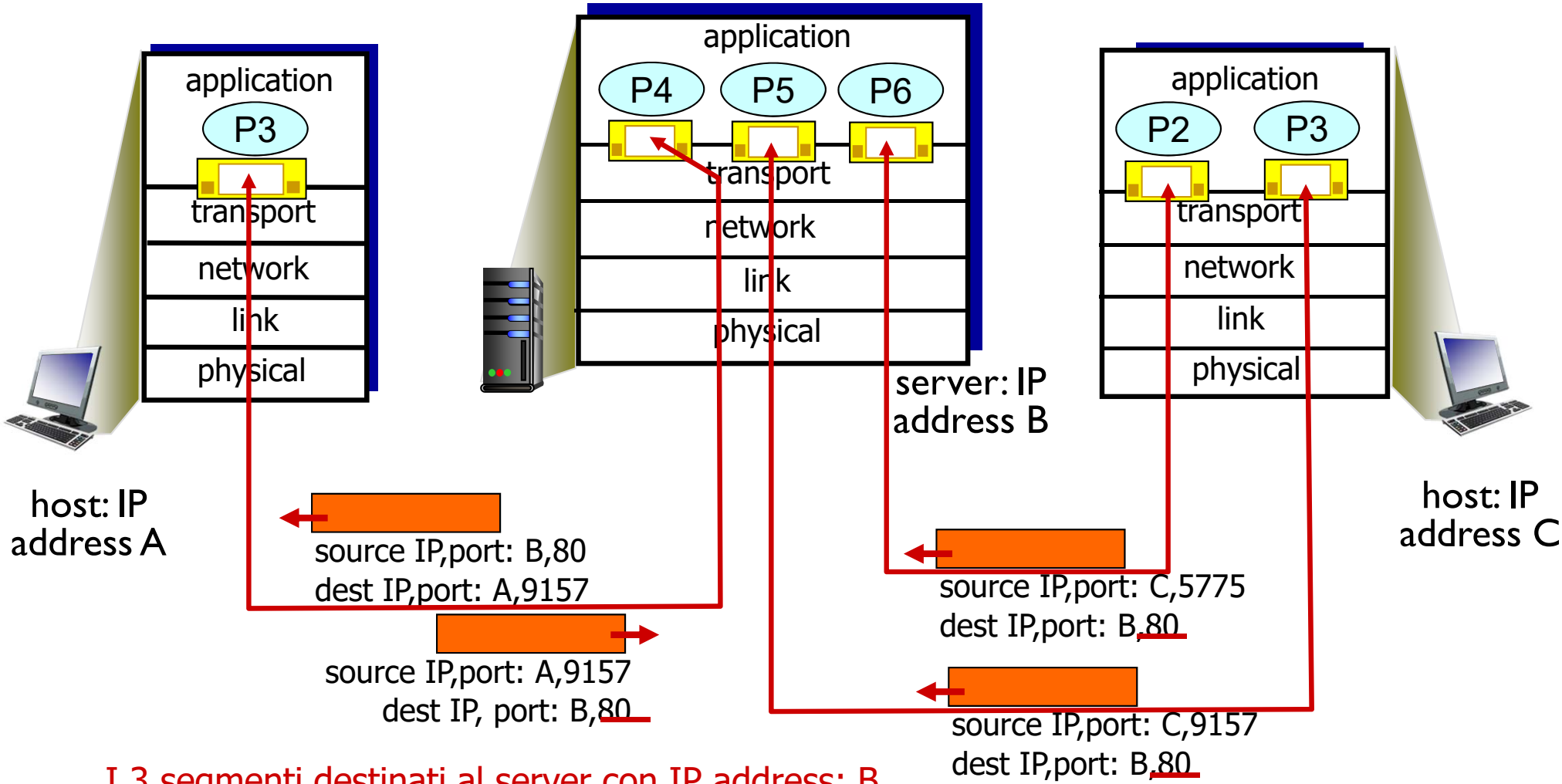


SP fornisce "l'indirizzo di ritorno"

Demultiplexing orientato alla connessione

- **Una socket TCP è identificata da 4 parametri**
  - indirizzo IP di origine
  - numero di porta di origine
  - indirizzo IP di destinazione
  - numero di porta di destinazione
- **L'host ricevente usa i quattro parametri per inviare i segmenti alla socket appropriata**
- **Un host server può supportare più socket TCP contemporaneamente**
  - ogni socket è identificata dai suoi 4 parametri
- **I server web hanno socket differenti per ogni connessione**
  - HTTP non-persistente avrà una socket differente per ogni richiesta

Demultiplexing Connection-oriented



I 3 segmenti destinati al server con IP address: B,
e dest port: 80 sono inoltrati a socket diverse

Indirizzamento

- **Statico**
  - Le applicazioni più diffuse hanno dei numeri di porta assegnati (well-known port numbers)
    - **Intervallo: 0 - 1023**
  - L'elenco dei port number è gestito dalla IANA (www.iana.org) ed aggiornato in tempo reale

Numero	Applicazione
7	Echo
21	FTP (File Transfer Protocol)
23	TELNET
25	SMTP (Simple Mail Transport Protocol

Numero	Applicazione
37	Time
53	Domain Name Server
80	HTTP
119	NNTP (USENET New Transfer Prot.)

- **Dinamico**
  - sono identificativi assegnati direttamente dal sistema operativo al momento dell'apertura della connessione
  - si utilizzano valori maggiori di 1023

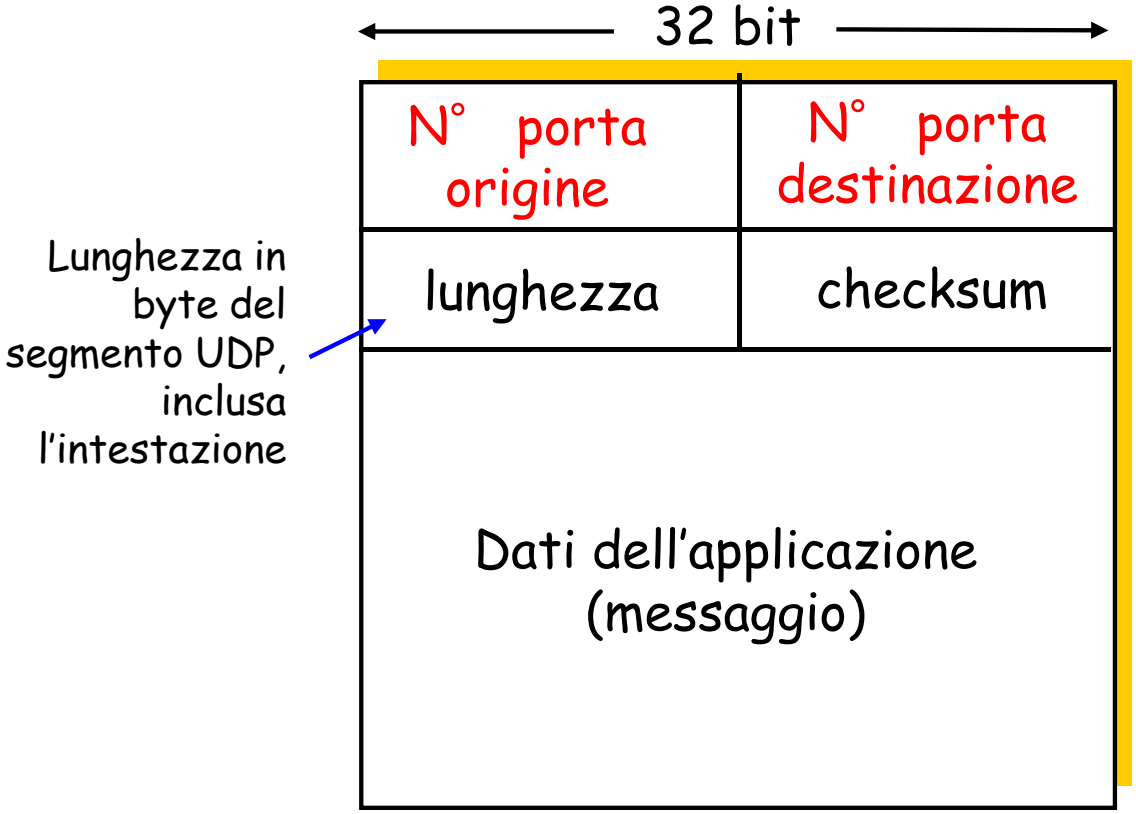
User Datagram Protocol
UDP

UDP: User Datagram Protocol [RFC 768]

- **Protocollo di trasporto "semplice"**
- **I segmenti UDP possono essere:**
  - perduti
  - consegnati fuori sequenza all'applicazione
- **Senza connessione**
  - no handshaking tra mittente e destinatario UDP
  - ogni segmento UDP è gestito in modo indipendente dagli altri
- **Senza controllo di congestione**
  - UDP emette i dati senza controllo

UDP

- **Utilizzato nelle applicazioni multimediali**
  - tollerano piccole perdite
  - sensibili al bit rate
- **Altri impieghi di UDP: protocolli su base transazionale**
  - DNS
  - SNMP
- **Trasferimento affidabile con UDP**
  - Aggiungere funzioni di affidabilità e recupero di errore al livello di applicazione



Struttura del segmento UDP

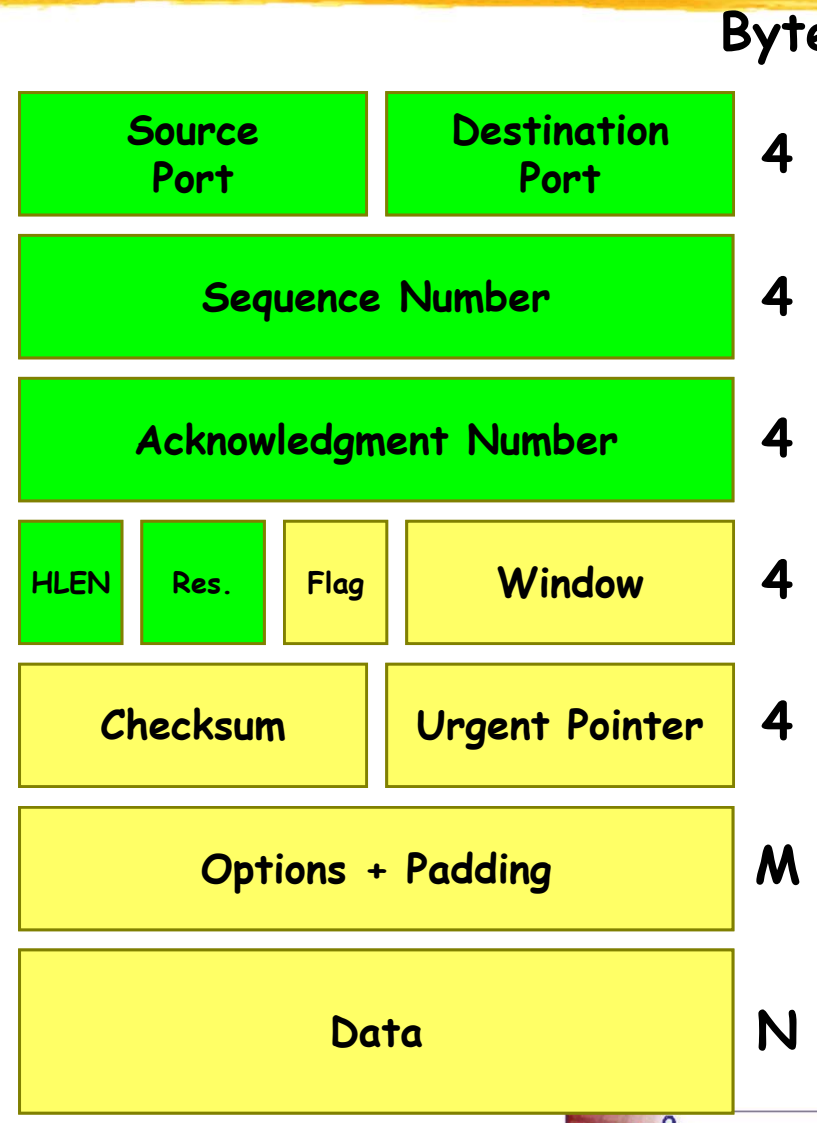
Transport Control Protocol
TCP

Il protocollo TCP

- E' un protocollo con connessione (RFC 793, 1122, 1323, 2018, 2581)
- Interpreta il flusso di dati proveniente dallo strato applicativo come sequenza di byte
- Funzioni
  - indirizzamento di una specifica applicazione
  - controllo di sequenza delle unità informative
  - controllo e recupero di errore
  - controllo di flusso
  - controllo di congestione

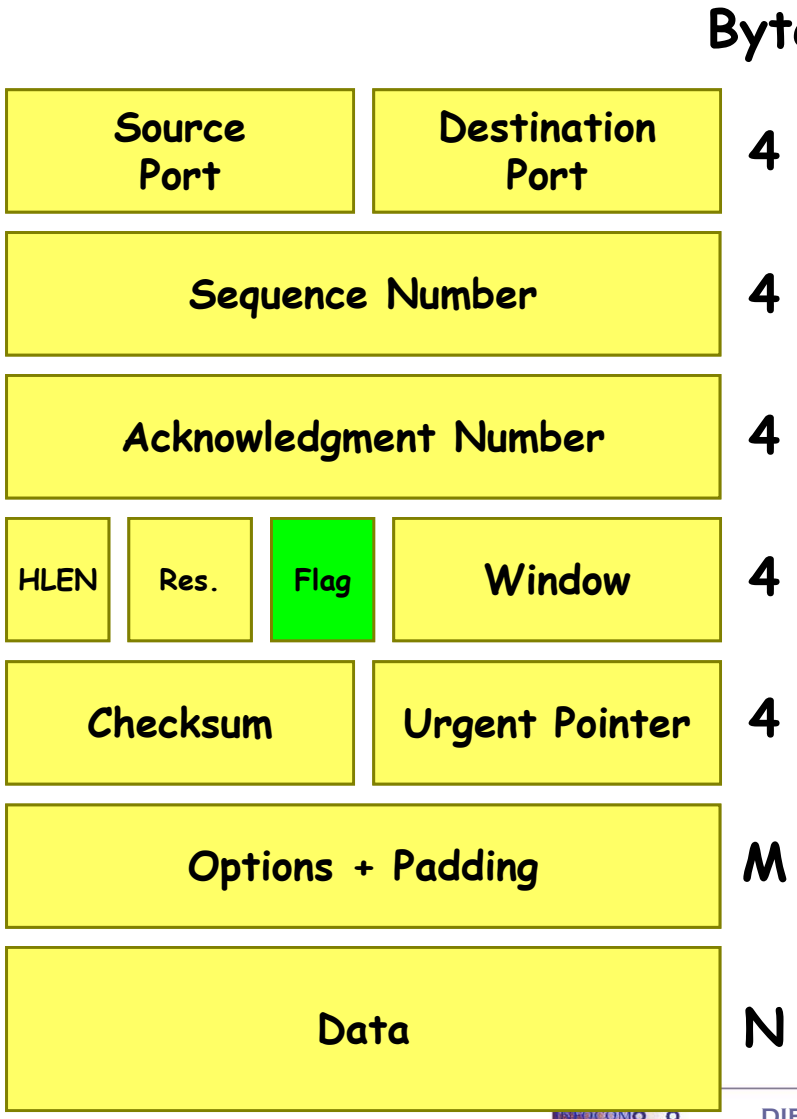
Segmento TCP

- **Source Port e Destination Port (16 bit ciascuno)**
- **Sequence Number (32 bit)**
  - Numero d'ordine del primo byte di dati contenuto nel campo dati
- **Acknowledgment Number (ACKNum) (32 bit)**
  - Contiene un valore valido se il bit ACK del campo Flag è uguale a 1
  - Contiene il numero di sequenza del prossimo byte che l'entità ricevente si aspetta di ricevere
- **HLEN (4 bit)**
  - contiene il numero di parole di 32 bit contenute nell'intestazione del segmento
  - l'intestazione del segmento non supera i 60 ottetti ed è sempre un multiplo di 32
- **Reserved (6 bit)**
  - riservato per usi futuri



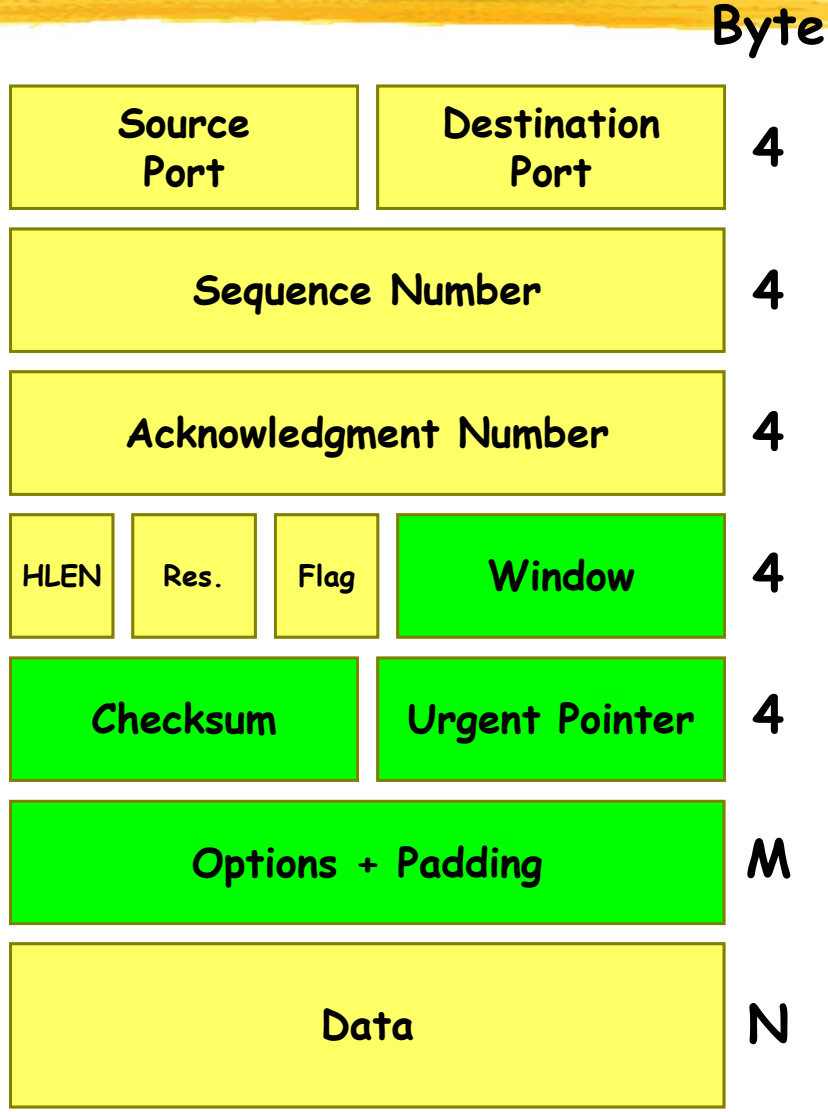
Segmento TCP

- Flag (6 bit)
  - **URG (urgent)**: è uguale a 1 quando il campo "Urgent Pointer" contiene un valore significativo
  - **ACK (riscontro)**: è uguale a 1 quando il campo "Acknowledgement Number" contiene un valore valido
  - **PSH (push)**: è uguale a 1 quando l'applicazione indica che i dati vengano consegnati all'applicazione ricevente prescindendo dal riempimento dei buffer di ricezione
  - **RST (reset)**: è uguale a 1 in caso di richiesta di re-inizializzazione della connessione
  - **SYN (sincronizzazione)**: è uguale a 1 solo nel primo segmento inviato durante la fase di sincronizzazione fra le entità TCP
  - **FIN (fine)**: è uguale a 1 quando la sorgente ha esaurito i dati da trasmettere (fase di release)



Segmento TCP

- **Window (16 bit)**
  - larghezza della finestra misurata in ottetti
  - è il numero di ottetti che, ad iniziare dal valore di ACK Number, l'emettitore del segmento autorizza a trasmettere
- **Checksum (16 bit)**
  - protegge l'intero segmento più alcuni campi dell'header IP (pseudo header)
- **Urgent Pointer (16 bit)**
  - contiene il numero di sequenza dell'ultimo byte dei dati che devono essere consegnati urgentemente al processo ricevente
- **Options (di lunghezza variabile)**
  - sono presenti solo raramente
- **Padding (di lunghezza variabile)**
  - impone che l'intestazione abbia una lunghezza multipla di 32 bit



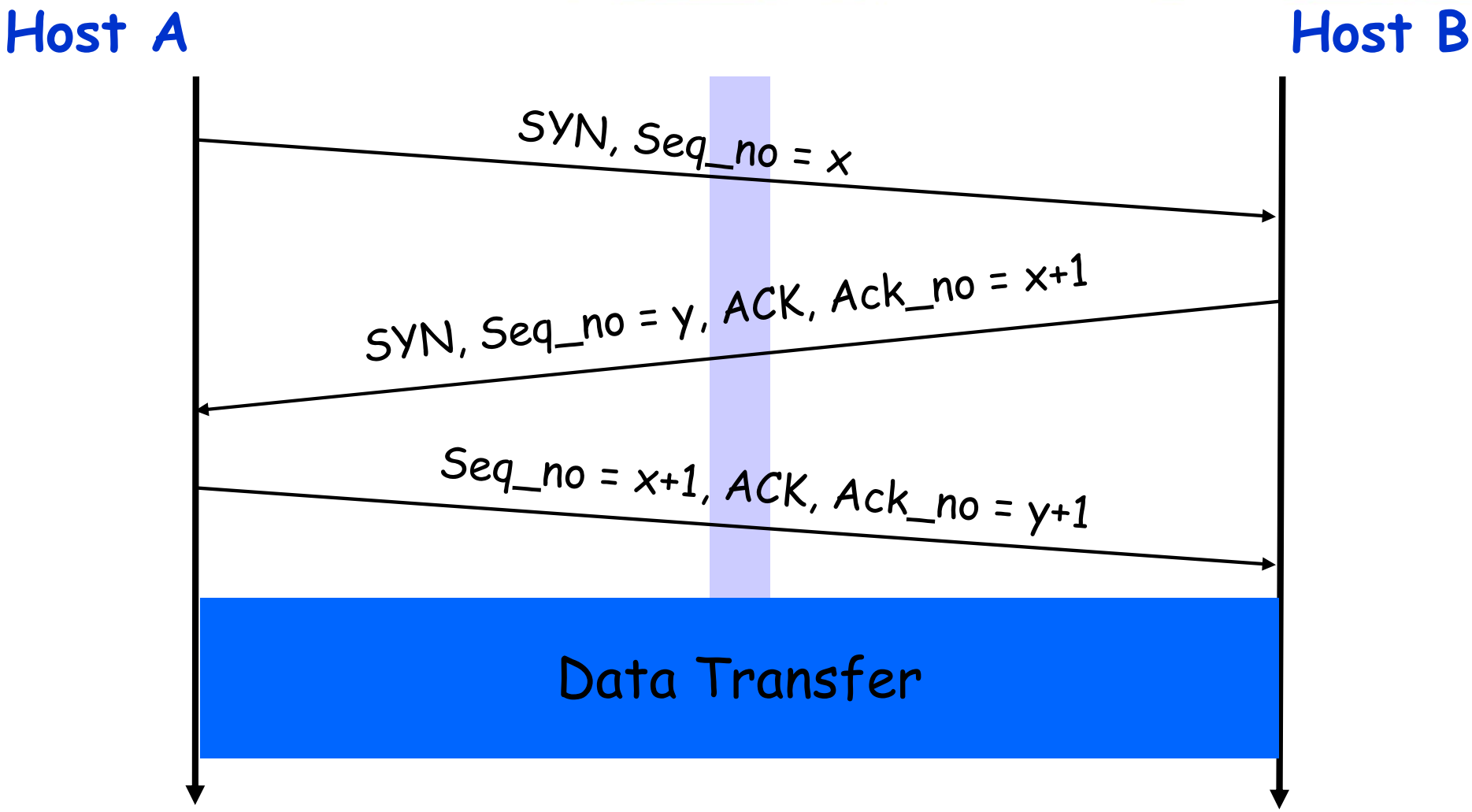
TCP

Gestione della connessione

La connessione TCP

- **Il protocollo TCP è un protocollo del tipo con connessione**
- **Nella fase di instaurazione della connessione le due entità TCP remote si sincronizzano scambiandosi**
  - Gli identificatori dei socket (port, IP address)
  - Il proprio numero di sequenza iniziale, che rappresenta il numero a partire dal quale tutti gli ottetti emessi saranno sequenzialmente numerati
  - Il valore iniziale della finestra di ricezione
- **Handshaking a tre passi**
  - 1) l'host A invia un segmento SYN all'host B
    - specifica il numero di sequenza iniziale utilizzato nel verso A→B
    - Non contiene dati
  - 2) l'host B riceve il segmento SYN e risponde con un segmento SYN ACK
    - l'host B alloca i buffer
    - specifica il numero di sequenza iniziale del server utilizzato nel verso B→A
  - 3) L'host A riceve un segmento SYN ACK e risponde con un segmento ACK, che può contenere dati

Three-way handshake

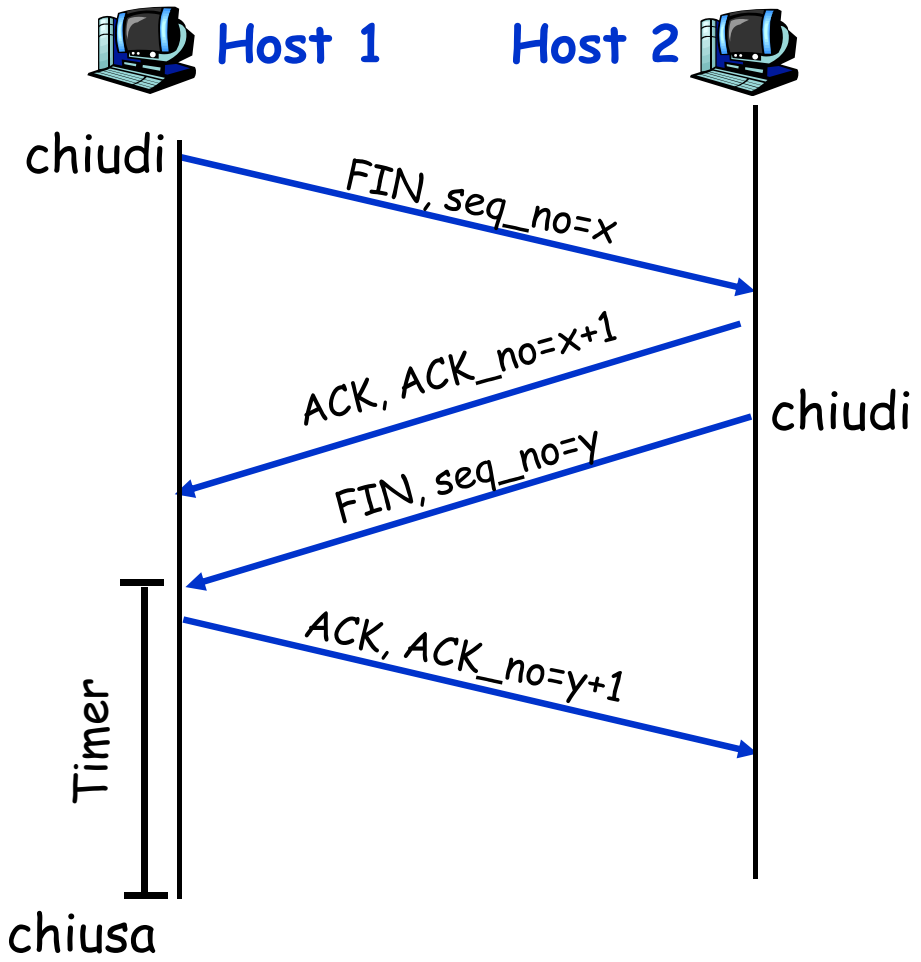


Maximum Segment Size (MSS)

- Quando l'entità TCP emittente invia la prima TCP-PDU (SYN) può inserire l'informazione relativa alla massima dimensione del campo dei dati di utente di una TCP-PDU (Maximum Segment Size - MSS)
- L'entità ricevente risponde comunicando la propria MSS
- Nel caso di uno scambio bidirezionale, la dimensione della MSS è scelta in modo indipendente nei due versi e può quindi essere diversa nelle due direzioni

Chiusura (release) di una connessione

- L'Host 1 invia un segmento di controllo FIN al server
- L'Host 2 riceve il segmento FIN, risponde con un ACK
- L'Host 2 chiude la connessione e invia un FIN
- L'Host 1 riceve FIN e risponde con un ACK
- Viene attivato un timer
  - si risponde con un ACK ai FIN ricevuti
- L'Host 2 riceve un ACK
  - La connessione viene chiusa

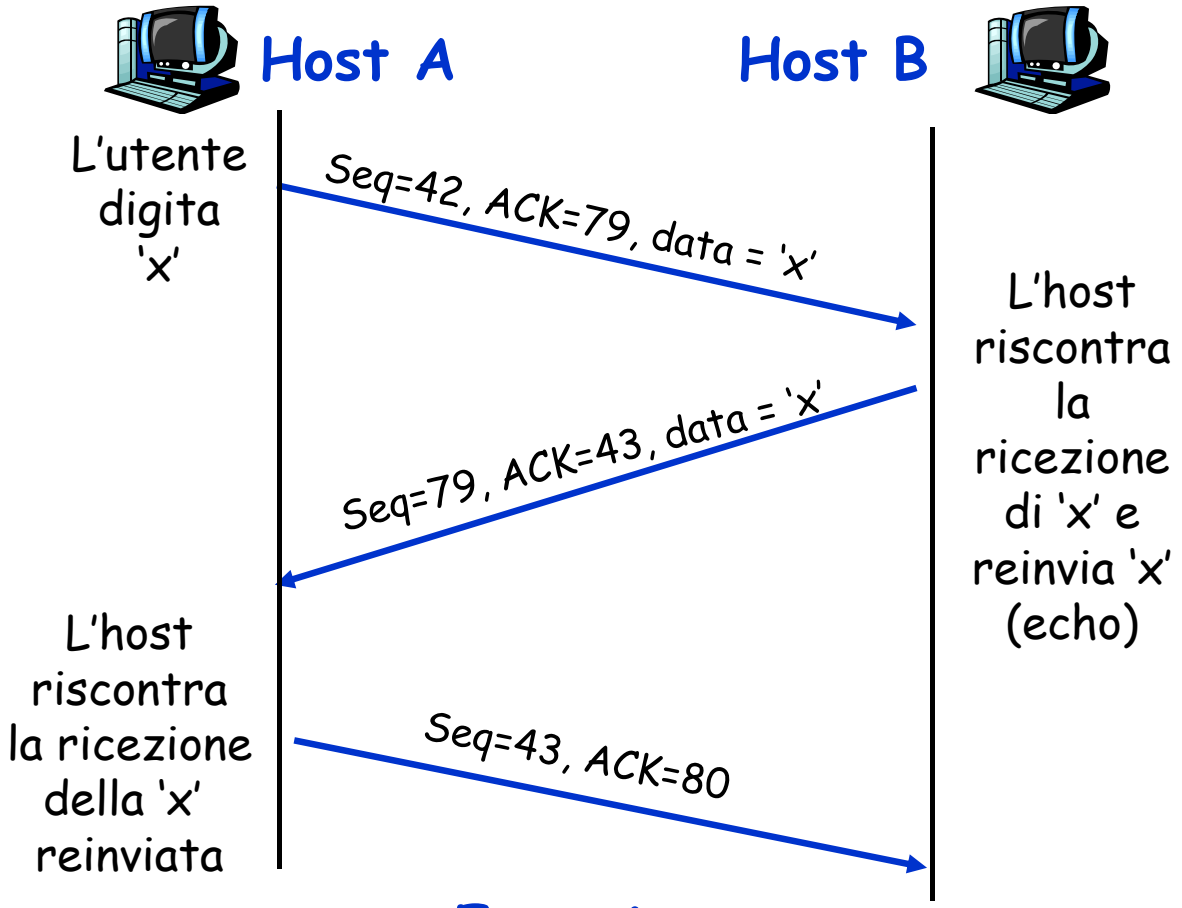


TCP

Controllo di sequenza e Controllo d'errore

Numeri di sequenza e ACK

- **Numeri di sequenza:**
  - "numero" del primo byte del segmento nel flusso di byte
- **ACK:**
  - numero di sequenza del prossimo byte atteso dall'altro host
  - ACK cumulativo
- **La gestione dei segmenti fuori sequenza non è specificata dallo standard**
  - dipende dall'implementazione (Es. scarto o memorizzazione dei segmenti fuori sequenza)



Esempio TELNET

Controllo d'errore

- **In TCP il controllo d'errore è basato sull'impiego di**
  - una codifica a rivelazione d'errore che
    - è effettuata dall'entità TCP emittente e il cui risultato è inserito nell'intestazione del segmento (Checksum)
    - è utilizzata dall'entità TCP ricevente per la rivelazione di eventuali errori
  - riscontri positivi (ACK), che possono essere inoltrati dall'entità TCP ricevente con segmenti vuoti (senza dati) oppure in modalità "piggybacking"
  - Retransmission Timeout (RTO)
    - È un temporizzatore di valore variabile in modo adattativo attivato dall'entità emittente
    - è attivato nel momento in cui un segmento viene inoltrato su una connessione uscente (il timer è associato all'ultimo segmento non riscontrato)
    - è disattivato nel momento in cui viene ricevuto un ACK relativo al segmento corrispondente e quando tale ricezione avviene prima che l'RTO si esaurisca

Riscontri

- **L'entità TCP ricevente può emettere i riscontri (ACK) secondo due modalità**
  - **Immediata**, appena vengono accettati i dati, emette immediatamente un segmento vuoto (senza dati) che contiene l'appropriato numero di riscontro
  - **Cumulativa**, appena vengono accettati i dati, tiene memoria della necessità di inviare un riscontro, ma aspetta un segmento in uscita nel quale inserirlo
    - per evitare lunghi ritardi, attiva un timer di finestra
    - se il tempo di questo timer si esaurisce prima che venga inviato un riscontro, emette un segmento vuoto che contiene l'appropriato numero di riscontro

Round Trip Time (RTT) e timeout

- **Come impostare il valore del timeout di TCP ?**
- **Più grande di RTT**
  - ma RTT varia
- **Troppo piccolo: timeout prematuro**
  - ritrasmissioni non necessarie
- **Troppo grande: reazione lenta alla perdita dei segmenti**

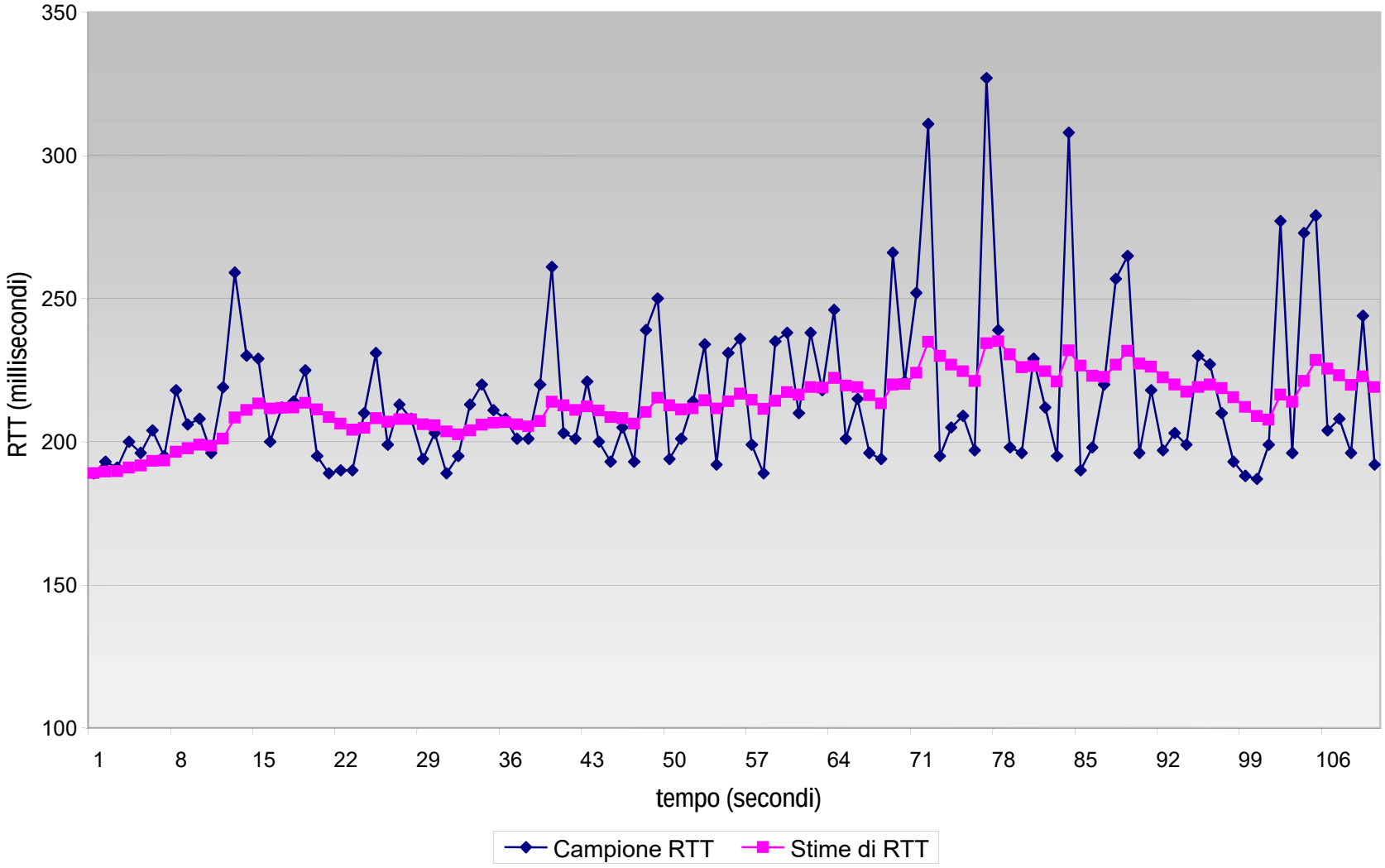
- **Come stimare RTT ?**
- **SampleRTT: tempo misurato dall'istante di trasmissione di un segmento fino all'istante di ricezione dell'ACK relativo**
  - ignora le ritrasmissioni
- **SampleRTT varia, quindi occorre una stima "smoothed" di RTT**
  - media di più camponi recenti del RTT (non semplicemente il valore corrente di SampleRTT)

Round Trip Time (RTT) e timeout

$$EstimatedRTT = (1 - \alpha)*EstimatedRTT + \alpha*SampleRTT$$

- Media mobile esponenziale ponderata
- L'influenza dei vecchi campioni decresce esponenzialmente
- Valore di default: $\alpha = 0{,}125$

Esempio di stima di RTT



Determinazione del Timeout

- **EstimatedRTT più un "margine di sicurezza"**
  - grande variazione di EstimatedRTT -> margine di sicurezza maggiore
- **Stima della deviazione standard dell'EstimatedRTT**

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT-EstimatedRTT|$$

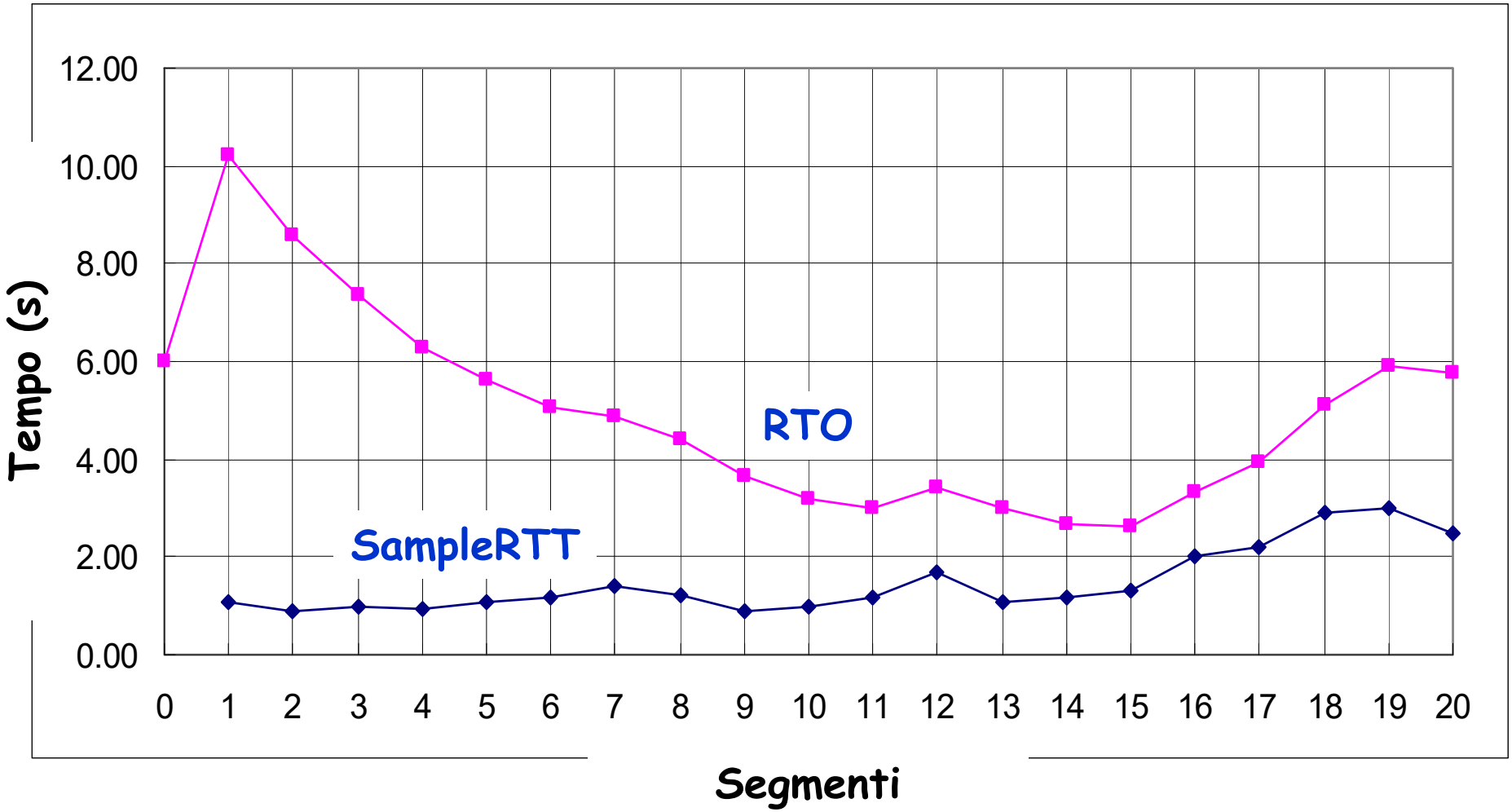
  - $\beta = 0,25$
- **Valore Retransmission TimeOut (RTO)**

$$RTO = EstimatedRTT + 4*DevRTT$$

estimated RTT

margine di sicurezza

Esempio di calcolo del RTO



Exponential RTO Backoff

- **Determina il valore di RTO associato ad un <u>segmento riemesso</u>**
  - è consigliabile variare il valore di RTO sui segmenti riemessi perché l'esaurimento dell'RTO è probabilmente dovuto a congestione in rete
  - è consigliabile variare il valore di RTO delle sorgenti che sono coinvolte nella congestione per evitare riemissioni contemporanee
- **Una sorgente TCP aumenta il valore di RTO per ogni riemissione (*exponential backoff process*) (normalmente q=2)**

$$RTO_{i+1} = q \cdot RTO_i$$

Algoritmo di Karn

- **L'entità TCP ricevente non distingue se il riscontro si riferisce**
  - alla prima emissione del segmento (RTO troppo elevato con perdita di efficienza e inutili ritardi)
  - alla riemissione del segmento (RTO troppo breve e quindi riemissioni eccessive e nuovi errori di misura).
- **L'algoritmo di Karn stabilisce di**
  - non considerare il RTT dei segmenti riemessi
  - usare come RTO il valore dato dalla procedura di exponential backoff
  - aggiornare il valore di RTO solo al momento della ricezione di un ACK di un segmento non riemesso

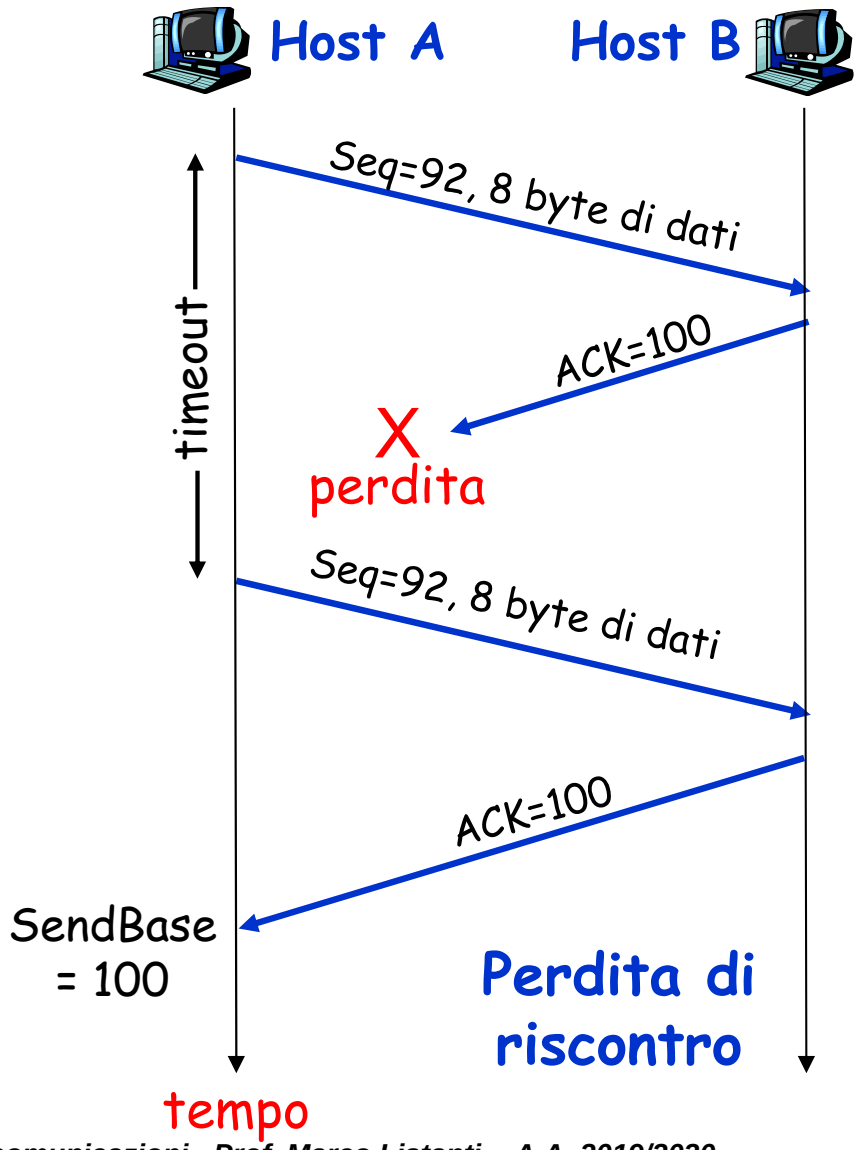
TCP

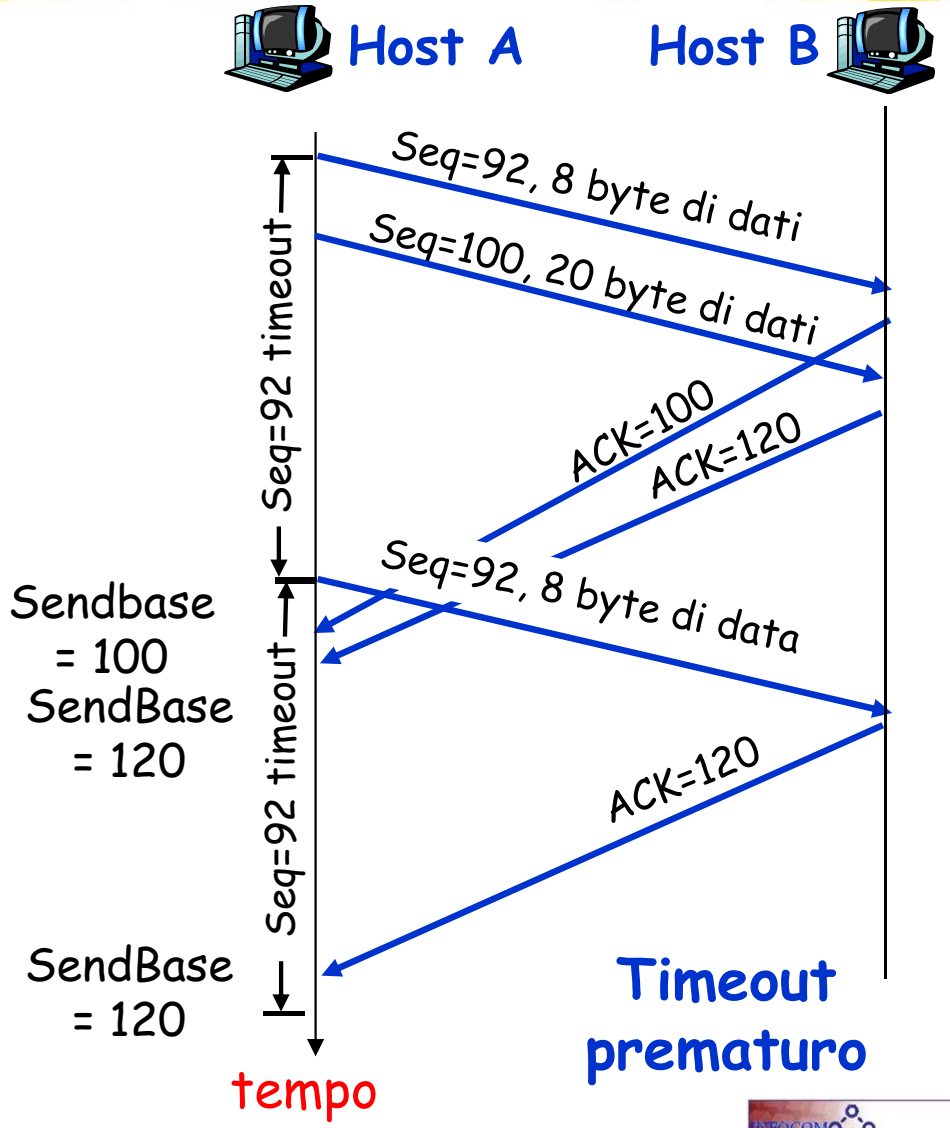
Controllo d'errore

Controllo d'errore

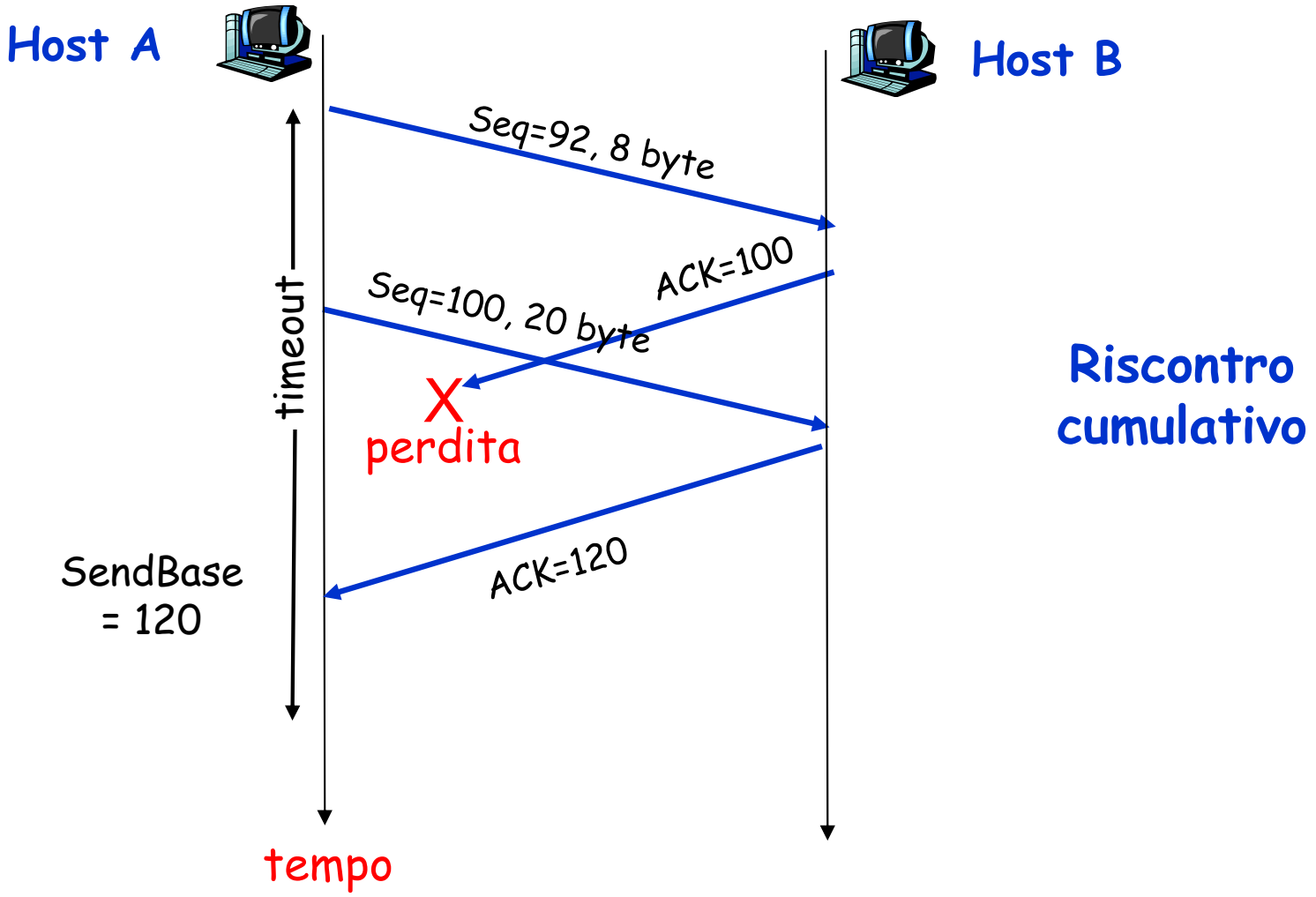
- TCP ha lo scopo di offrire un servizio di trasferimento dati affidabile utilizzando il servizio inaffidabile offerto dallo strato di rete (IP)
- Si utilizzano esclusivamente:
  - segmenti ACK
  - timeout di ritrasmissione
- Le ritrasmissioni sono avviate da
  - esaurimento del timeout
  - ACK duplicati

Ritrasmissione





Ritrasmissione



Riscontro cumulativo

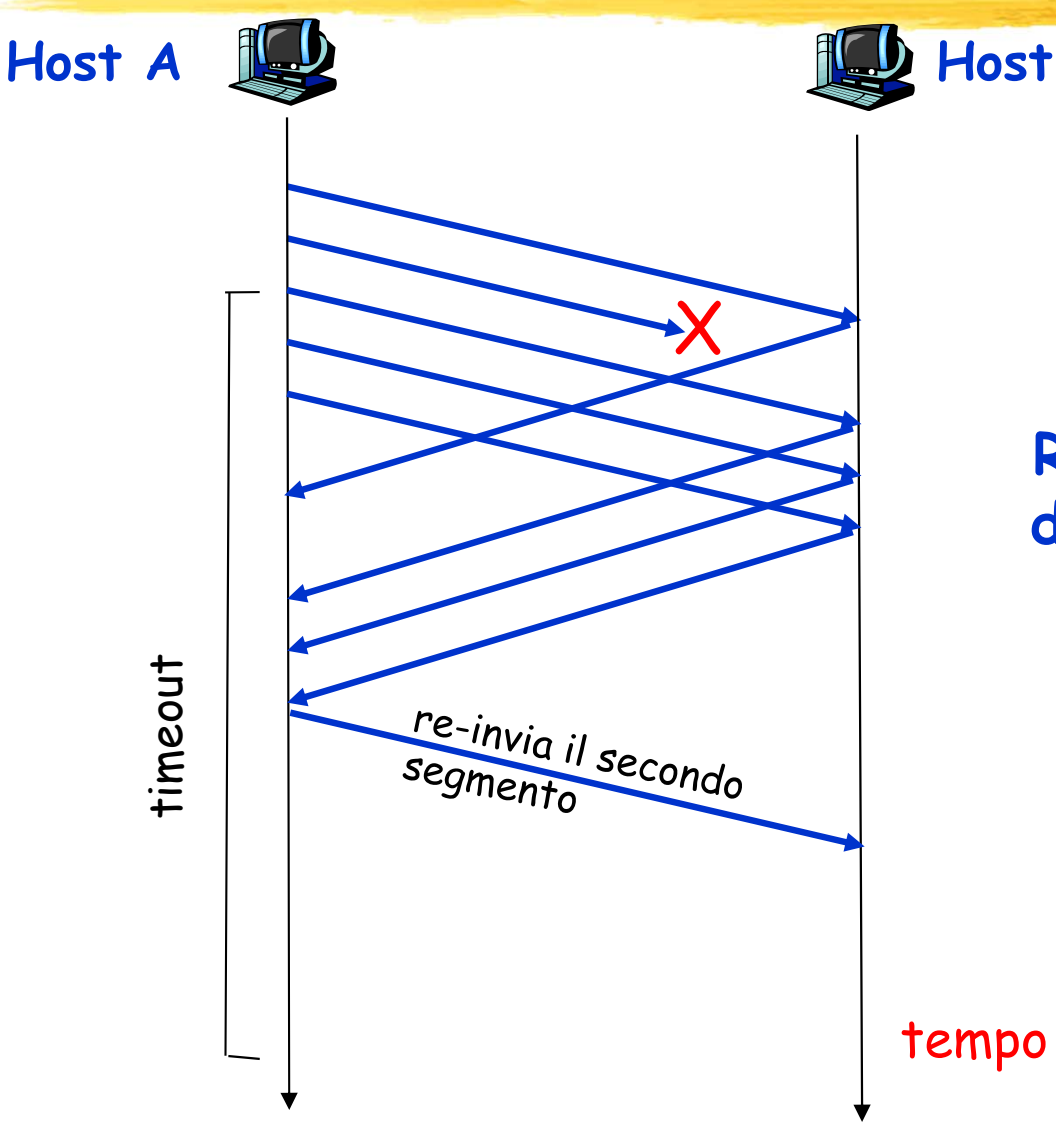
Generazione di ACK [RFC 1122, RFC 2581]

Evento presso il ricevente	Azione del ricevente
Arrivo ordinato di un segmento. Tutti i dati fino al numero di sequenza atteso sono già stati riscontrati	ACK ritardato. Attende fino a 500 ms l'arrivo del prossimo segmento. Se il segmento non arriva, invia un ACK
Arrivo ordinato di un segmento. Un altro segmento è in attesa di trasmissione dell'ACK	Invia immediatamente un singolo ACK cumulativo, riscontrando entrambi i segmenti
Arrivo non ordinato di un segmento con numero di sequenza superiore a quello atteso, viene rilevato un "fuori sequenza"	Invia immediatamente un ACK duplicato, indicando il numero di sequenza del prossimo byte atteso
Arrivo di un segmento che ripristina parzialmente o completamente il "fuori sequenza"	Invia immediatamente un ACK, ammesso che il segmento sia in sequenza con l'ultimo segmento riscontrato

Fast retransmit

- **Il periodo di timeout spesso è relativamente lungo**
  - elevato ritardo prima di ritrasmettere il pacchetto perduto
- **L'entità TCP emittente può rivelare precocemente i segmenti perduti tramite l'analisi degli *ACK* duplicati**
  - L'entità TCP emittente spesso invia molti segmenti
  - Se un segmento viene smarrito, è probabile che ci saranno molti *ACK* duplicati
- **Se l'entità TCP emittente riceve 3 *ACK* duplicati per lo stesso dato, suppone che il segmento che segue il dato riscontrato sia andato perduto**
  - ritrasmissione rapida
  - si ritrasmette il segmento prima che scada il timer

Fast retransmit



Ritrasmissione di un segmento dopo un triplice ACK duplicato

TCP

Controllo di flusso

Controllo di Flusso

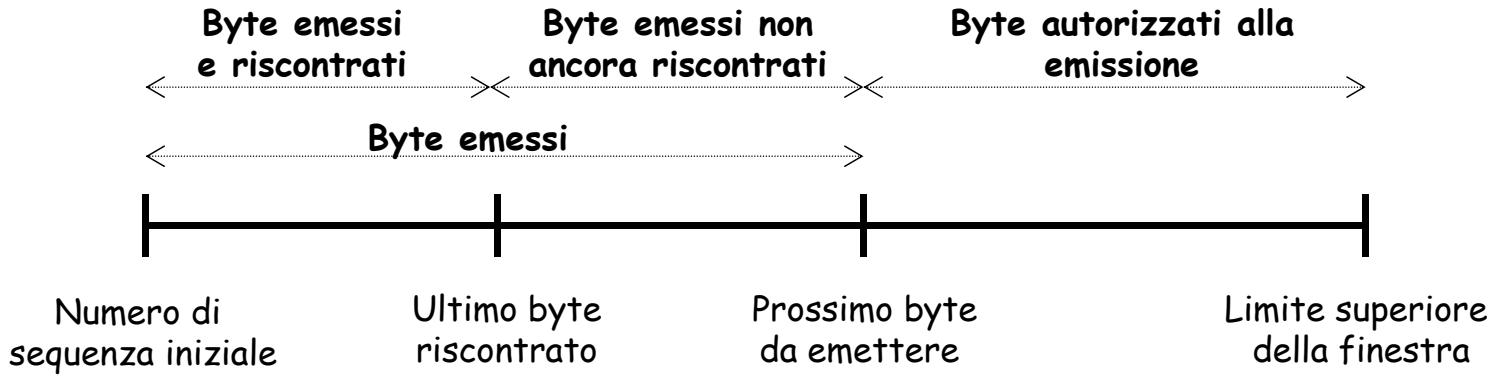
- **Il controllo di flusso ha lo scopo di limitare il ritmo di emissione dei dati da parte di un host per evitare la saturazione della capacità del buffer di ricezione**
- **TCP utilizza un controllo di flusso basato su una finestra scorrevole di larghezza variabile**
  - Lo scorrimento e la larghezza della finestra sono controllati dall'entità TCP ricevente
- **Il controllo di flusso opera a livello di byte**
  - Gli ottetti sono numerati in sequenza a partire dal numero scelto durante il 3-way handshaking (procedura di instaurazione della connessione)

Controllo di Flusso

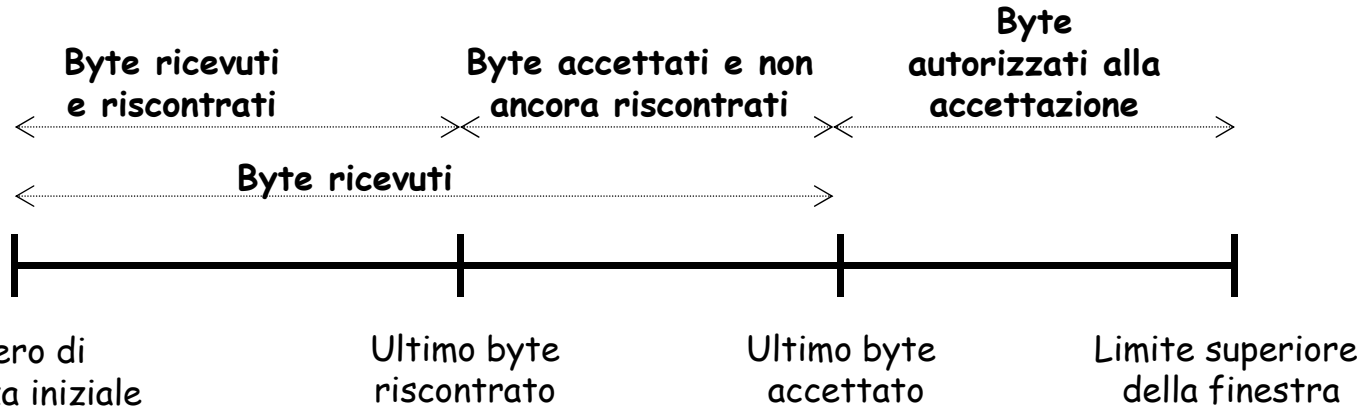
- **La procedura di controllo di flusso TCP utilizza i seguenti parametri:**
  - SN (Sequence Number)
    - SN si riferisce al primo byte contenuto nel segmento
  - AckN (Acknowledgement Number)
    - Identifica il numero di sequenza del prossimo byte che l'entità ricevente aspetta di ricevere
  - RecWindow (Window).
    - Identifica il numero massimo di byte che l'entità emittente può emettere consecutivamente senza ricevere riscontro per alcuno di questi
- **Un riscontro (AckN=X e RecWindow=W) significa che**
  - sono riscontrati tutti gli ottetti ricevuti fino a quello numerato con X–1;
  - l'entità TCP emittente è autorizzata a trasmettere fino a ulteriori W ottetti, ovvero fino all'ottetto numerato con X+W – 1

Controllo della Finestra

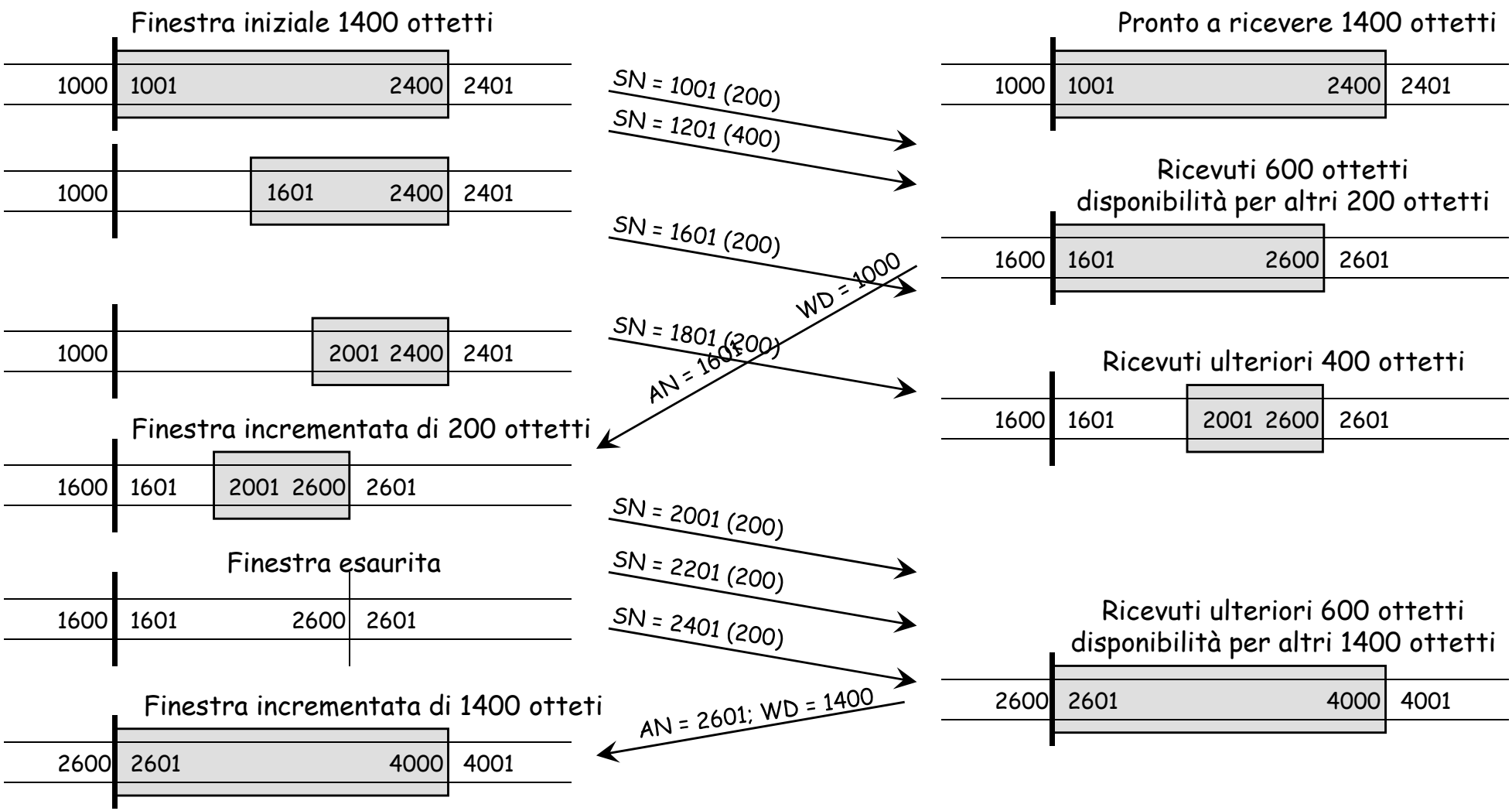
- **Puntatori per il controllo a finestra lato emittente**



- **Puntatori per il controllo a finestra lato ricevente**



Esempio



Throughput di una connessione TCP

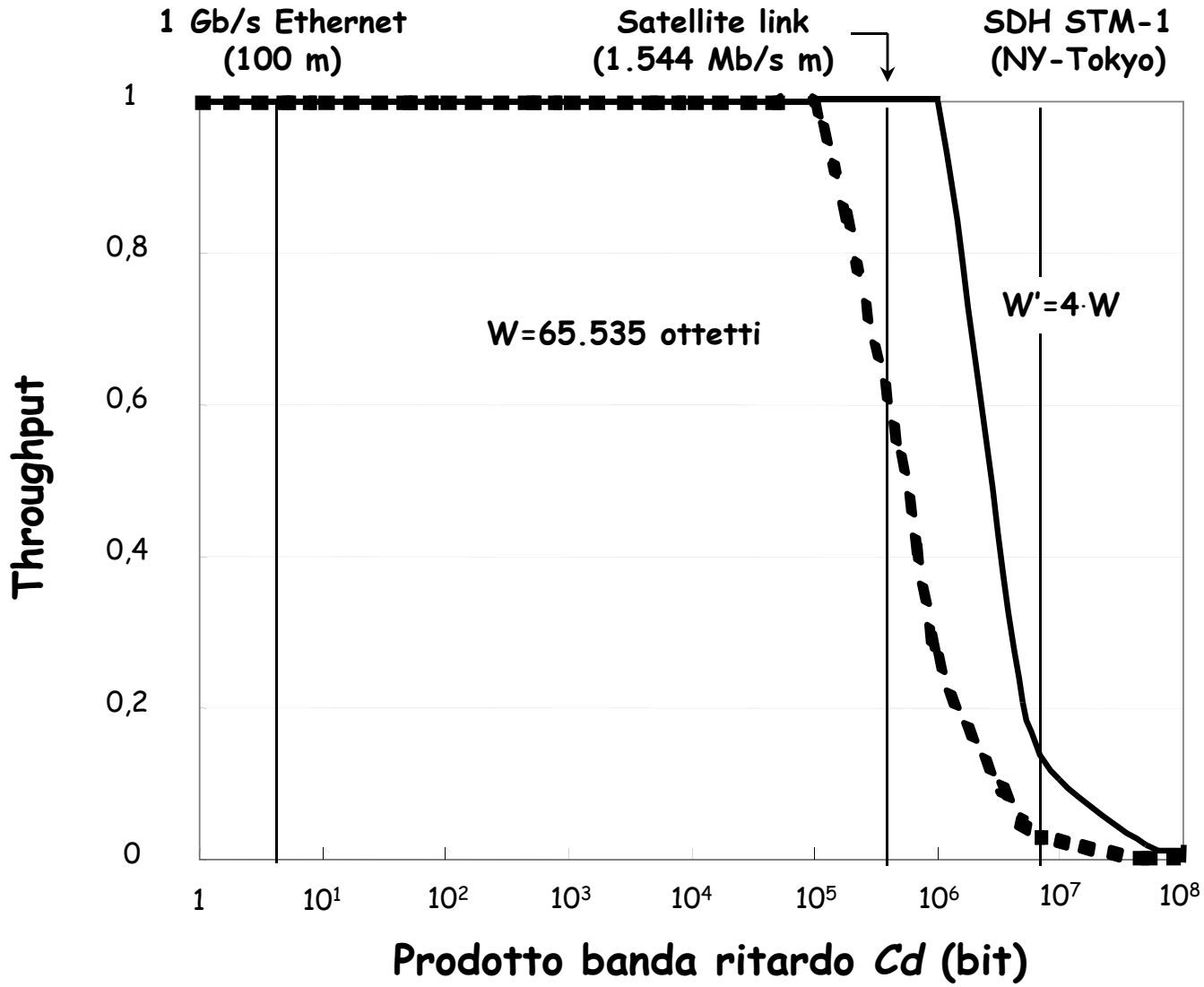
- **L'efficienza (throughput - TH) di una connessione TCP, nell'ipotesi di overhead nullo e di assenza di ritrasmissioni, è dato da**

$$TH = \frac{tempo\ di\ trasmissione\ utile\ in\ un\ RTT}{RTT} = min\left[1, \frac{W/C}{2\alpha/C}\right] = min\left[1, \frac{W}{2\alpha}\right] =$$

$$= \begin{cases} 1 & se \quad W \geq 2\alpha \\ \frac{W}{2\alpha} & se \quad W < 2\alpha \end{cases}$$

- **dove**
  - W : larghezza (byte) della finestra in trasmissione
  - C : bit rate della connessione
  - d : ritardo di propagazione sulla connessione
  - Cd : prodotto banda-ritardo
  - a = C d/8 (prodotto banda ritardo espresso in byte)

Throughput di una connessione TCP



- **Il TCP non è adatto a collegamenti ad elevato bitrate o con elevato ritardo**
  - Se il prodotto banda ritardo è elevato il throughput del TCP decresce sensibilmente
- **L'aumento della dimensione della finestra non è un intervento risolutivo**

TCP

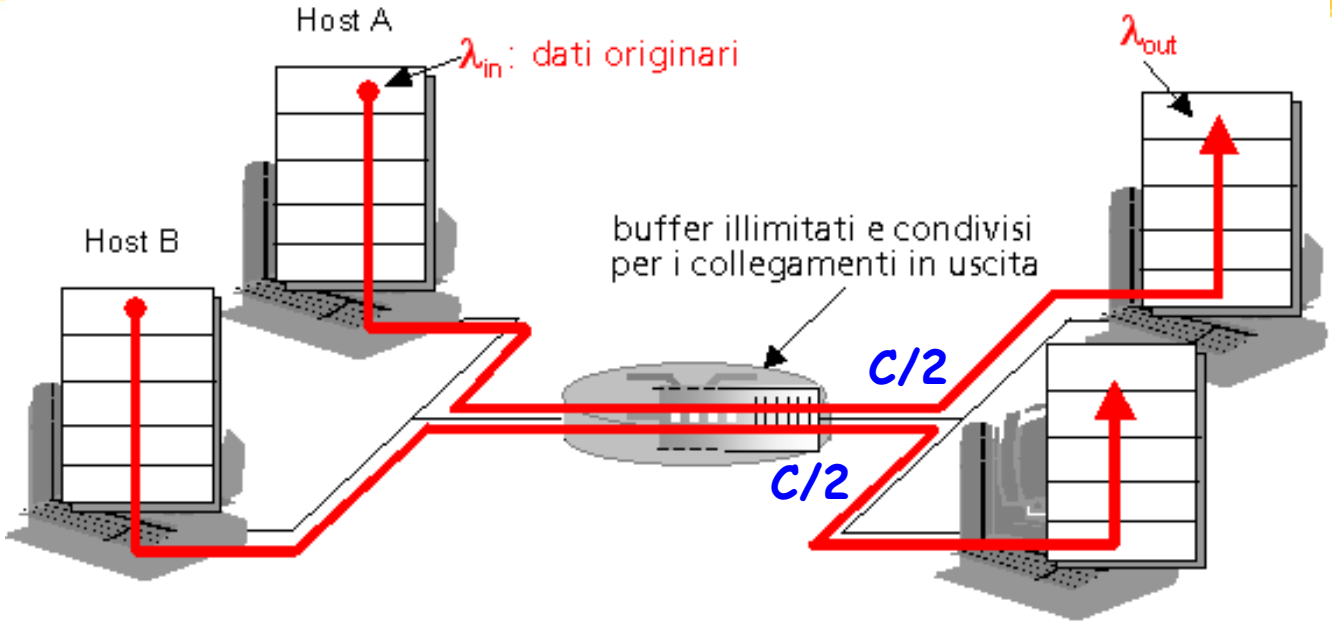
Controllo di congestione

Principi del controllo di congestione

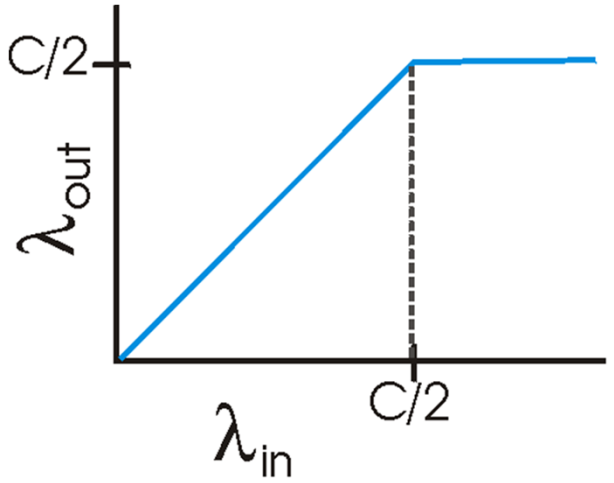
- **Definizione**
  - "le sorgenti trasmettono dati ad un rate che la rete non è in grado di gestire"
- **Il controllo di congestione è diverso dal controllo di flusso**
  - Il controllo di congestione riguarda la rete
  - Il controllo di flusso riguarda l'entità ricevente
- **Sintomi**
  - elevati ritardi (accodamento nei buffer dei router)
  - pacchetti persi (overflow nei buffer dei router)

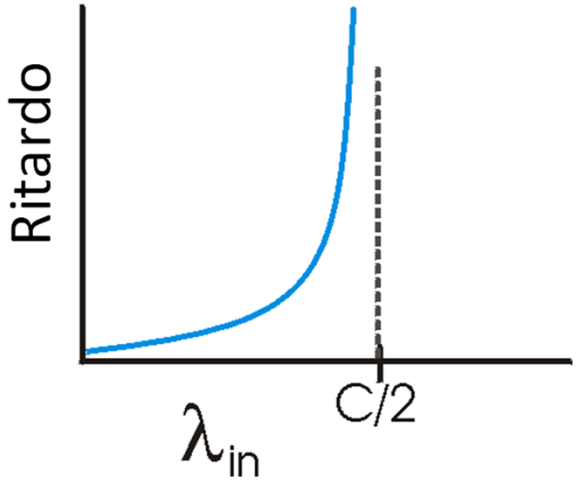
Esempio: scenario 1

- Due mittenti, due destinatari
- Condivisione di un link di capacità *C*
- Buffer illimitati
- Nessuna ritrasmissione



Massimo throughput di ciascuna connessione

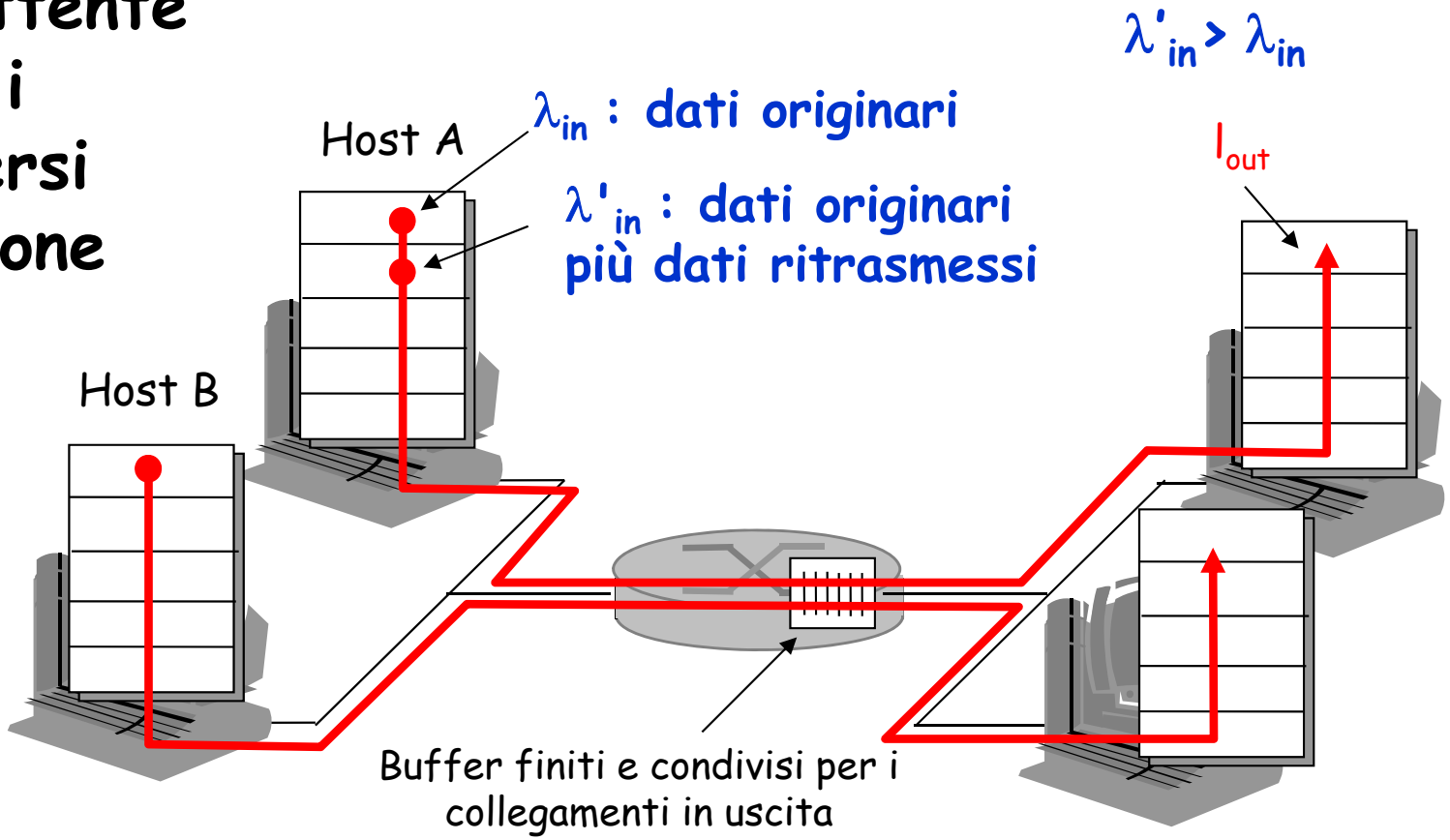




Il delay cresce se λ_{in} si avvicina alla capacità massima della connessione

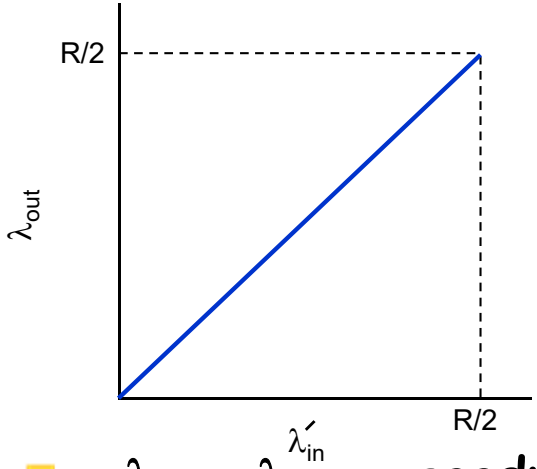
Esempio: scenario 2

- Buffer finito
- L'entità emittente ritrasmette i pacchetti persi per saturazione del buffer

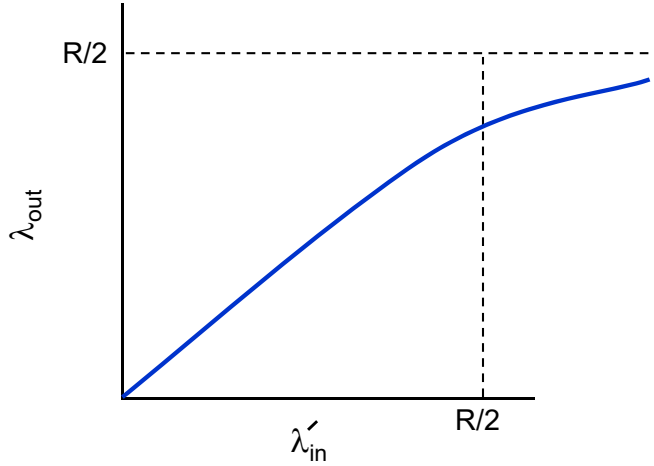


Esempio: scenario 2

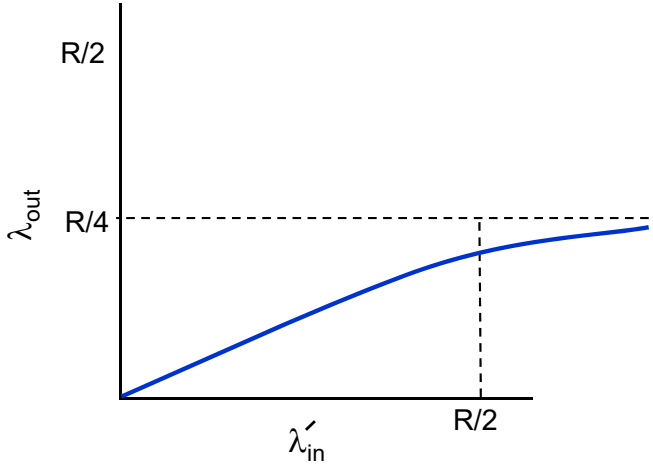
a – Assenza congestione



b – Ritrasmissioni solo per perdita per congestione



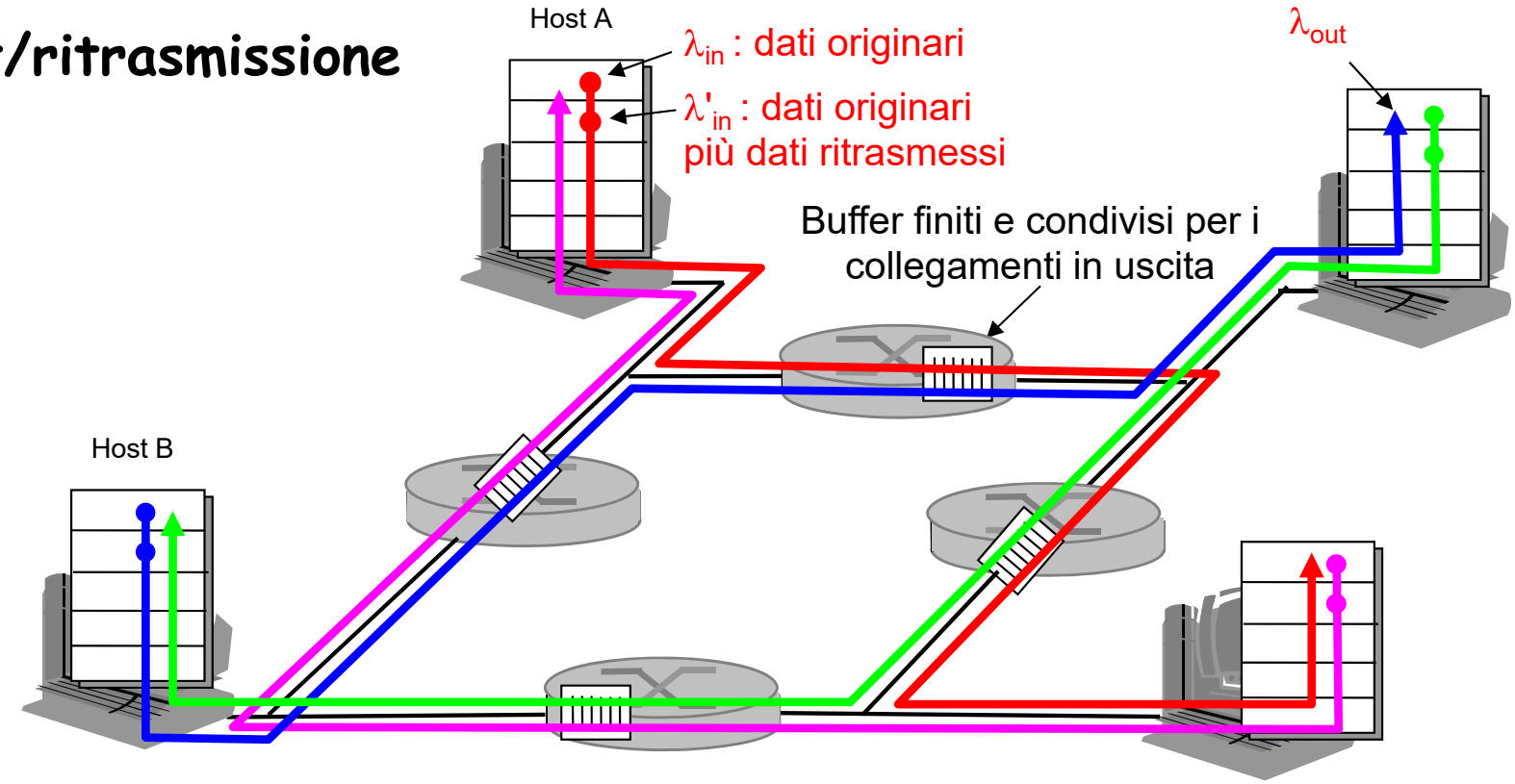
c – Ritrasmissioni anche per ritardi eccessivi



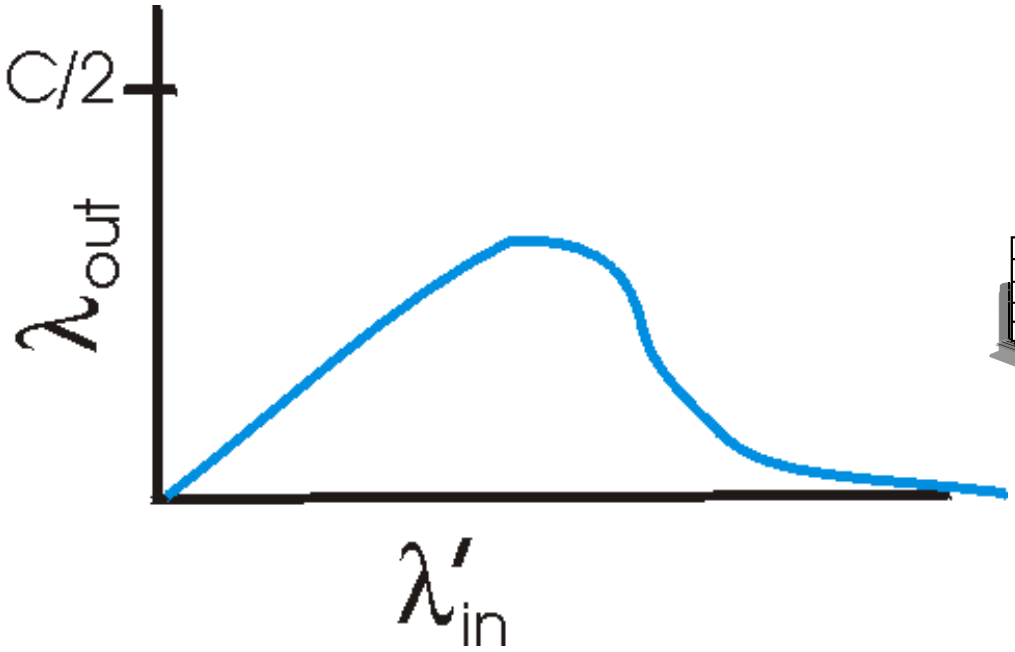
- $\lambda_{in} = \lambda_{out}$ = **goodput**
  - rate di arrivo di pacchetti utili a destinazione
- **Caso a: tutti i pacchetti arrivano a destinazione ($\lambda'_{in} = \lambda_{in} = \lambda_{out}$)**
- **Caso b: le ritrasmissioni rendono λ'_{in} maggiore di λ_{out} quindi il goodput λ_{out} diminuisce ($\lambda'_{in} > \lambda_{in} = \lambda_{out}$)**
- **Caso b: la ritrasmissione dei pacchetti ritardati aumenta ancora λ'_{in} ed il goodput diminuisce ulteriormente ($\lambda'_{in} >> \lambda_{in} = \lambda_{out}$)**

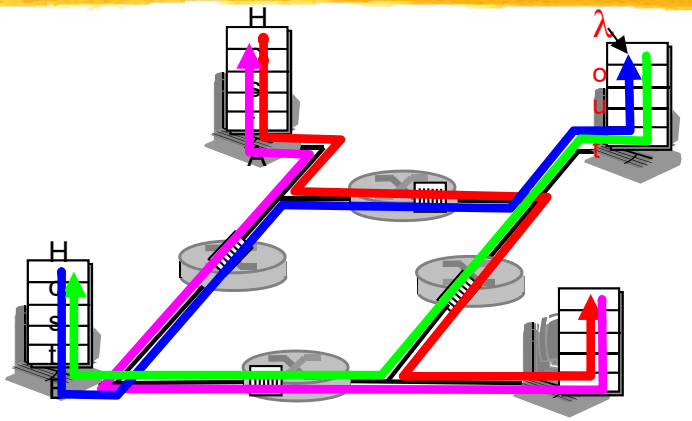
Esempio: scenario 3

- Quattro mittenti
- Percorsi multihop
- Timeout/ritrasmissione



Esempio: scenario 3





- Un altro "costo" della congestione:
- Quando un pacchetto viene perso, la capacità trasmissiva utilizzata sui collegamenti di upstream per instradare il pacchetto risulta sprecata

Approcci al controllo della congestione

- Controllo di congestione punto-punto
- 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 (Explicit Congestion Notification - ECN)
  - L'entità ricevente comunica in modo esplicito al mittente il bitrate in trasmissione

Controllo di congestione TCP

- **Il protocollo TCP utilizza i seguenti meccanismi**
  - esaurimento dell'RTO come un sintomo di congestione
  - la finestra di congestione (**Congestion Window - Congwin**)
    - La finestra di congestione si affianca alla finestra di ricezione operante nel controllo di flusso e impone una limitazione addizionale alla quantità di traffico che un host può inviare in una connessione
  - soglia (Threshold)
    - il valore della soglia è pari alla metà del valore della Congwin al momento in cui viene rilevata una perdita
    - all'inizio della connessione (slow start) la soglia è posta uguale a ∞
- **L'entità emittente determina nel tempo il valore della finestra disponibile (Avalilable Window - Awdn)**
  - Awdn = numero di segmenti di lunghezza massima (MSS) che possono essere inviati senza riscontro

Controllo di congestione TCP

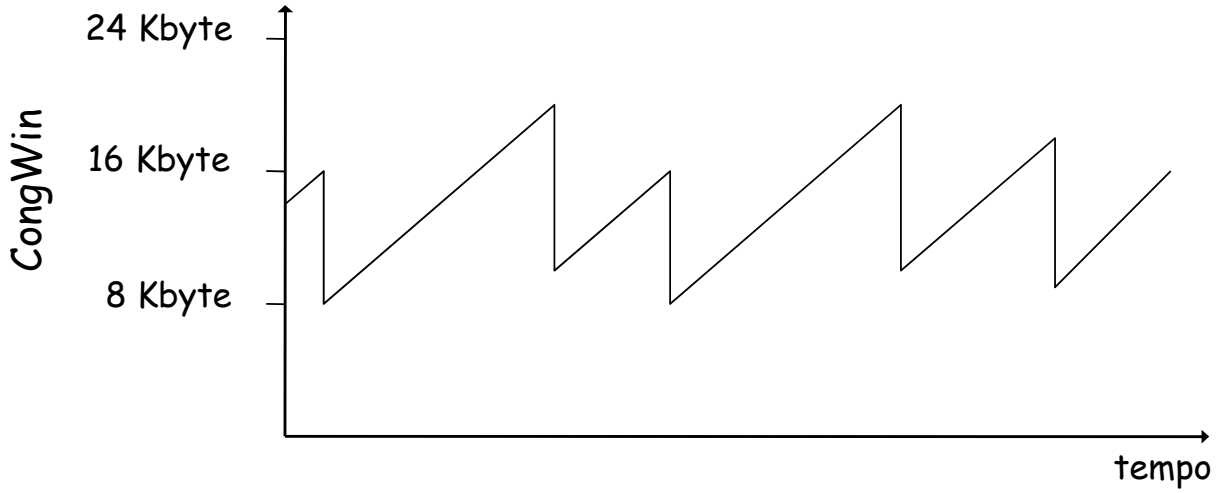
- **Il valore di Awdn**
  - non deve superare il minimo tra le larghezze **Congwin** della finestra di congestione e **RecWindow** della finestra di ricezione

$$Awdn \leq \min \{Congwin, RecWindow\}$$

  - Congwin ed RecWindow sono quantità espresse in numero di segmenti MSS
  - RecWindow è la larghezza comunicata nell'ultimo ACK ricevuto e ottenuta dall'entità TCP emittente dividendo il numero contenuto nel campo Window di questo ACK per il numero di ottetti che compongono una MSS

Additive-Increase Multiplicative-Decrease (AIMD)

- **Aumenta il valore di CongWin (sondando la rete) fino a quando non si verifica una perdita**
- **Incremento additivo**
  - aumenta CongWin di 1 MSS a ogni RTT in assenza di eventi di perdita
- **Decremento Moltiplicativo**
  - riduce a metà CongWin dopo un evento di perdita



Controllo di congestione AIMD

Controllo di congestione TCP

- Approssimativamente il rate di emissione dei segmenti è dato da

$$\text{Frequenza d'invio} = \frac{\text{CongWin}}{\text{RTT}} \text{ byte/sec}$$

- CongWin è una funzione dinamica della congestione percepita

- Il mittente percepisce la congestione se
  - esaurimento timeout
  - ricezione di 3 ACK duplicati
- Il mittente TCP riduce la frequenza d'invio (CongWin) dopo un evento di perdita

Fasi della procedura

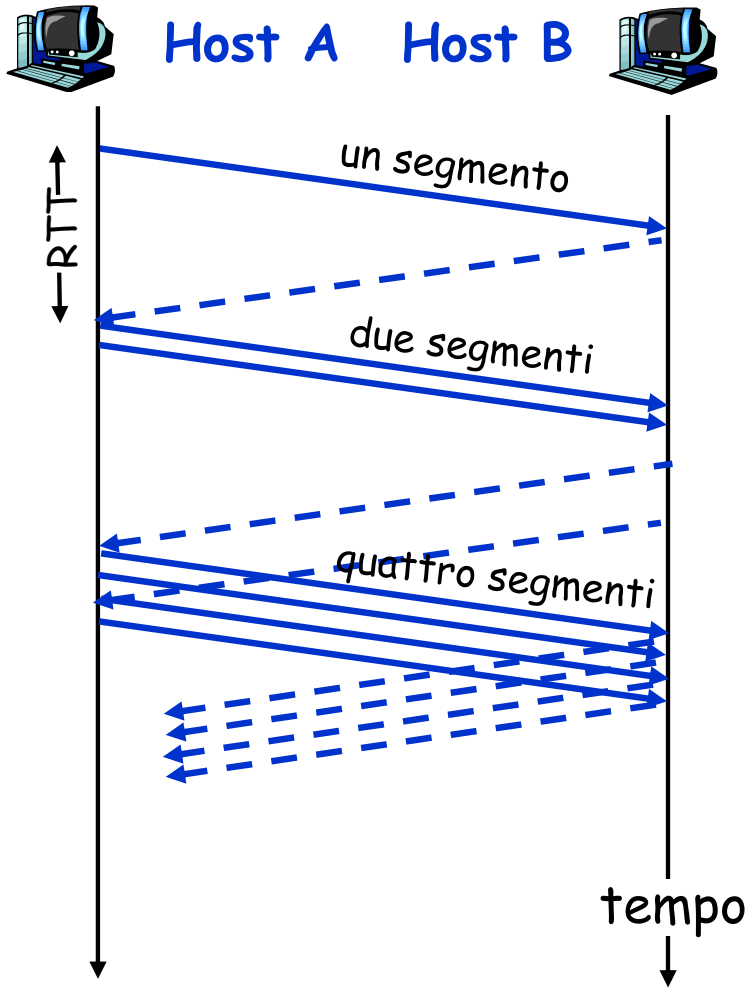
- **Per evitare la congestione, l'emettitore TCP segue una procedura ciclica in cui ogni ciclo è composto da due fasi**
  - Slow Start
    - Incremento esponenziale della Congwin
  - Congestion Avoidance
    - Incremento lineare della Congwin

Slow start

- **Quando si stabilisce una connessione**
  - CongWin = 1 MSS
  - Soglia = ∞
  - Esempio
    - MSS = 500 byte
    - RTT = 200 msec
  - Frequenza iniziale = 20 kbps
- **La larghezza di banda disponibile potrebbe essere >> MSS/RTT**
  - Consente di raggiungere rapidamente una bitrate d'invio significativa
- **Quando inizia la connessione, la frequenza aumenta in modo esponenziale, fino a quando non si verifica un evento di perdita**
- **Quando si verifica un evento di perdita si pone**
  - CongWin(new) = 1 MSS
  - Soglia = CongWin(old)/2

Slow start

- **Quando inizia la connessione, la frequenza aumenta in modo esponenziale, fino a quando non si verifica un evento di perdita**
  - raddoppia CongWin a ogni RTT
  - ciò avviene incrementando di un seg.MSS il valore della CongWin per ogni ACK ricevuto
- **L'incremento iniziale è lento, ma poi cresce in modo esponenziale**



Congestion Avoidance

- **Se l'aumento che si ha nella fase Slow Start raggiunge e supera il valore di soglia, e cioè se *Congwin* ≥ *Soglia*, l'incremento di *Congwin* diventa lineare al crescere di *RTT***
  - Se *Congwin* = *w* e se *w* ≥ *Soglia*, dopo l'arrivo di *w* riscontri consecutivi, la larghezza *Cwdn* viene incrementata di 1 seg.MSS in ciascun *RTT* in cui si registra l'arrivo di un intero gruppo di riscontri dei contenuti della finestra di congestione

Congestion Avoidance

- Questo incremento lineare continua finchè i riscontri arrivano prima dei loro rispettivi *RTO*
- Questo aumento ha un limite superiore corrispondente al raggiungimento di uno stato di saturazione su uno dei collegamenti lungo il percorso o in uno dei nodi attraversati
- Nell'ipotesi che *Congwin < Recwindow* , il limite superiore dell'aumento della *Congwin* è determinato dal verificarsi di un evento di perdita di un segmento e di un conseguente raggiungimento del relativo *RTO*

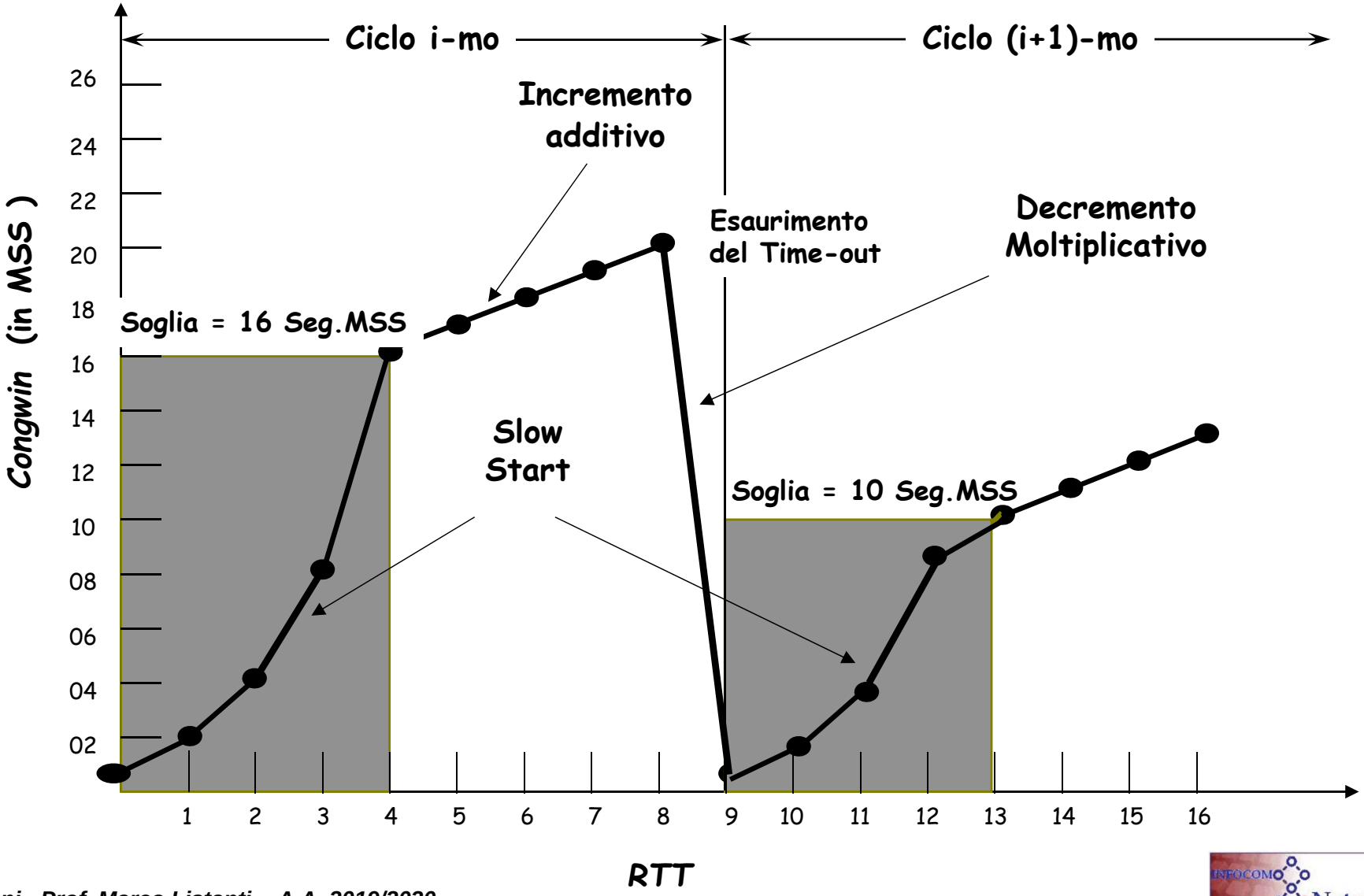
Decremento moltiplicativo

- **Quando si verifica l'esaurimento di un *RTO (evento di perdita di un segmento)*, inizia un nuovo ciclo**
- **Le operazioni effettuate sono le seguenti**
  - il valore *Soglia* viene impostato a metà del valore attuale di *Congwin* ed è quindi ridotto esponenzialmente rispetto a quello massimo raggiunto al termine della prima fase
  - il valore successivo di *Congwin* viene portato ad 1 MSS
  - l'incremento lineare continua finchè i riscontri arrivano prima dei loro rispettivi *RTO*

Procedura complessiva

- **In conclusione, se si trascura la fase di slow start, una entità TCP emittente**
  - incrementa *Cwdn* di 1 Seg.MSS per ogni *RTT* quando il suo percorso di rete non è congestionato
  - diminuisce *Cwdn* di un fattore 2 per ogni *RTT* quando il percorso è congestionato
- **Per questo motivo questa procedura di controllo di congestione è usualmente indicata come algoritmo di incremento additivo e di decremento moltiplicativo (AIMD, Additive-Increase, Multiplicative-Decrease)**

Procedura complessiva



Procedura complessiva

- **La procedura è illustrata nella figura seguente in cui**
  - il valore *Soglia* iniziale è uguale a 16 MSS
  - durante la fase di slow start, la Soglia è raggiunta all'istante 4
  - il valore di *Congwin* cresce poi linearmente, finchè non si verifica una perdita (istante 8) e quando *Congwin* = 20 MSS
  - il valore *Soglia* è allora ridotto a 0,5 *Congwin* = 10 MSS e la finestra di congestione è successivamente posta a 1 MSS
  - la fase di slow start ricomincia poi all'istante 9 e ha termine all'istante 13, quando *Congwin* ha raggiunto il valore 10 MSS
  - da quest'ultimo valore ricomincia l'incremento additivo di *Congwin* che avrà termine quando si verificherà una nuova perdita

Fast recovery

- **Ha lo scopo di ridurre l'impatto sull'efficienza in caso di perdita di singoli segmenti**
- **Dopo 3 ACK duplicati**
  - CongWin è ridotto a metà
  - la finestra poi cresce linearmente
- **Dopo un evento di esaurimento del timeout**
  - CongWin è impostata a 1 MSS
  - la finestra cresce in modo esponenziale fino a un valore di soglia, poi cresce linearmente
- **Spiegazione**
  - 3 ACK duplicati indicano che un segmento è stato perso, ma i successivi sono stati ricevuti dall'entità ricevente
  - un timeout prima di 3 ACK duplicati è invece un indizio "più allarmante" dello stato di congestione della rete

Riassunto

- Quando CongWin è sotto la soglia, il mittente è nella fase di slow start; la finestra cresce in modo esponenziale
- Quando CongWin è sopra la soglia, il mittente è nella fase di congestion avoidance; la finestra cresce in modo lineare
- Quando si verificano tre ACK duplicati, il valore di Soglia viene impostato a CongWin/2 e CongWin viene impostata al valore di Soglia
- Quando scade il timeout, il valore di Soglia viene impostato a CongWin/2 e CongWin è impostata a 1 MSS

Controllo di congestione del mittente TCP

Stato	Evento	Azione del mittente TCP	Commenti
Slow Start (SS)	Ricezione di ACK per dati precedente-mente non riscontrati	CongWin = CongWin + MSS, If (CongWin > Threshold) imposta lo stato a "Congestion Avoidance"	CongWin raddoppia a ogni RTT
Congestion Avoidance (CA)	Ricezione di ACK per dati precedente-mente non riscontrati	CongWin = = CongWin + MSS * (MSS/CongWin)	Incremento additivo: CongWin aumenta di 1 MSS a ogni RTT
SS o CA	Rilevato un evento di perdita da tre ACK duplicati	Threshold = CongWin/2, CongWin = Threshold, imposta lo stato a "Congestion Avoidance"	Ripristino rapido con il decremento moltiplicativo. CongWin non sarà mai minore di 1 MSS
SS o CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, imposta lo stato a "Slow Start"	Entra nello stato "Slow Start"
SS o CA	ACK duplicato	Incrementa il conteggio degli ACK duplicati per il segmento in corso di riscontro	CongWin e Threshold non variano

Throughput TCP

- **Qual è il throughput medio di TCP in funzione della dimensione della finestra e di RTT ?**
  - Ignoriamo le fasi di slow start
  - Sia W la dimensione della finestra quando si verifica una perdita
- **Quando la finestra è W, si ha**

$$TH1 = W / RTT$$

- **Subito dopo la perdita, la finestra si riduce a W/2, quindi**

$$TH2 = W/2*RTT$$

- **Poiché l'aumento è lineare**

$$TH = (TH1+TH2)/2 = 0,75\ W/RTT$$

TCP su "long, fat pipes"

- **Esempio:**
  - Segmenti di lunghezza 1500 byte
  - RTT = 100 ms
  - *C* = 10 Gbps

$$Cd = 100\ 10^{-3}\ 10\ 10^{9} = 10^{9} \qquad W(segmenti) = \frac{10^{9}}{12000} = 83{,}333$$

- **Per avere TH=1 si deve avere W=83,333 (segmenti)**

- **Il bit rate (in bit/s) in funzione della probabilità di perdita p di un segmento**

$$TH = \frac{1.22 \cdot MSS}{RTT \cdot \sqrt{\pi}}$$

- **Ponendo TH=1, se si vuole ottenere un bit rate uguale a 10 Gbit/s si deve avere una probabilità di perdita $p \leq 2 \cdot 10^{-10}$**
- **Per collegamenti a larga banda sono necessarie altre versioni del TCP**