Università di Bergamo

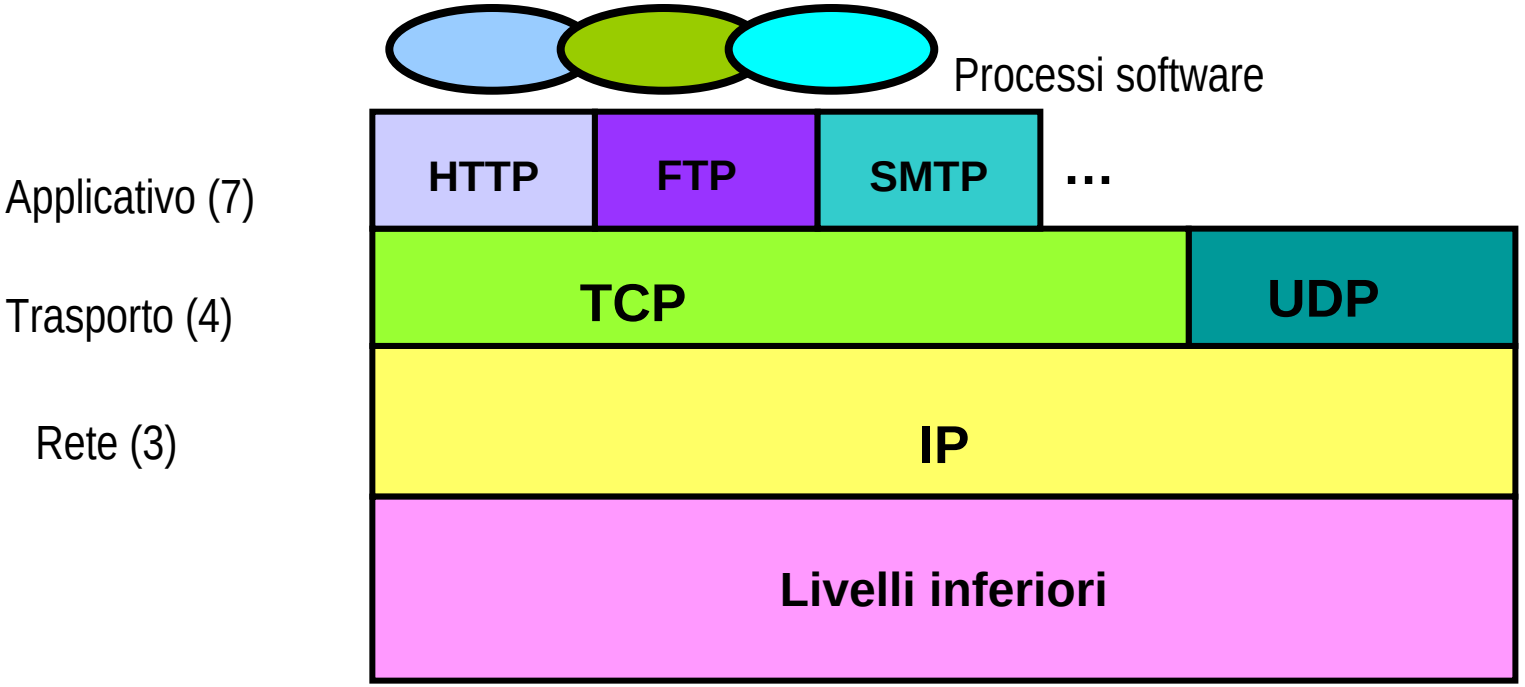
Dipartimento di Ingegneria dell'Informazione e Metodi Matematici

4 - Il livello di trasporto

Architetture e Protocolli per Internet

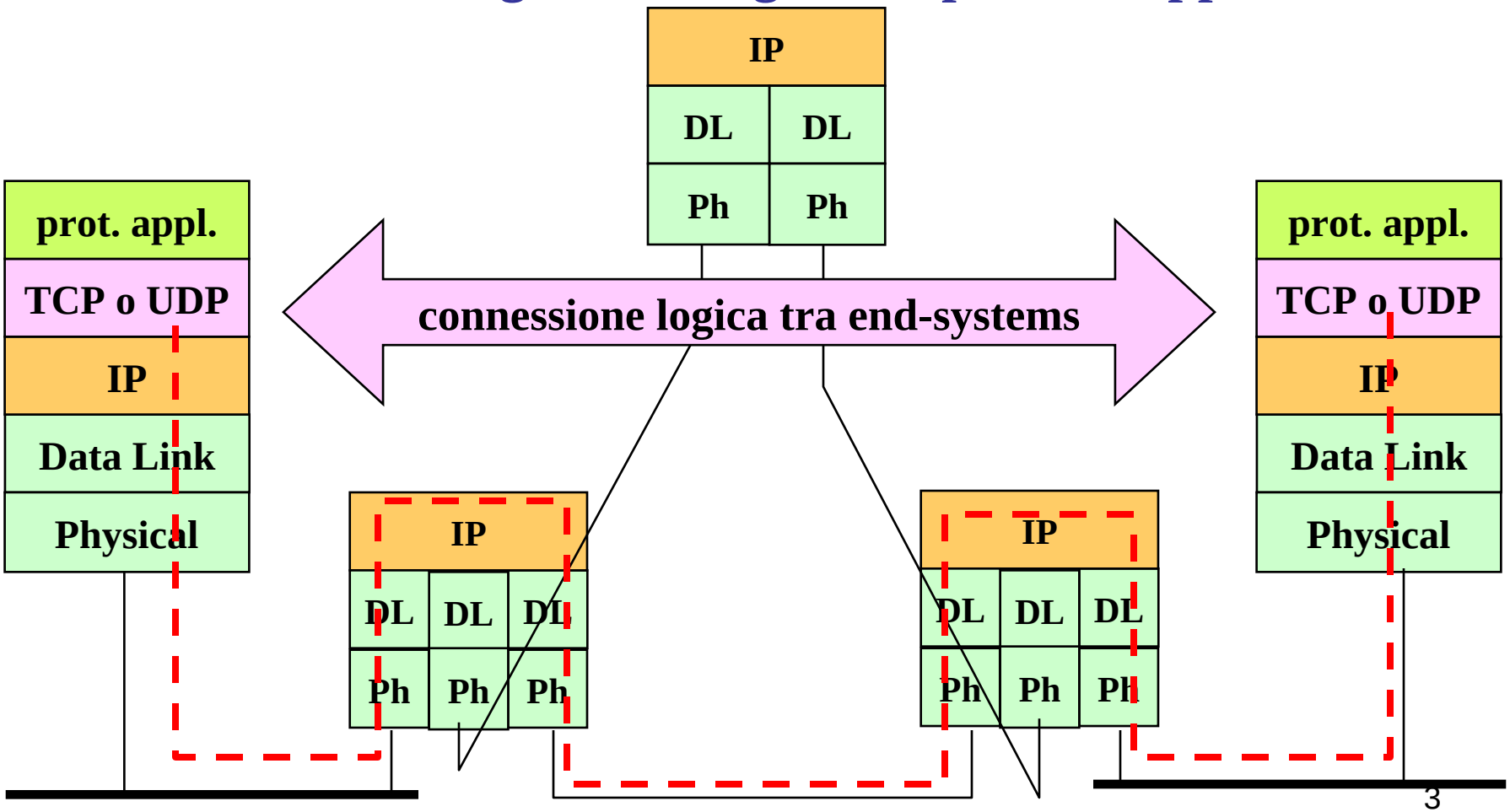
Servizio di trasporto

- il livello di trasporto ha il compito di instaurare un collegamento logico tra le applicazioni residenti su host remoti
- Nella suite TCP/IP, i protocolli applicativi si appoggiano direttamente sul livello di trasporto



Servizio di trasporto

- il livello di trasporto è presente solo negli end-systems (hosts)
- esso consente il collegamento logico tra processi applicativi

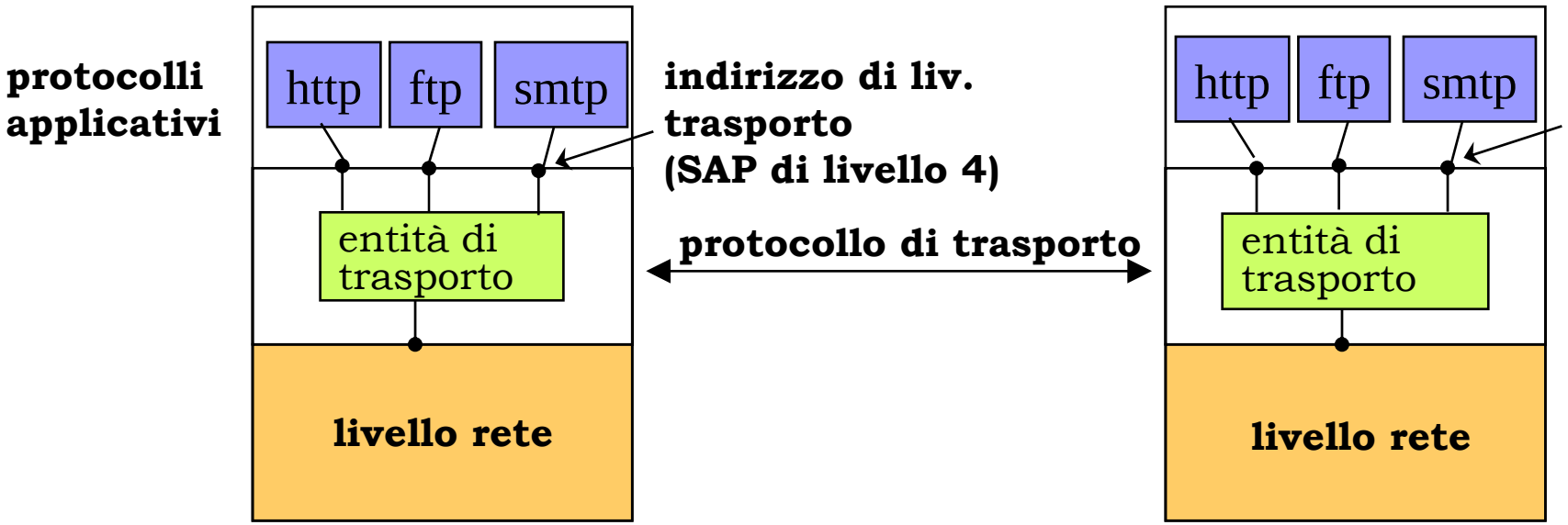


Servizio di trasporto

- **Da applicazione ad applicazione:**
  - **i messaggi di un'applicazione vengono segmentati e trasformati in PDU (Protocol Data Unit) di livello 4 (trasporto), detti *segmenti*.**
  - **il livello di trasporto passa le 4-PDU al livello di rete che le incapsula in PDU di livello 3 e le inoltra in rete**
  - **a destinazione i messaggi passano dal livello 3 al 4 e i messaggi dell'applicazione vengono ricostruiti**
- **il livello di trasporto rende trasparente il trasporto fisico dei messaggi alle applicazioni**

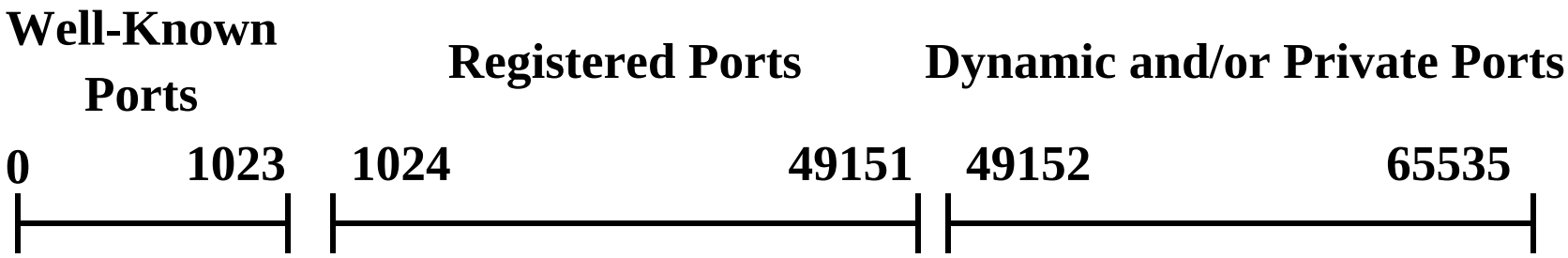
Servizio di trasporto

- Più applicazioni possono essere attive su un end-system
  - il livello di trasporto svolge funzioni di multiplexing/demultiplexing
  - ciascun collegamento logico tra applicazioni è indirizzato dal livello di trasporto



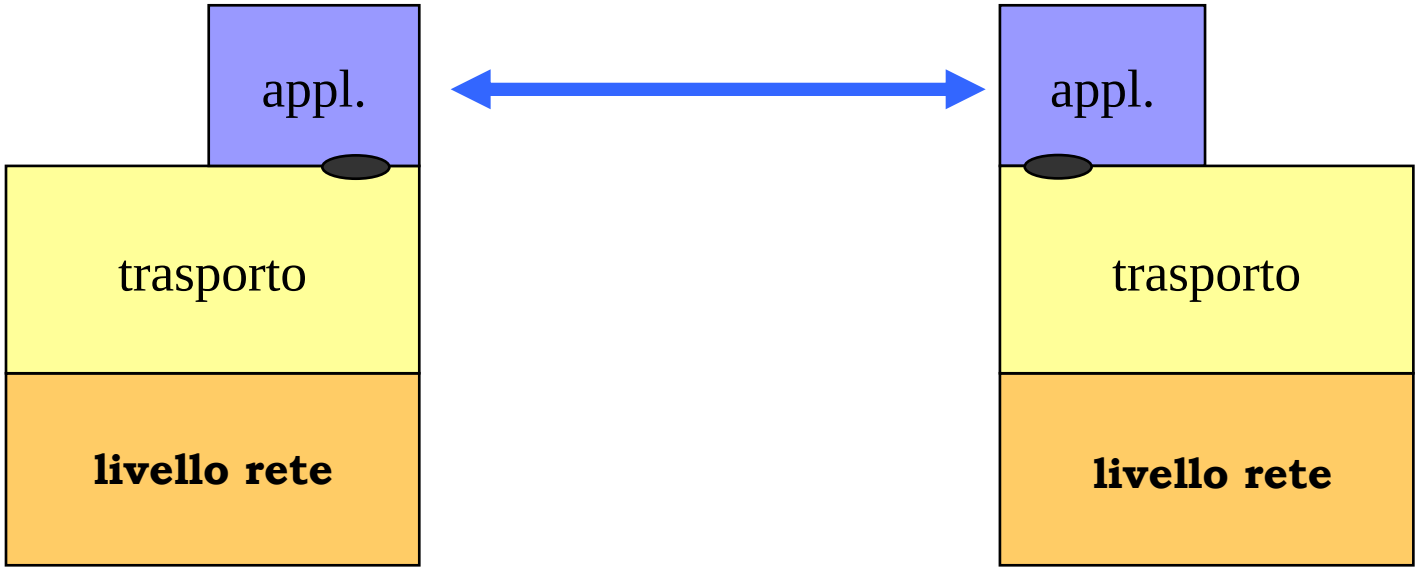
Le porte

- In Internet le funzioni di multiplexing/demultiplexing vengono gestite mediante indirizzi contenuti nelle PDU di livello trasporto
- tali indirizzi sono lunghi 16 bit e prendono il nome di *porte*
- i numeri di porta possono assumere valori compresi tra 0 e 65535
- i numeri noti sono assegnati ad applicativi importanti dal lato del server (HTTP, FTP, SMTP, DNS, ecc.)
- http://www.iana.org/assignments/port-numbers
- i numeri dinamici sono assegnati dinamicamente ai processi applicativi lato client



Socket

- **Il numero di porta e l'indirizzo IP identifica in modo univoco un processo applicativo (client o server) in esecuzione su un host**
- **la coppia di indirizzi prende il nome di indirizzo di *socket***
- **i socket dei due processi in colloquio sono sempre contenuti negli header di livello IP e trasporto**

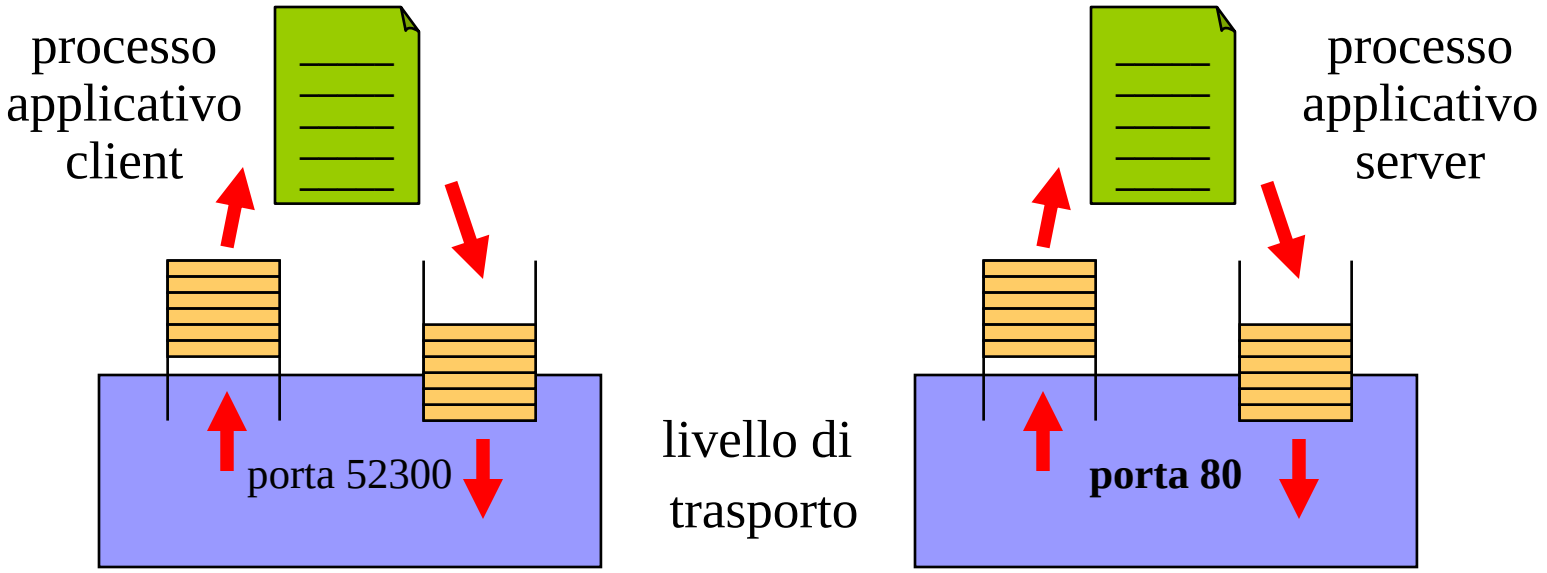


Servizio di trasporto

- Il servizio di trasporto fornito può essere di vari tipi
  - trasporto affidabile (garanzia di consegna dei messaggi nel corretto ordine)
  - trasporto non affidabile (solo funzionalità di indirizzamento)
  - ma ovviamente il servizio realmente fornito all'applicazione dipende dal livello rete sottostante
- Nella suite IP sono definiti due tipi di trasporto
  - TCP (Transmission Control Protocol), orientato alla connessione e affidabile
  - UDP (User Datagram Protocol), senza connessione e non affidabile

Servizio di trasporto

- I protocolli di trasporto sono implementati nei più diffusi sistemi operativi
- i sistemi operativi forniscono ai programmatori le funzioni di base per poter usare i protocolli di trasporto e far comunicare processi remoti
- quando un processo viene associato ad una porta (lato client o lato server) viene associato dal sistema operativo a due code, una d'ingresso e una d'uscita



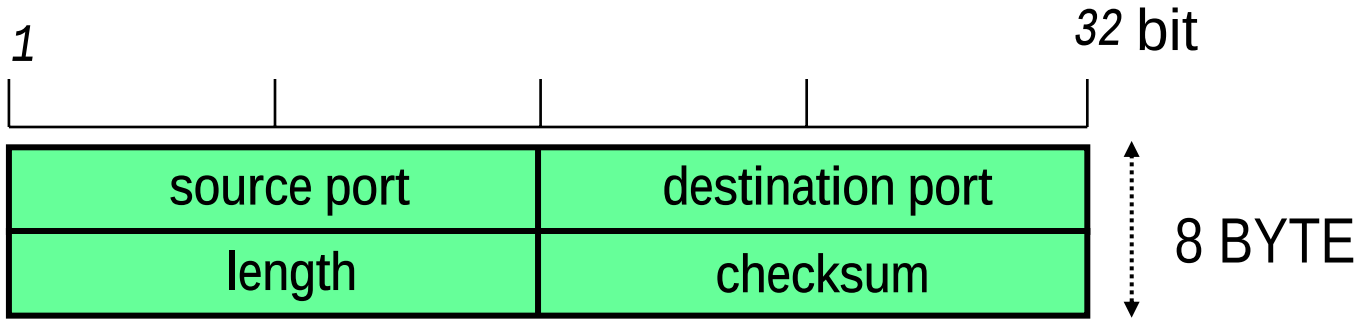
Applicazioni e trasporto

- **In base al tipo di applicazione viene scelto il tipo di protocollo di trasporto più adatto**

Applicazione	Protocollo applicativo	Protocollo di trasporto
electronic mail	*SMTP*	*TCP*
remote terminal access	*Telnet*	*TCP*
Web	*HTTP*	*TCP*
file transfer	*FTP*	*TCP*
remote file server	*NFS*	*typically UDP*
streaming multimedia	*proprietary*	*typically UDP*
Internet telephony	*proprietary*	*typically UDP*
Network Management	*SNMP*	*typically UDP*
Routing Protocol	*RIP*	*typically UDP*
Name Translation	*DNS*	*typically UDP*

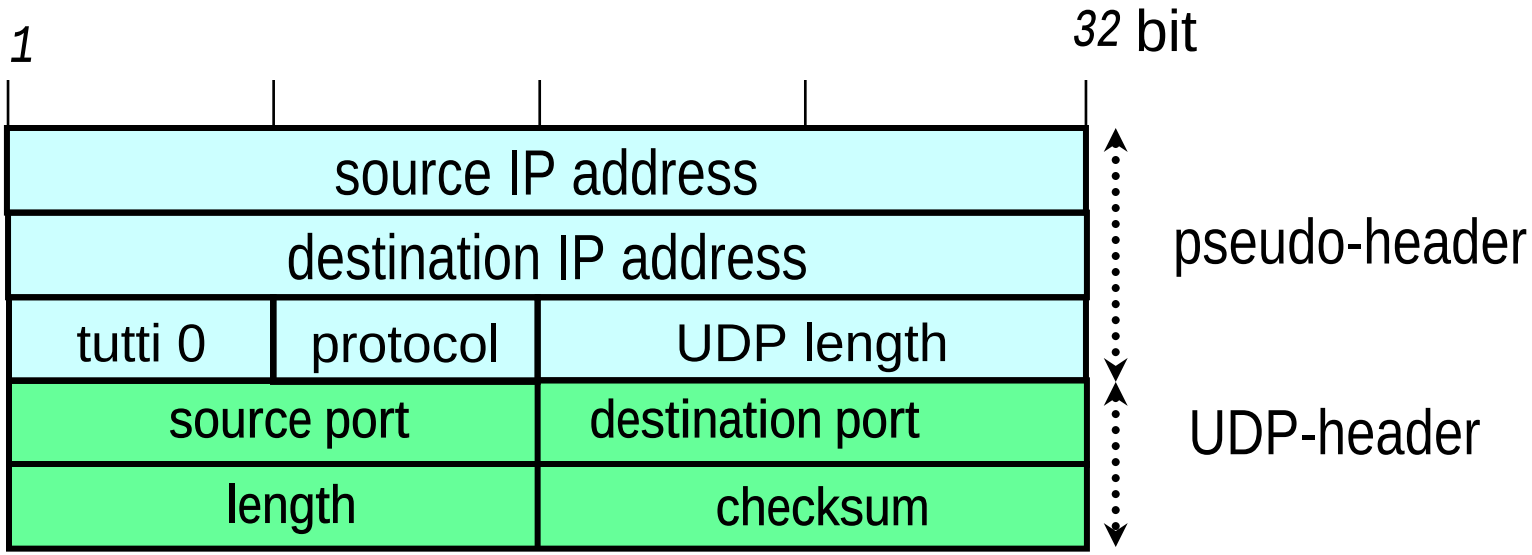
User Datagram Protocol (UDP)

- E’ il modo più semplice di usare le funzionalità di IP
- Non aggiunge nulla a IP se non:
  - indirizzamento delle applicazioni
  - blando controllo d’errore sull’header
- … e quindi
  - è un protocollo datagram
  - non garantisce la consegna
  - non esercita nessun controllo (né di flusso, né di errore)



UDP: checksum

- **Length include header e dati (length minima = 8 byte)**
- **Il checksum si calcola in modo analogo a quello del checksum IP…**
- **…ma non viene calcolato solo considerando l'header UDP, bensì anche uno pseudo-header IP**

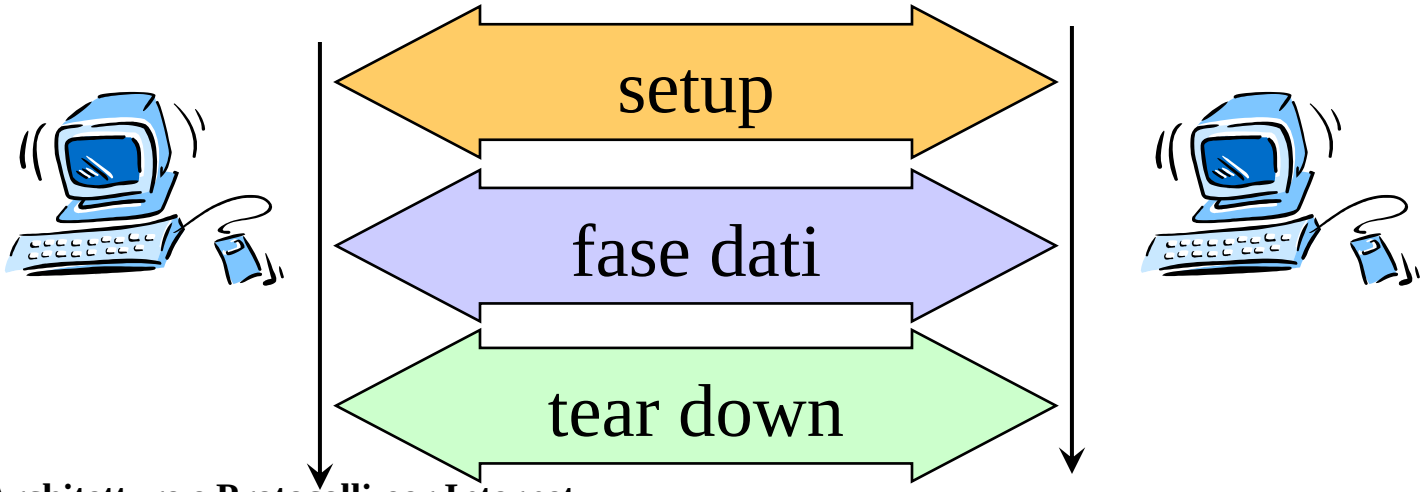


Transmission Control Protocol (TCP)

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

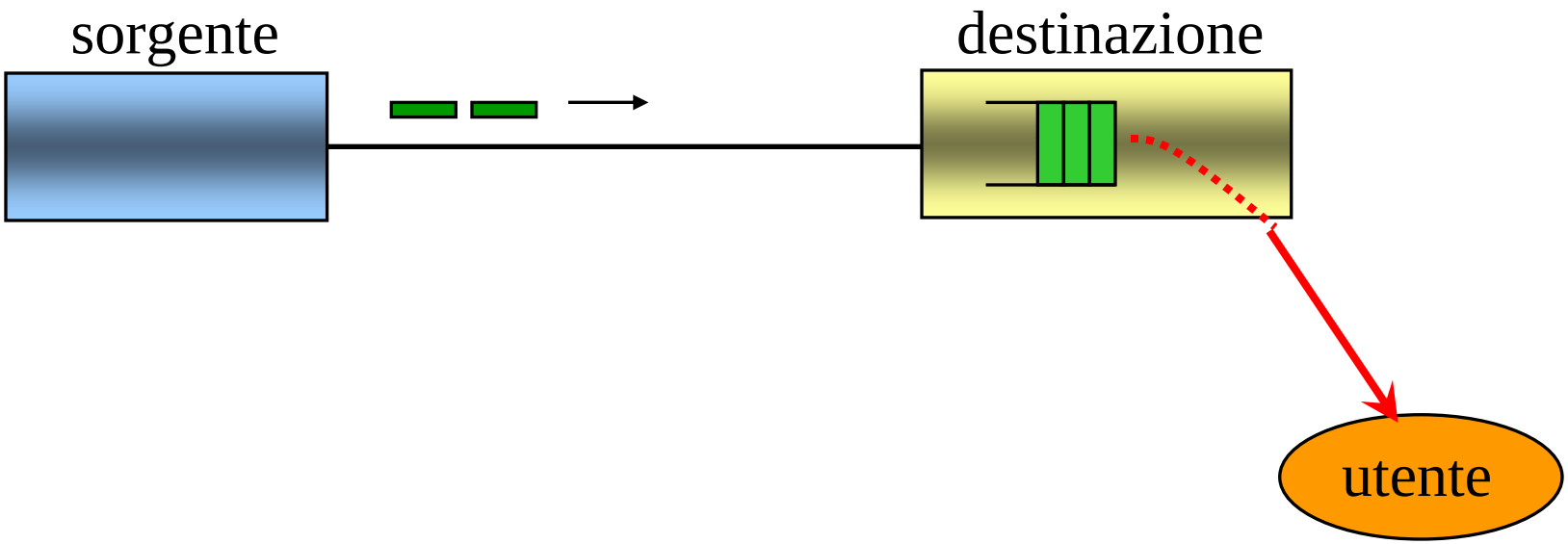
TCP: connection oriented

- **Il TCP è orientato alla connessione (connection oriented):**
  - **prima del trasferimento di un flusso dati occorre instaurare una connessione mediante opportuna segnalazione**
  - **le connessioni TCP si appoggiano su una rete connectionless (datagram)**
  - **le connessioni TCP possono essere solo di tipo *full-duplex* (esiste sempre un flusso di dati in un verso e nel verso opposto, anche se questi possono essere quantitativamente diversi)**
  - **per questo motivo, TCP è adottato in combinazione con protocolli a livello rete di tipo datagram, come per esempio IP**



TCP: controllo di flusso

- **Il TCP usa un controllo di flusso:**
  - **il flusso dei dati in ingresso in rete è regolato sulla base della capacità del ricevitore di riceverli**
  - **il controllo è basato su una sliding window**

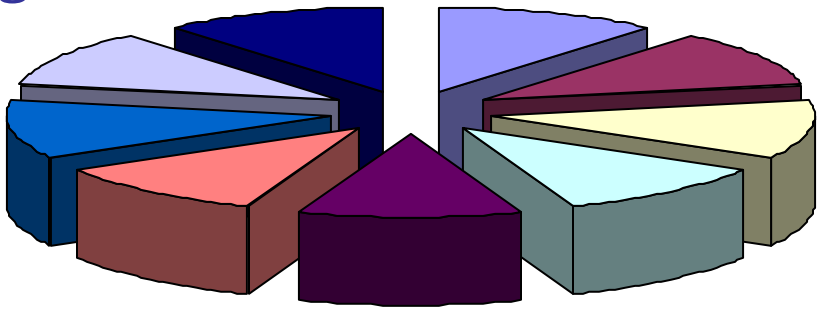


TCP: controllo di congestione

- Il TCP ha dei meccanismi di controllo della congestione
  - il flusso dei dati in ingresso in rete è anche regolato dalla situazione di traffico in rete
  - se il traffico in rete porta a situazioni di congestione il TCP riduce velocemente il traffico in ingresso
  - in rete non vi è nessun meccanismo per notificare esplicitamente le situazioni di congestione
  - il TCP cerca di scoprire i problemi di congestione sulla base degli eventi di perdita dei pacchetti

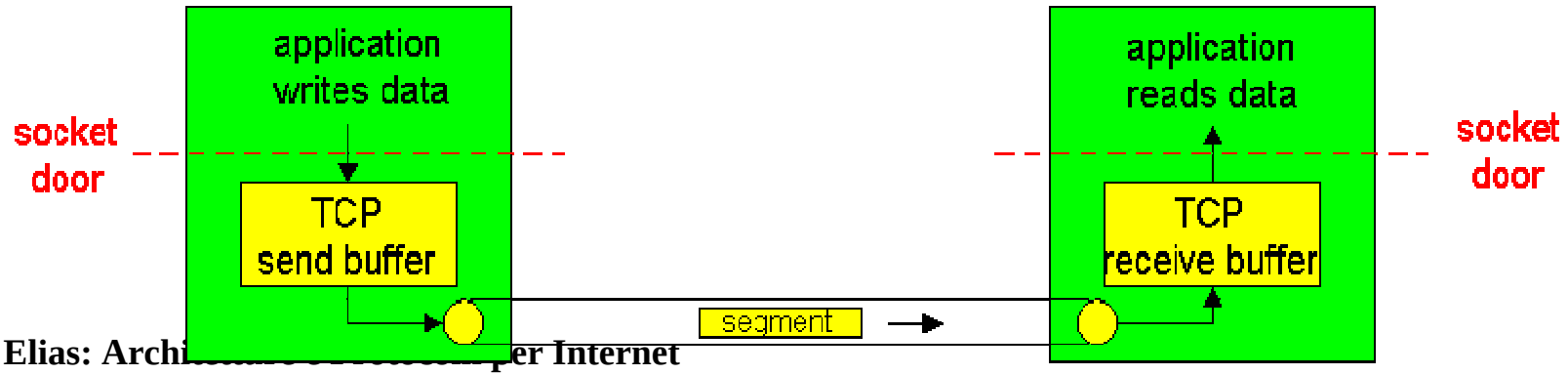
TCP: controllo di congestione

- il meccanismo si basa ancora sulla sliding window la cui larghezza viene dinamicamente regolata in base alle condizioni in rete
- in linea di principio scopo del controllo è far si che il flusso emesso da ciascuna sorgente venga regolato in modo tale che il flusso complessivo offerto a ciascun canale non superi la sua capacità
- tutti i flussi possono essere ridotti in modo tale che la capacità della rete venga condivisa da tutti in misura se possibile uguale



TCP: flusso dati

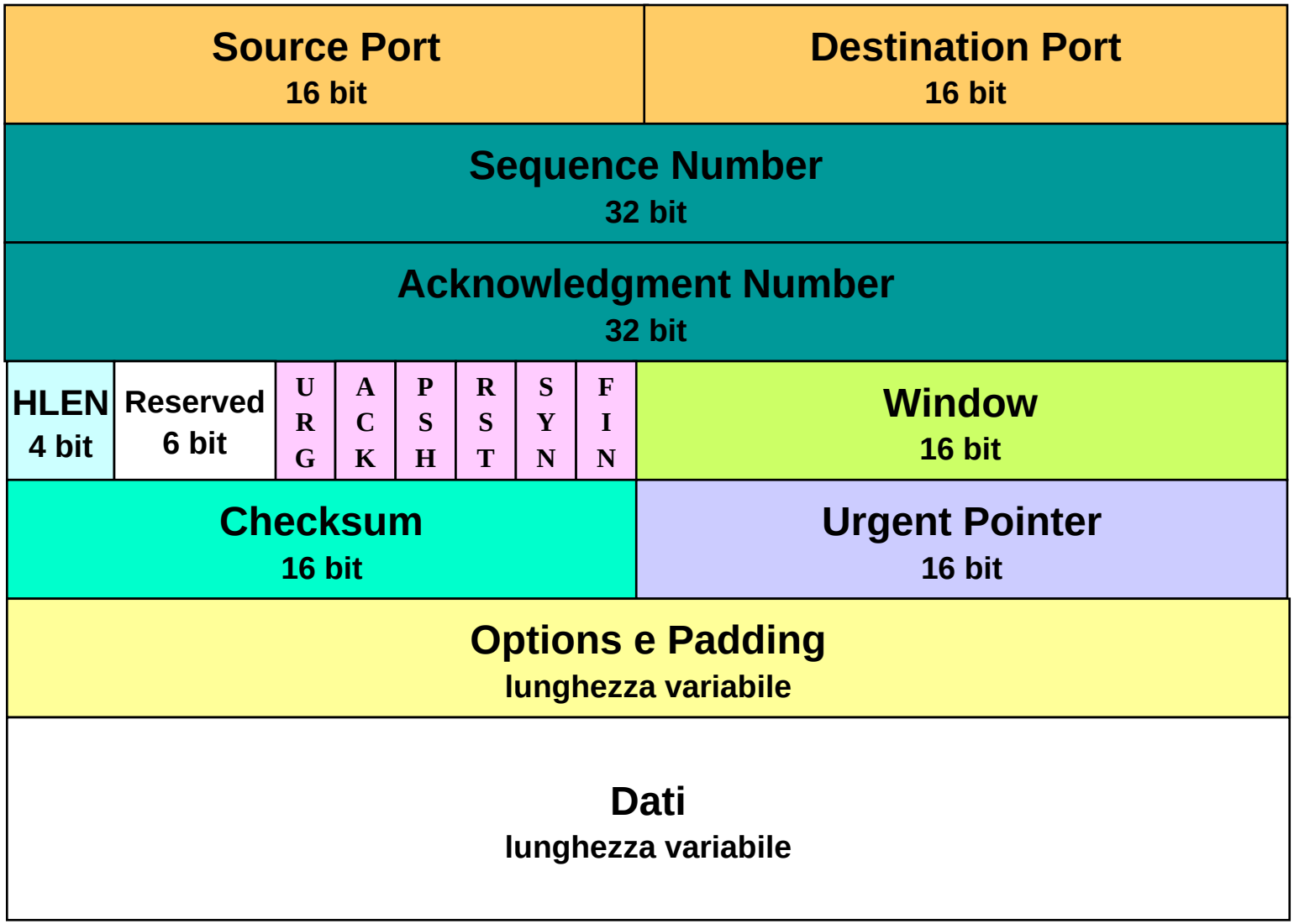
- Il TCP è orientato alla trasmissione di flussi continui di dati (stream di byte)
- il TCP converte il flusso di dati in *segmenti* che possono essere trasmessi in IP
- le dimensioni dei segmenti sono variabili
- l'applicazione trasmittente passa i dati a TCP e TCP accumula i dati in un buffer.
- periodicamente,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



TCP: controllo d'errore

- Per assicurare il trasferimento affidabile del flusso dati su una rete che non garantisce affidabilità il TCP adotta un meccanismo per il controllo delle perdite di pacchetti di tipo go-back-n
- sistema di numerazione e di riscontro dei dati inviati
  - TCP numera ogni byte trasmesso, per cui ogni byte ha un numero di sequenza
  - nell'header del segmento TCP è trasportato il numero di sequenza del primo byte nel segmento stesso
  - il ricevitore deve riscontrare i dati ricevuti inviando il numero di sequenza dell'ultimo byte ricevuto correttamente ed in sequenza + 1 (*next expected byte*)
  - se un riscontro non arriva entro un dato timeout, i dati sono ritrasmessi

Segmento TCP



Header TCP

- Source port, Destination port: **indirizzi di porta sorgente e porta destinazione di 16 bit**
- Sequence Number**: il numero di sequenza del primo byte nel payload**
- Acknowledgement Number: **numero di sequenza del prossimo byte che si intende ricevere (numero valido solo se il bit ACK è impostato ad 1)**
- HLEN: **contiene la lunghezza complessiva dell'header TCP, che DEVE essere un multiplo intero di 32 bit**
- Window: **contiene il valore della finestra di ricezione come comunicato dal ricevitore al trasmettitore**
- Checksum: **CRC calcolato su un header virtuale ottenuto aggiungendo gli indirizzi IP di sorgente e di destinazione**

Header TCP

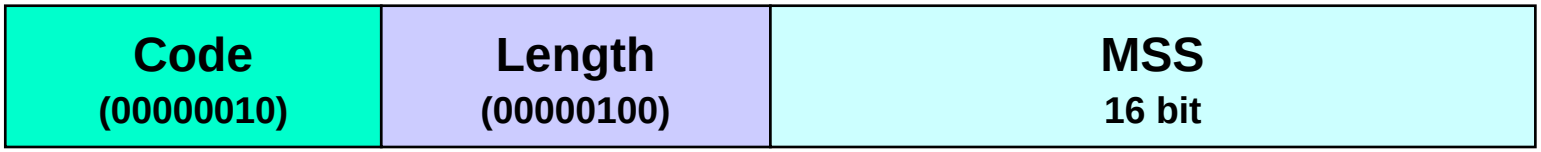
- **Flag:**
  - URG**: vale uno se vi sono dati ugenti; in questo caso urgent pointer punta al primo byte dei dati urgenti all'interno dei dati**
  - ACK**: vale uno se il pacchetto è un ACK valido; in questo caso l'acknowledgement number contiene un numero valido**
  - PSH: **vale uno quando il trasmettitore intende usare il comando di PUSH; il ricevitore può anche ignorare il comando (dipende dalle implementazioni)**
  - RST: **reset; resetta la connessione senza un tear down esplicito**
  - SYN: **synchronize; usato durante il setup per comunicare i numeri di sequenza iniziali**
  - FIN: **usato per la chiusura esplicita di una connessione**
- Options and Padding: **riempimento (fino a multipli di 32 bit) e campi opzionali come ad esempio durante il setup per comunicare il MSS (il valore di default è 536 byte)**

Opzioni

- Delle opzioni possono essere aggiunte all'header TCP
- Opzioni di 1 byte:
  - **no operation:** 00000001 (viene usata talora come riempimento per avere un header multiplo di 32 bit)
  - **end of option:** 00000000 (byte di riempimento finale, usato alla fine di tutte le opzioni)
- Opzioni lunghe:
  - maximum segment size
  - fattore di scala della finestra
  - timestamp

Opzioni: Maximum Segment Size (MSS)

- Definisce la dimensione massima del segmento che verrà usata nella connessione TCP
- La dimensione è decisa dal mittente (TCP sender) durante la fase di setup
- valore di default è 536 byte, il valore massimo 65535 byte



Kind=2 4 byte

Opzioni: Fattore di scala della finestra (TCP Window Scale Option)

- Definisce l'unità di misura della finestra (campo window dell'header TCP)
- Il valore di default è 1 byte
- con l'opzione il valore viene modificato di un fattore pari a 2 elevato al valore contenuto nel campo *fattore di scala*

Code (00000011)	Length (00000011)	Fattore di scala 8 bit

Kind=3 3 byte

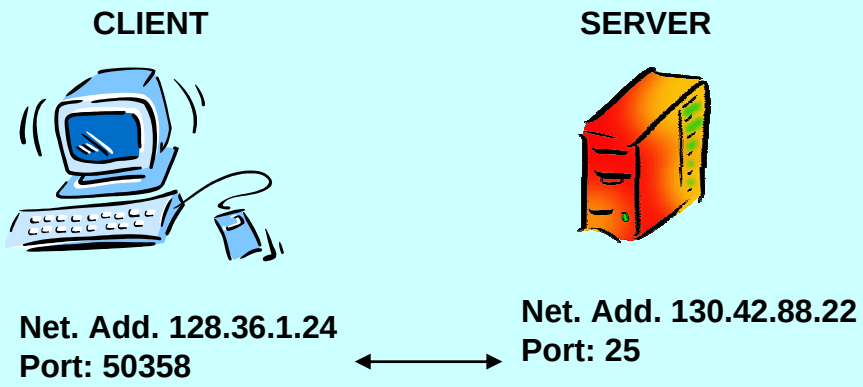
Servizi e porte

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

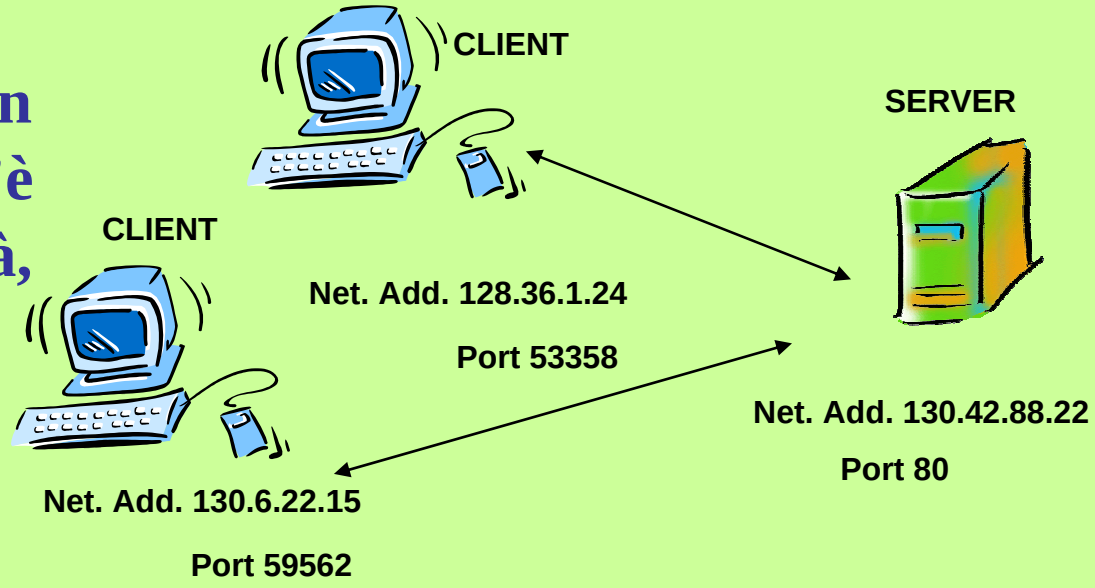
21	FTP signaling
20	FTP data
23	Telnet
25	SMTP
53	DNS
80	HTTP

Socket e connessioni

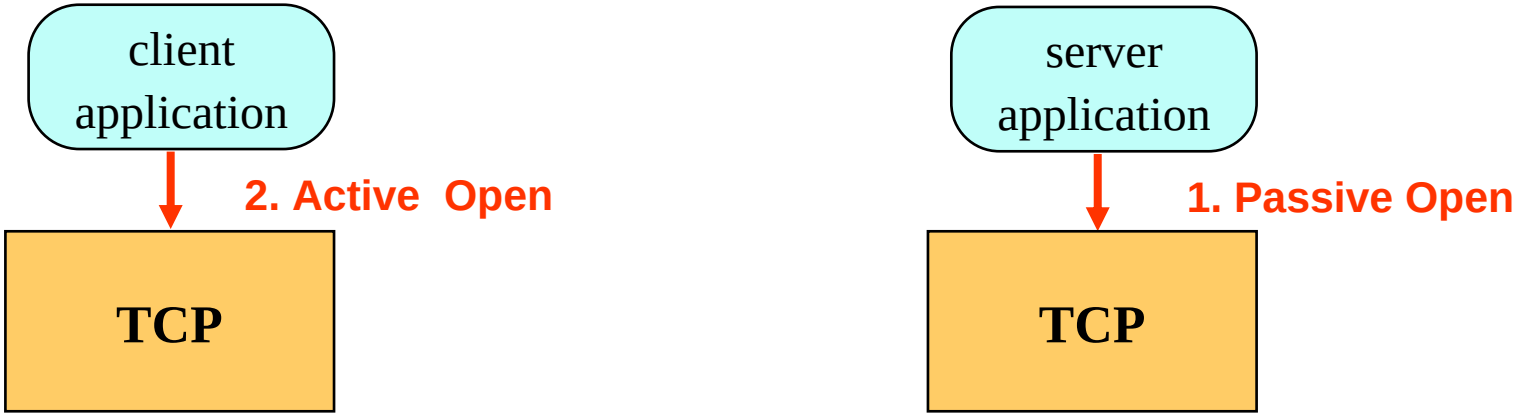
- **Un client si connette alla porta di un server SMTP remoto (servizio di posta elettronica)**



- **Due client accedono alla stessa porta di un server HTTP; non c'è comunque ambiguità, perché la coppia di socket è diversa**

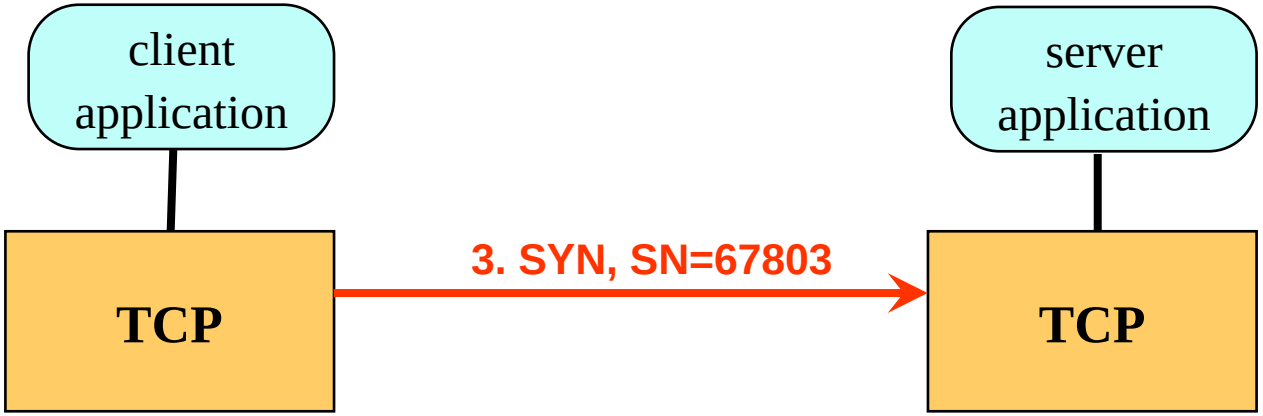


Setup (apertura) delle connessioni



- Prima del call setup le applicazioni dal lato client e dal lato server devono comunicare con il software TCP
  - 1. Il server fa una *Passive Open*, che comunica al TCP locale che è pronto per accettare nuove connessioni
  - 2. Il client che desidera comunicare fa una *Active Open*, che comunica al TCP locale che l'applicativo intende effettuare una connessione verso un dato socket

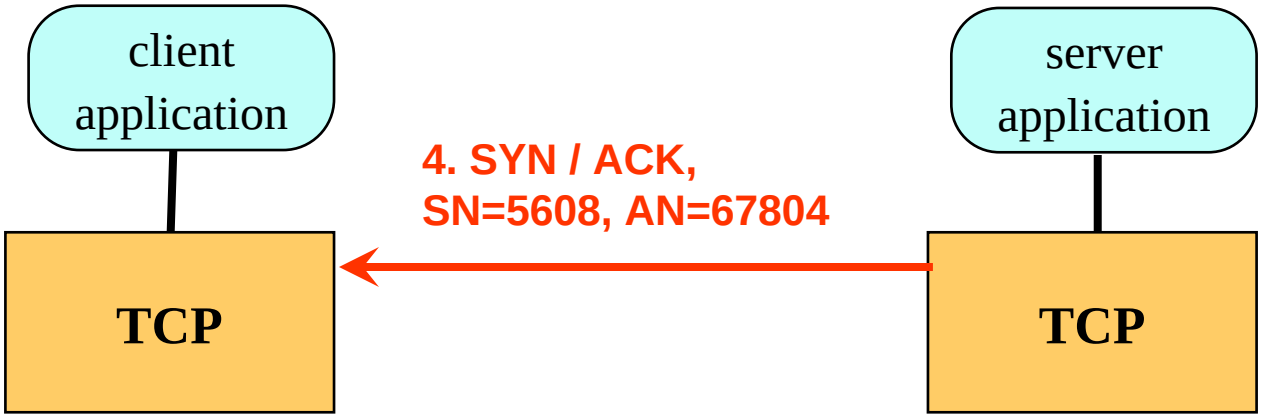
Setup delle connessioni



- **3. Il client TCP estrae a caso un numero di sequenza iniziale (ad es. 67803) e manda un messaggio di SYNchronize (flag SYN=1) contenente questo numero di sequenza**

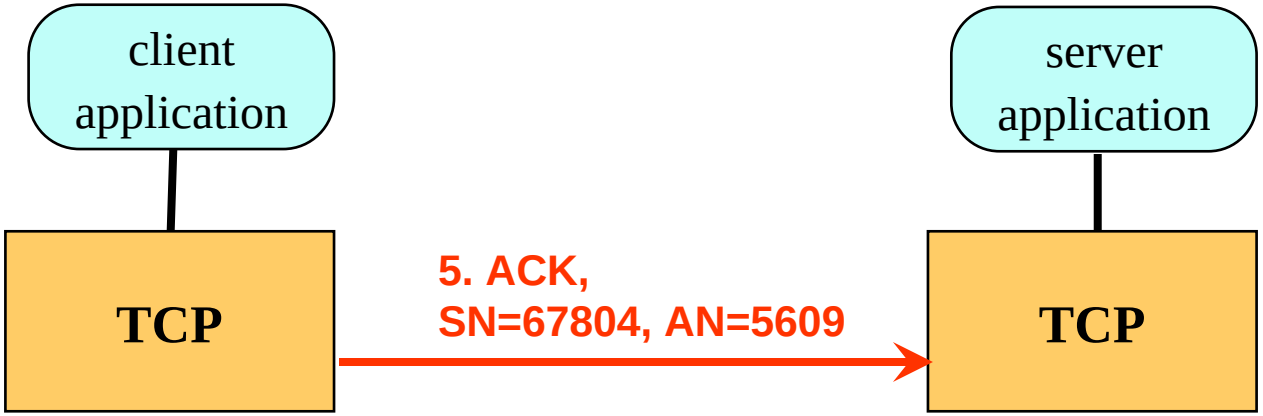
- ***L'estrazione del numero iniziale serve a evitare problemi nel caso in cui il setup non va a buon fine a causa della perdita di pacchetti e un nuovo setup viene iniziato subito dopo***

Setup delle connessioni



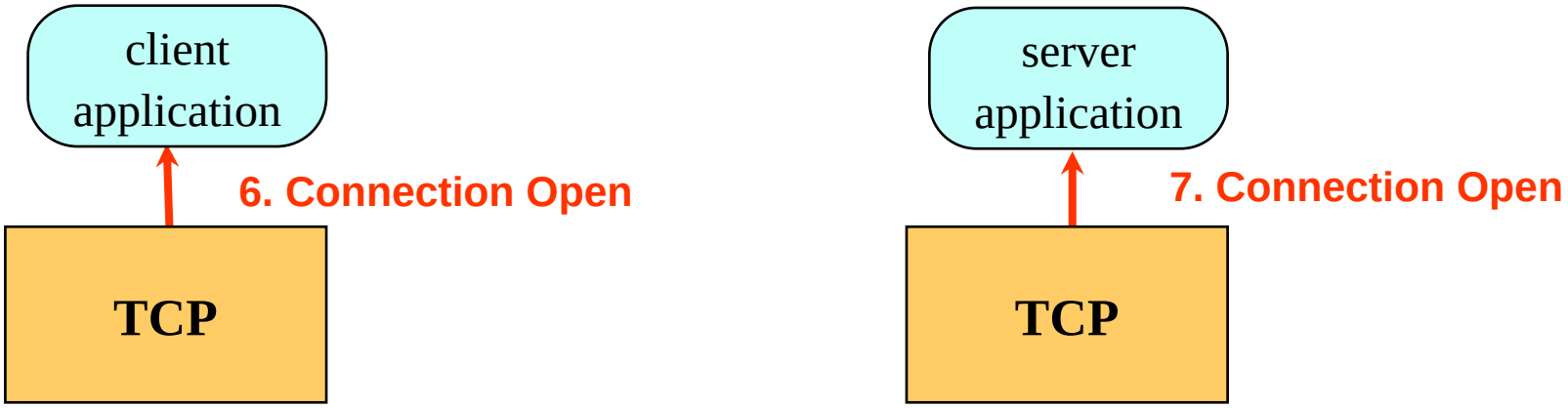
- **4. Quando riceve il SYN, il TCP server estrae a caso un numero di sequenza iniziale (ad es. 5608) e manda un segmento SYN/ACK (flag SYN=1, flag ACK=1) contenente anche un acknowledgment number uguale a 67804, per riscontrare il numero di sequenza iniziale precedentemente inviato dal TCP client.**

Setup delle connessioni



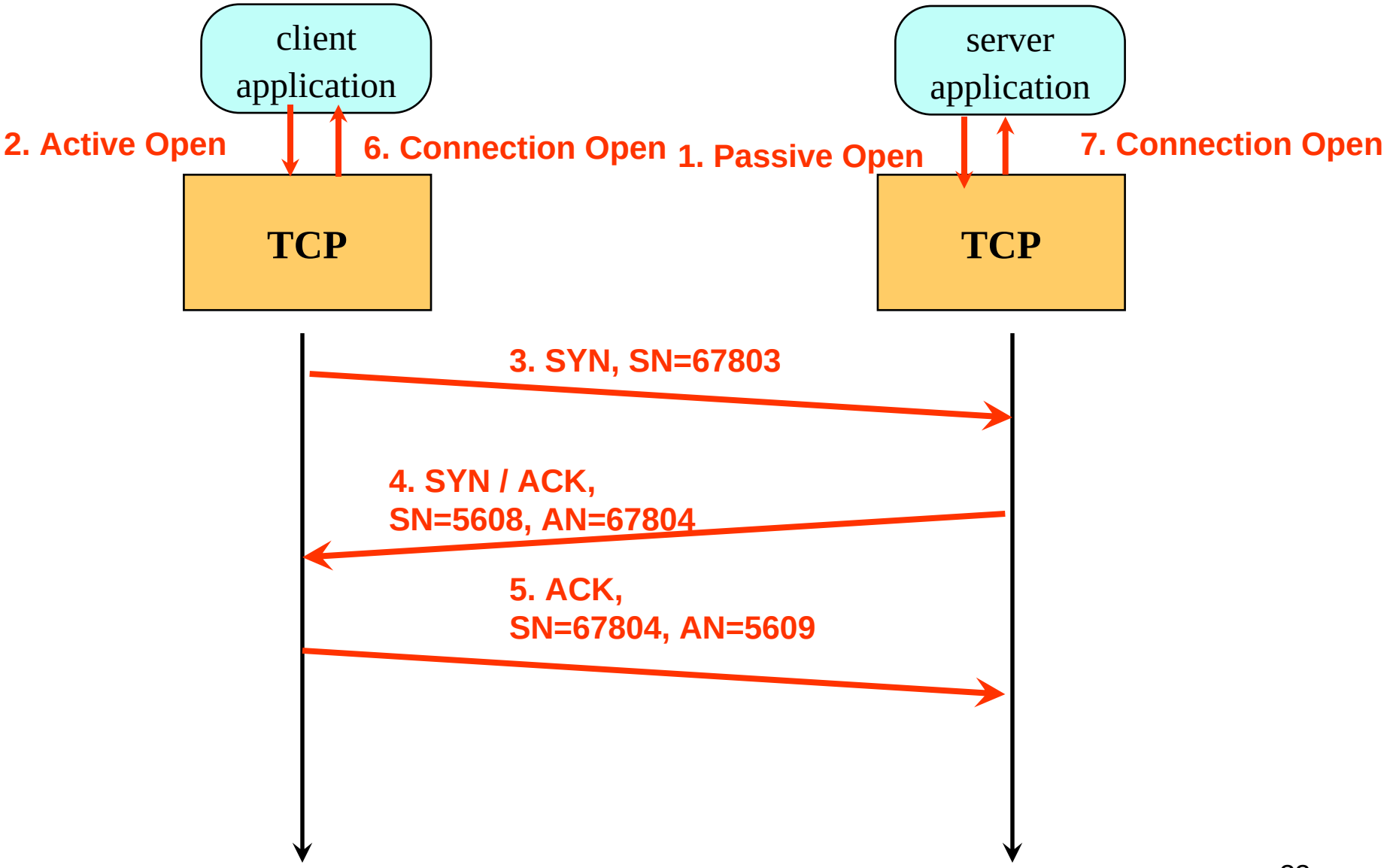
- **5. Il TCP client riceve il messaggio SYN/ACK del server, e invia un ACK per il 5609. Nel payload inserisce i primi dati della connessione con numero di sequenza del primo byte pari a 67804.**

Setup delle connessioni

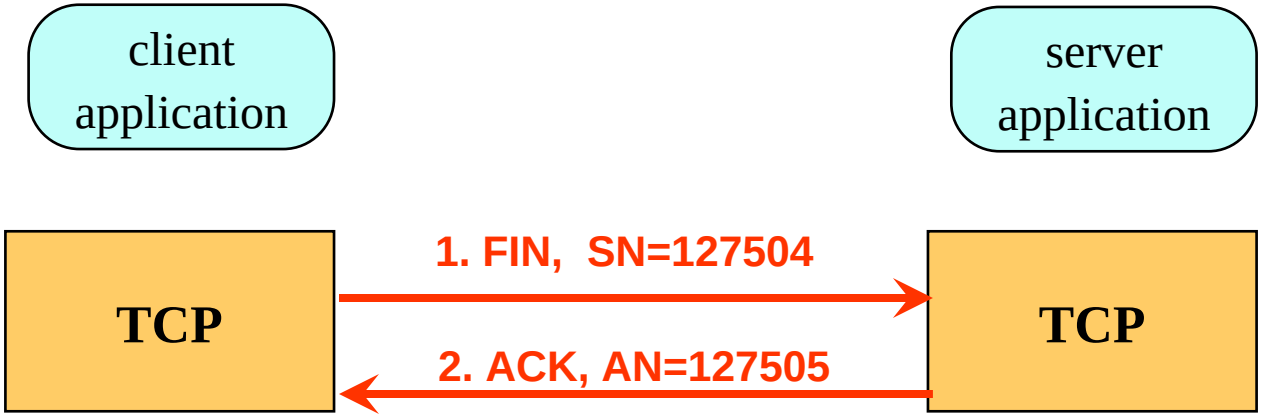


- **6. Il TCP client notifica all'applicazione che la connessione è aperta**
- **7. Quando il TCP server riceve l'ACK del TCP client, notifica all'applicazione che la connessione è aperta**

Setup delle connessioni

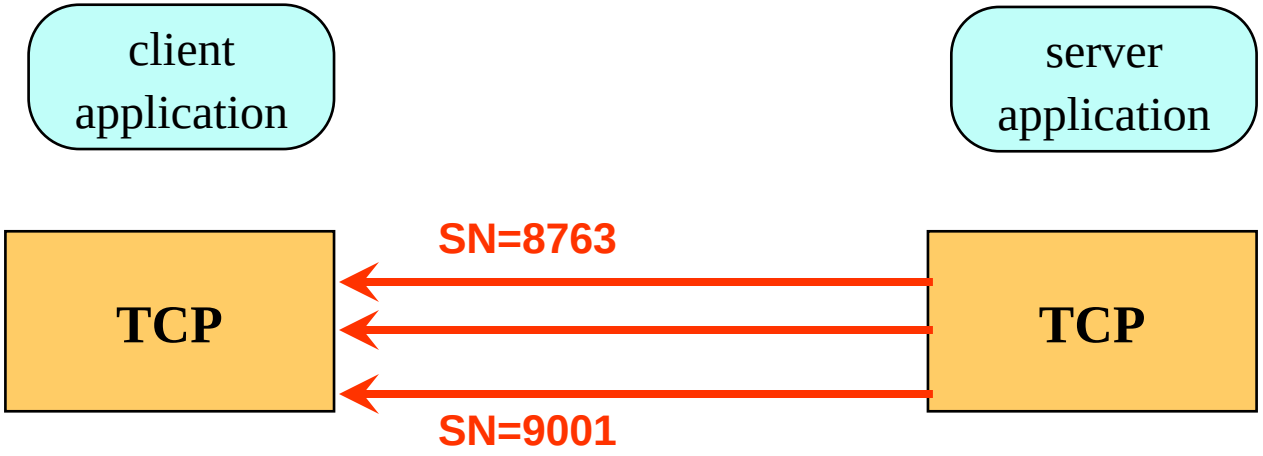


Tear down (chiusura) delle connessioni



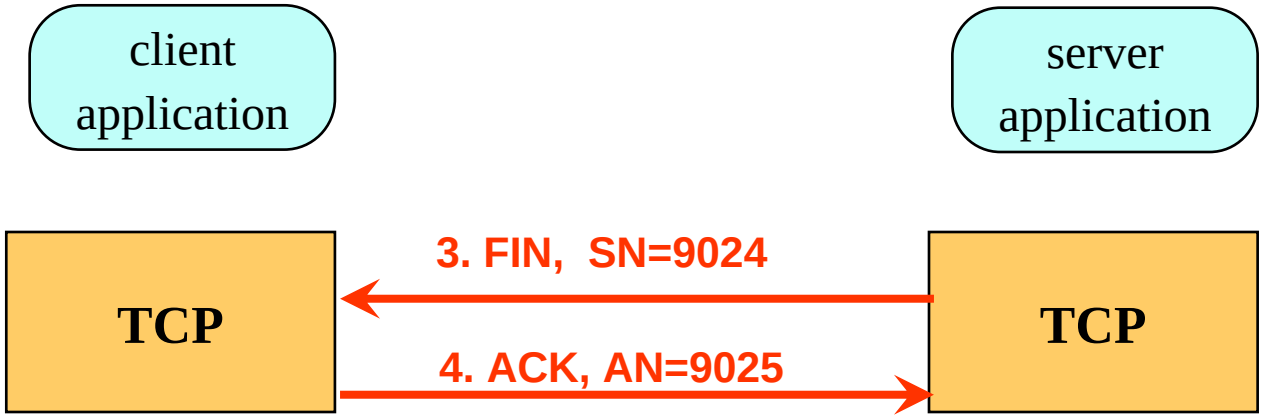
- **1. Il TCP che chiude la connessione invia un messaggio di FIN (flag FIN=1) con gli ultimi dati**
- **2. Il TCP dall'altra parte invia un ACK per confermare**

Tear down (chiusura) delle connessioni



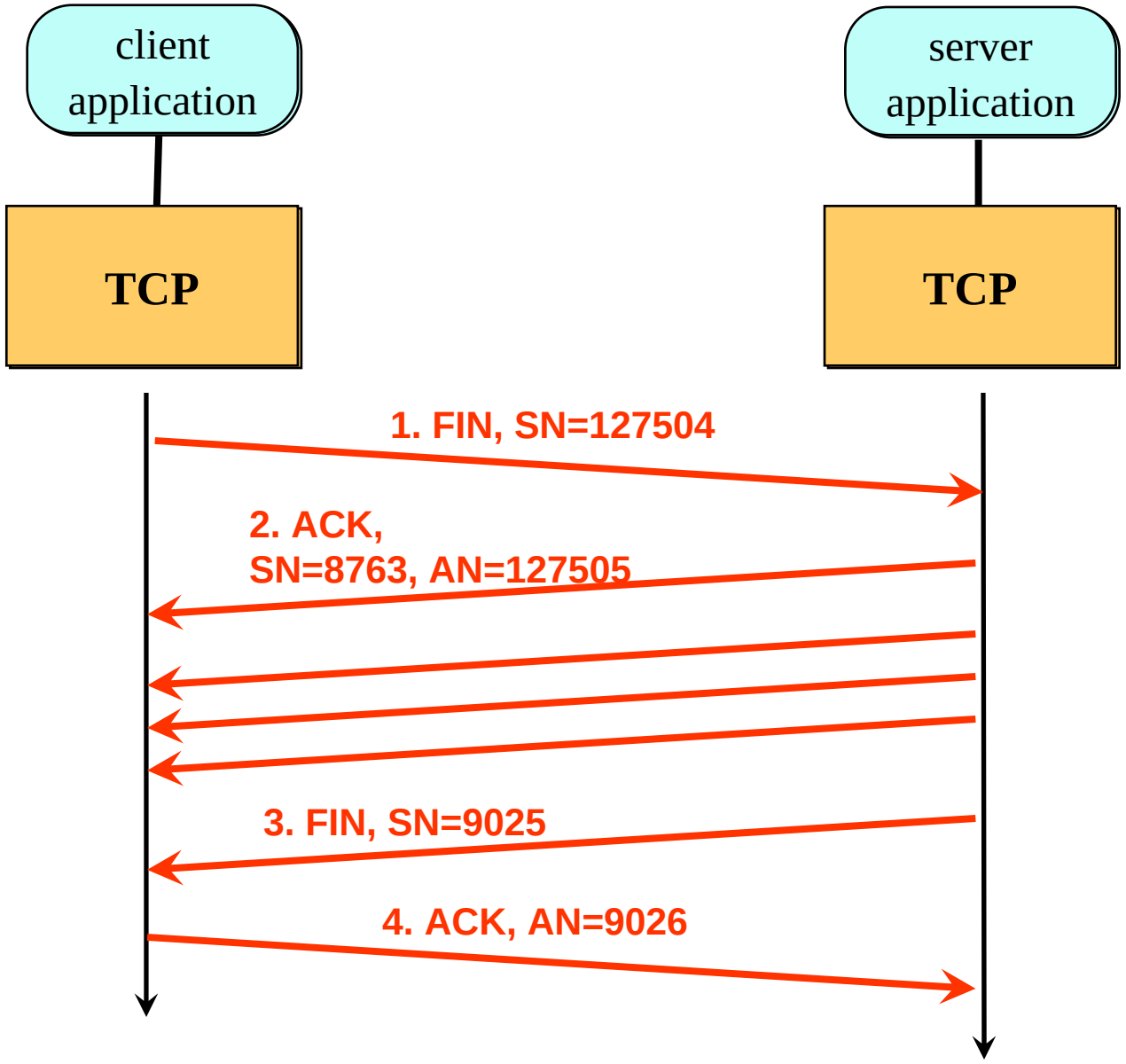
- **La connessione rimane comunque aperta nell'altra direzione e quindi il TCP dall'altra parte può continuare ad inviare dati (che verranno ovviamente riscontrati con degli ACK)**

Tear down (chiusura) delle connessioni



- **1. Infine, il TCP dall'altra parte chiude la connessione inviando un messaggio di FIN (flag FIN=1)**
- **2. Il TCP che aveva già chiuso la connessione in direzione opposta invia un ACK finale per confermare**

Tear down delle connessioni



Reset della connessione

- La connessione può anche essere chiusa senza scambio di messaggi nei due versi
- E’ possibile infatti settare il flag di RESET nel segmento e interrompere la connessione in entrambe le direzioni
- Il TCP che riceve un RESET chiude la connessione interrompendo ogni invio di dati

Controllo d'errore

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

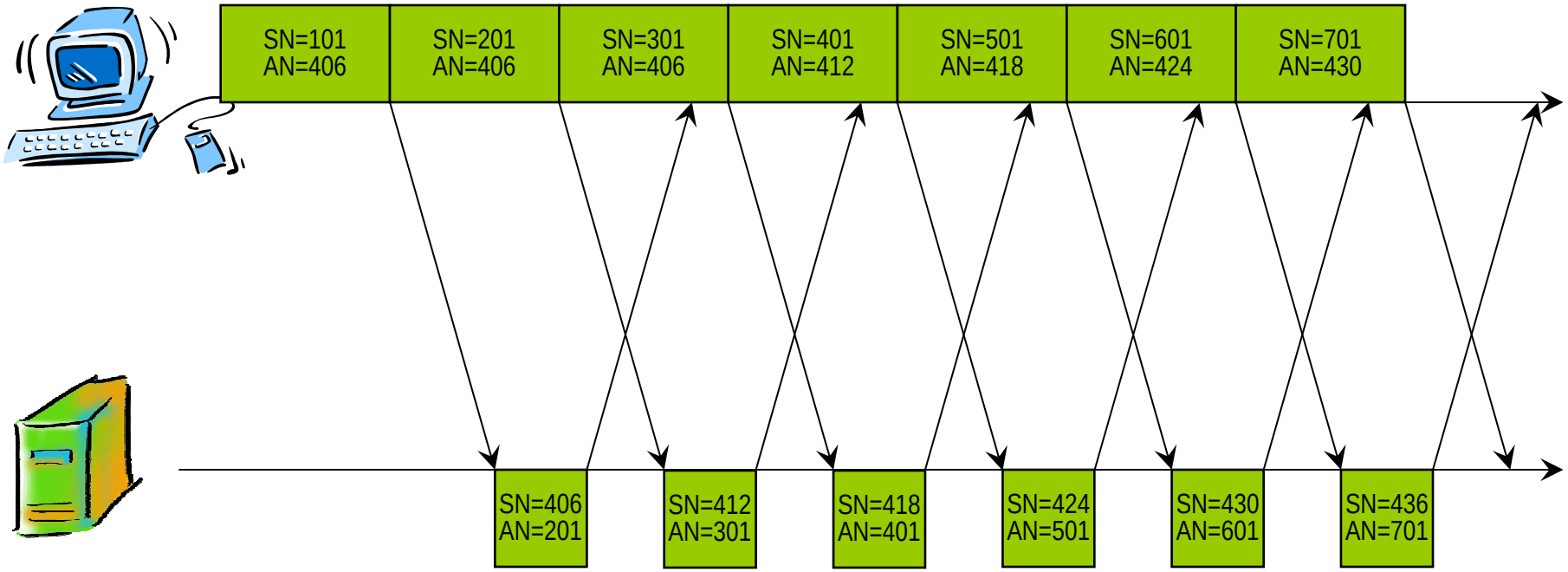
Controllo d'errore

esempio 1: senza errori

MSS=100 byte

Window= 4 MSS

Questo host invia segmenti lunghi 100 byte ciascuno



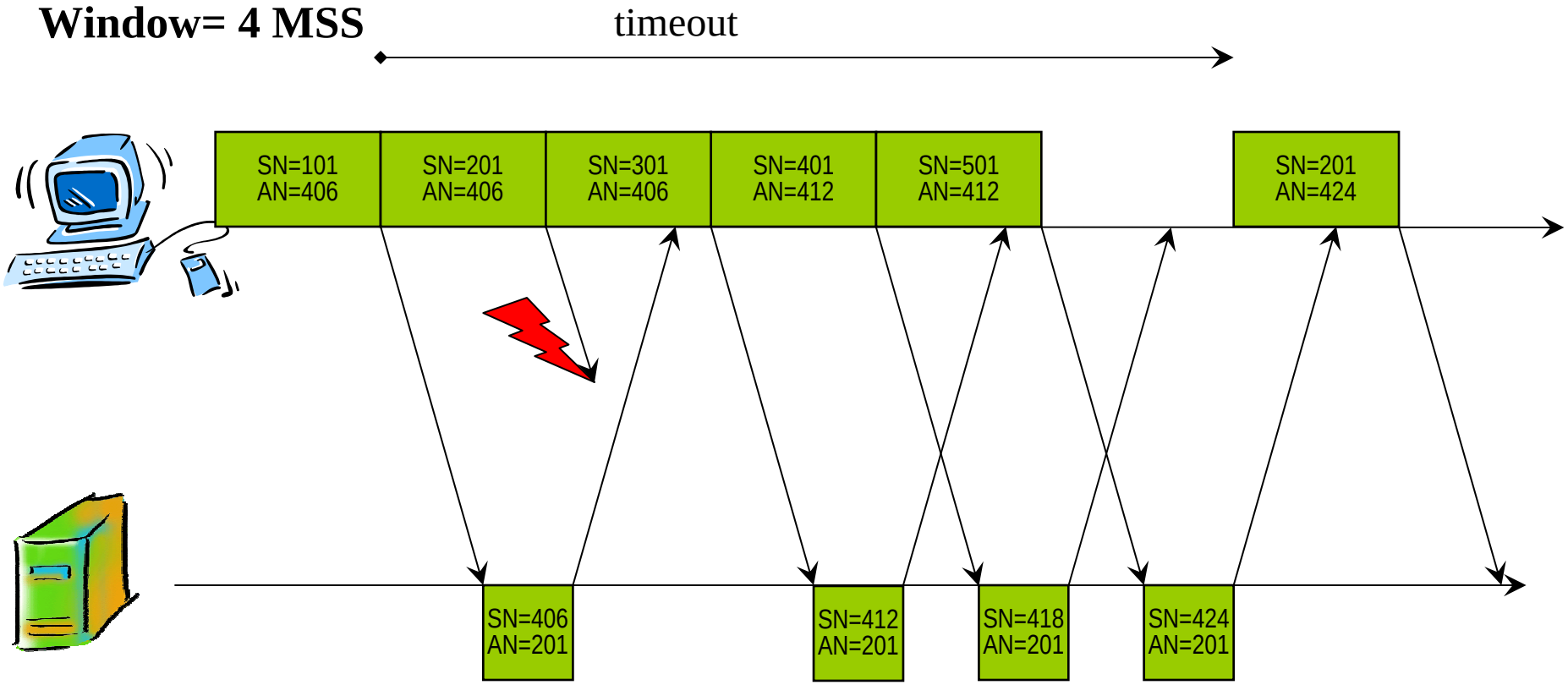
Questo host riscontra i segmenti inviati dall'altro, ed inoltre invia dati lunghi 6 byte ogni volta

Controllo d'errore

esempio 2: errore nei dati

MSS=100 byte

Window= 4 MSS

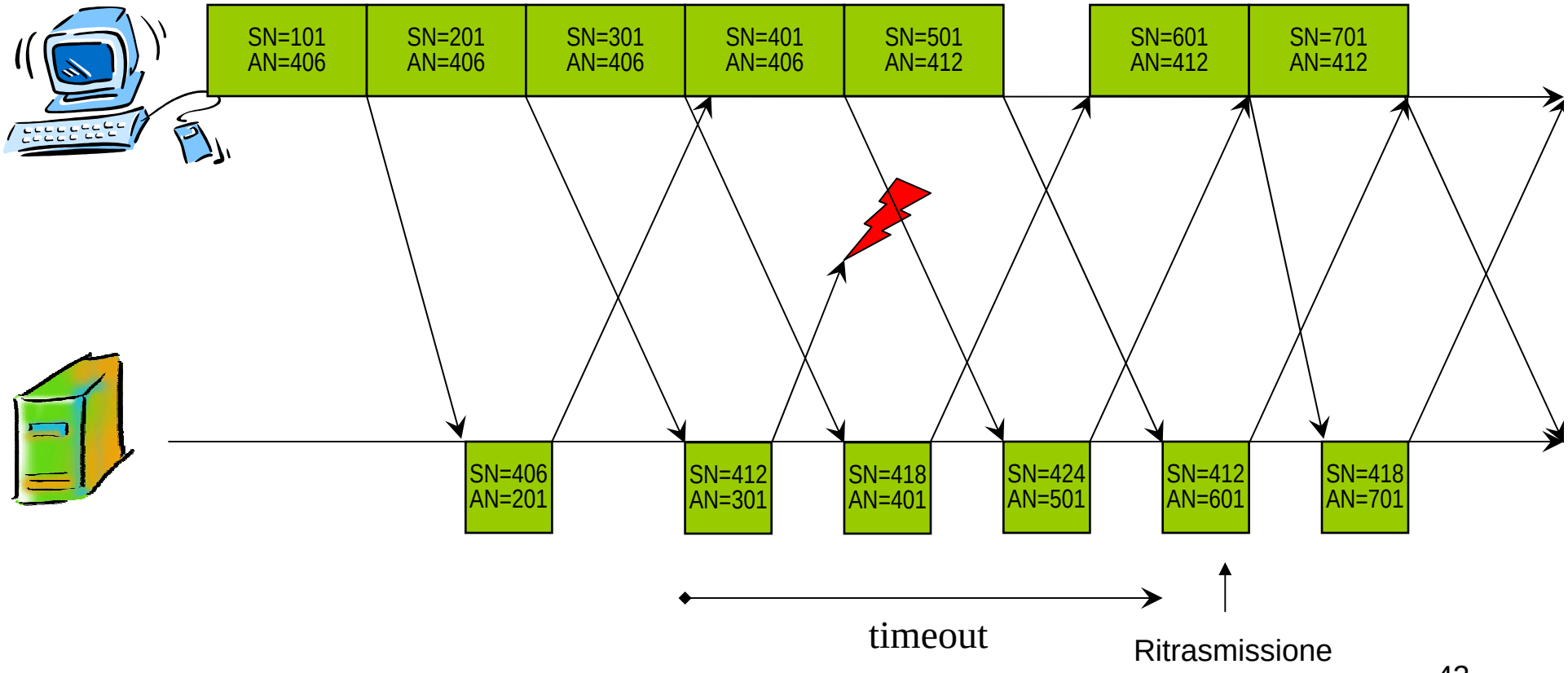


Controllo d'errore

esempio 3: errore nell'ack

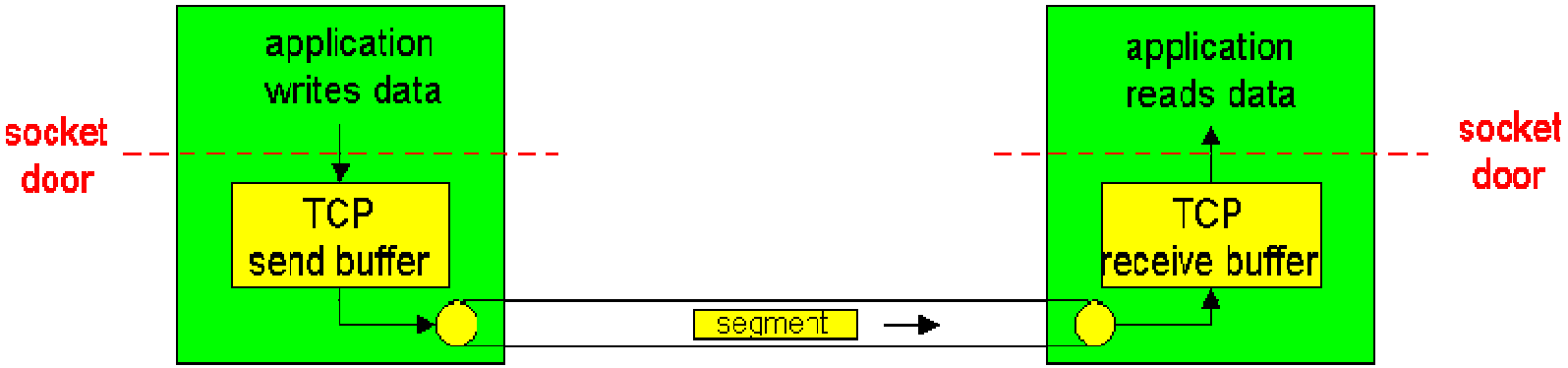
MSS=100 byte

Window= 4 MSS



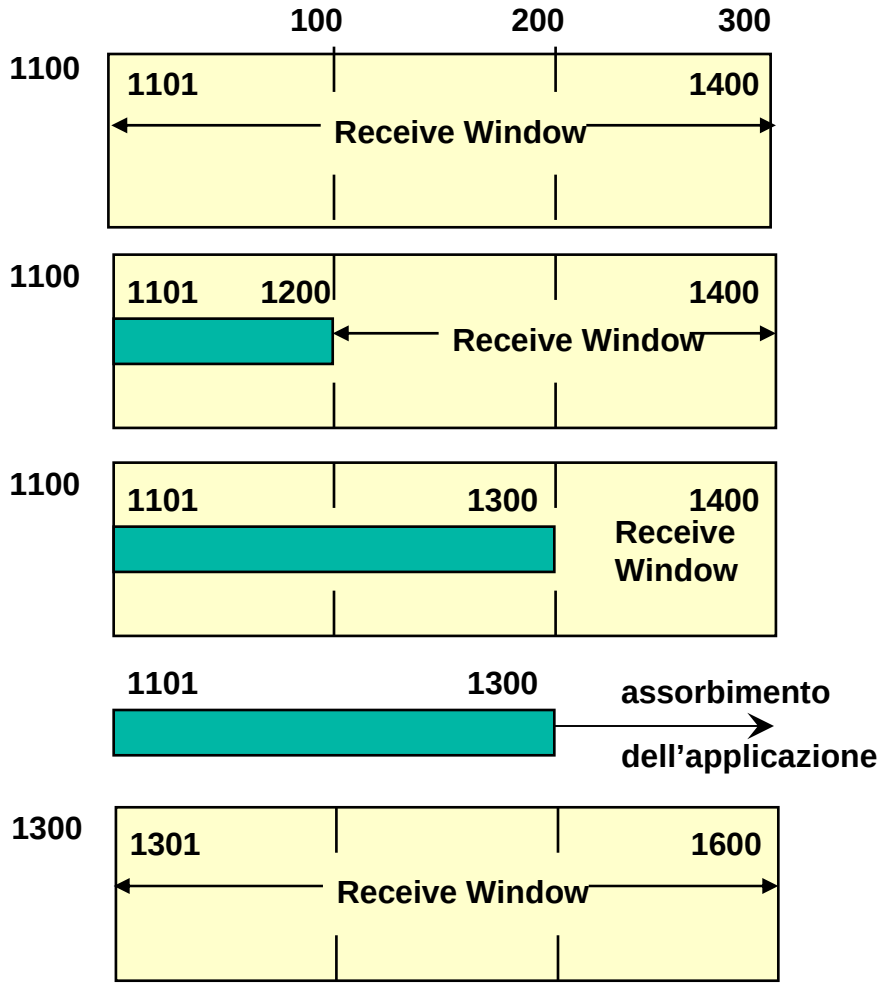
Controllo di flusso

- *Controllo di flusso:* **il TCP ricevente è responsabile del flusso di dati in ingresso. Il ricevitore decide quanti dati vuole ricevere, e comunica questo al trasmettitore**
  - **I dati in ingresso sono memorizzati nel buffer di ricezione fino a che l'applicazione ricevente è in grado di assorbirli**
  - **il ricevitore indica esplicitamente la dimensione della finestra di ricezione in ogni trama che viaggia in senso contrario**



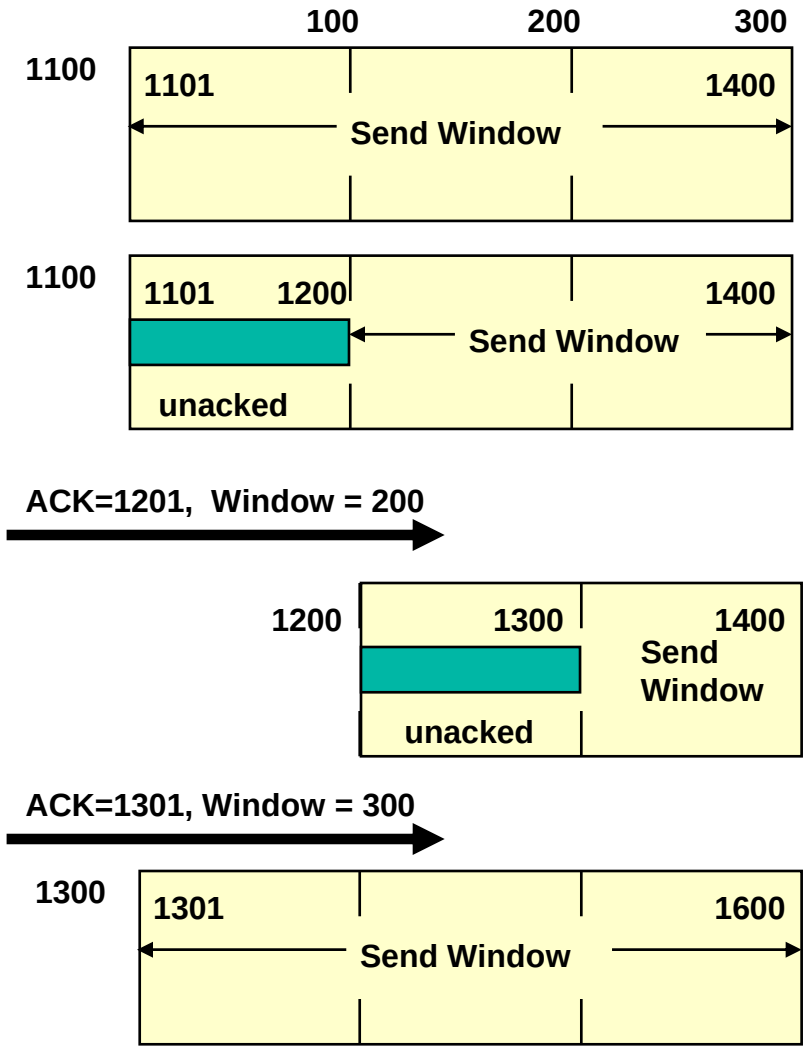
Controllo di flusso

- *Receive Window (RCVWND)*: la finestra di ricezione è lo spazio di buffer disponibile per ricevere nuovi dati
- Il buffer di ricezione può riempirsi, per esempio, a causa di congestione nel sistema operativo del ricevitore
- Il buffer di ricezione si estende dall'ultimo byte inoltrato all'applicazione fino alla fine del buffer



Controllo di flusso

- *Send Window (SNDWND)*
- il trasmettitore mantiene un buffer di trasmissione che tiene traccia di
  - dati che sono stati trasmessi ma non ancora riscontrati
  - dimensione della finestra di ricezione del partner
- Il buffer di trasmissione si estende dal primo byte non riscontrato all'estremo a destra della finestra di ricezione del ricevitore
- La finestra di trasmissione è la parte inutilizzata del buffer, e rappresenta i byte che possono essere trasmessi senza attendere ulteriori riscontri



Problemi con la finestra

- Silly window syndrome - lato ricevitore:
  - il ricevitore svuota lentamente il buffer di ricezione
  - invia segmenti con finestra molto piccola
  - il trasmettitore invia segmenti corti con molto overhead
- soluzione
  - il ricevitore “mente” al trasmettitore indicando una finestra nulla sino a che il suo buffer di ricezione non si è svuotato per metà o per una porzione almeno pari al MSS

min (1/2 Receive_Buffer_Size, Maximum_Segment_Size).

Problemi con la finestra

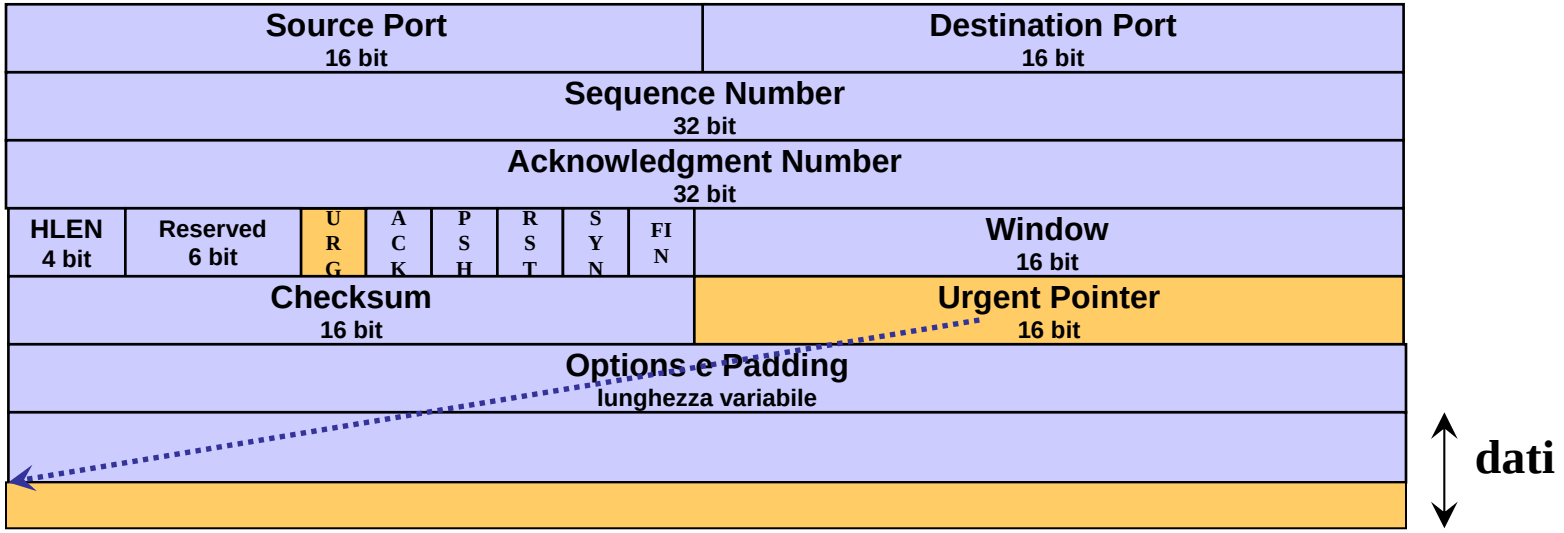
- Silly window syndrome - lato trasmettitore:
  - l'applicazione genera dati lentamente
  - invia segmenti molto piccoli man mano che vengono prodotti
- soluzione
  - il TCP sorgente invia la prima porzione di dati anche se corta
  - gli altri segmenti vengono generati e inviati solo se
    - ✓ il buffer d'uscita contiene dati sufficienti a riempire un MSS
    - ✓ oppure, quando si riceve un acknowledgement per il segmento precedente.

Push

- Il normale funzionamento dell'inoltro del flusso di byte può essere alterato esplicitamente quando ci sono dati che richiedono di essere immediatamente consegnati all'applicazione ricevente
- Per ottenere un inoltro immediato dei dati da parte del TCP ricevente l'applicazione può inviare un comando di PUSH
- Per ottenere analogo comportamento dall'applicazione ricevente viene settato il flag di PUSH nel segmento
- E' questo il caso di applicazioni come TELNET

Dati URGENT

- Alternativamente, i dati possono essere marcati come URGENT
- in questo caso il meccanismo costituisce un vero e proprio meccanismo di segnalazione in banda
- i dati urgenti sono identificati all'interno del flusso di dati e non seguono le regole del controllo di flusso



Gestione del Timeout

- Uno dei problemi è stabilire il valore ottimo del timeout:
- se il timeout è troppo breve, il trasmettitore riempirà il canale di ritrasmissioni di segmenti,
- Al contrario, se è troppo lungo impedisce il recupero veloce di reali errori
- il valore ottimale dipende fortemente dal ritardo in rete; esempi estremi:
  - rete locale
  - collegamento satellitare
- il TCP calcola dinamicamente un valore opportuno per il timeout stimando il RTT (Round Trip Time)

Stima del RTT

- Il TCP adatta il timeout di trasmissione alle condizioni reali della rete tramite gli algoritmi di **Karn e Jacobson**
- i campioni di round-trip-time $\{RTT^{(i)}\}$ sono definiti come il tempo che passa tra la trasmissione di un segmento e la ricezione del relativo riscontro

Stima del valor medio

- Sulla base delle misure, il sender TCP calcola lo Smoothed Round Trip Time (SRTT) tramite l'algoritmo di Jacobson

$$SRTT^{(i)} = (1-\alpha)\, SRTT^{(i-1)} + \alpha\, RTT^{(i)}.$$

- Con α compreso tra 0 e 1 (tipicamente 1/8)

Stima del RTT

Stima della deviazione standard

- **Oltre al valor medio viene anche stimata la deviazione standard dei RTT usando i seguenti campioni:**

$$DEV = |RTT^{(i)} - SRTT^{(i-1)}|$$

- **anche della deviazione standard viene calcolato un valore filtrato (smoothed):**

$$SDEV^{(i)} = 3/4\ SDEV^{(i-1)} + 1/4\ DEV$$

Calcolo del Timeout

- **Sulla base dei valori stimati il timeout è calcolato come**

$$TIMEOUT = SRTT + 2\ SDEV$$

- **All'inizio SRTT viene posto uguale a zero e SDEV = 1.5 s, e quindi il valore del timeout parte a 3 s**
- **A seguito di una ritrasmissione è meglio passare all'*algoritmo di Karn*:**
  - **RTT non viene aggiornato**
  - **il timeout è moltiplicato per un fattore fisso (tipicamente 2)**
  - **il timeout cresce fino ad un valore massimo**
  - **dopo un numero massimo di ritrasmissioni la connessione viene chiusa**

Persistenza

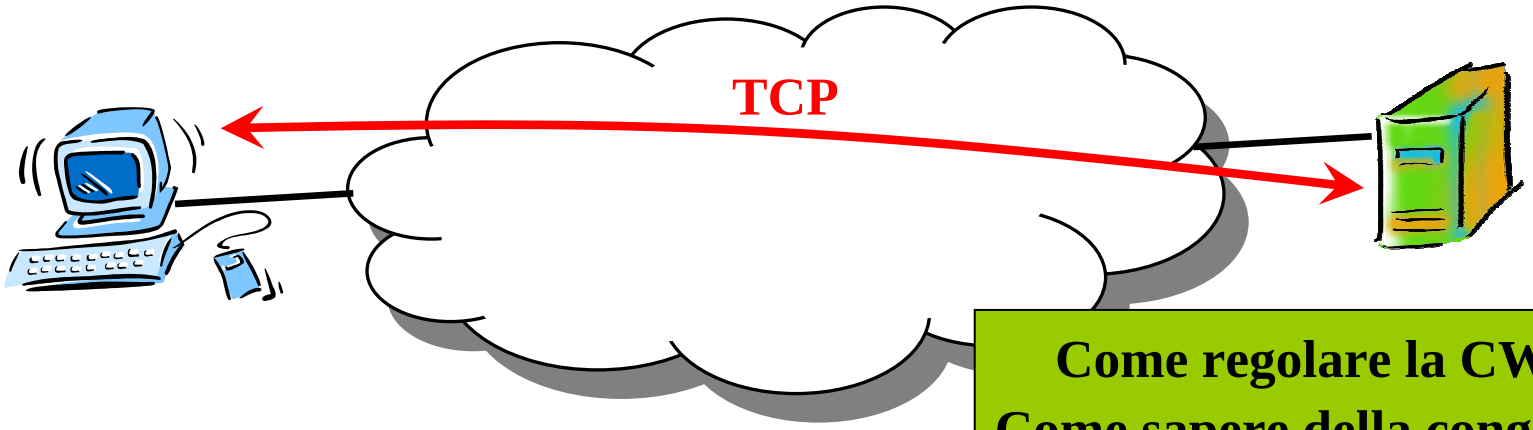
- Se il destinatario fissa a zero la finestra di ricezione, la sorgente TCP interrompe la trasmissione
- la trasmissione riprende quando il destinatario invia un ACK con una dimensione della finestra diversa da zero
- nel caso in cui questo ACK andasse perso la connessione rimarrebbe bloccata
- per evitare questa situazione si usa un *timer di persistenza* che viene attivato quando arriva un segmento con finestra nulla
- se il timer di persistenza scade (valore di timeout uguale a quello di ritrasmissione) viene inviato un piccolo segmento di sonda (probe)
- se viene ricevuto un ACK si esce dallo stato critico altrimenti al nuovo scadere del timeout si invia un altro probe

Controllo di congestione

- Utilizzando le finestre di trasmissione e di ricezione, il TCP può eseguire un controllo di flusso efficace
- La finestra di ricezione (RCVWND) dipende dalla disponibilità di buffer per l'inoltro alle applicazioni (controllo di flusso)
- D'altra parte, questo meccanismo non è sufficiente ad evitare la congestione nella rete
- Nella rete INTERNET attuale non ci sono meccanismi sofisticati di controllo di congestione a livello di rete (come ad esempio meccanismi di controllo del traffico in ingresso)
- il controllo di congestione è delegato al TCP !!!
- Essendo il TCP implementato solo negli host, il controllo di congestione è di tipo end-to-end

Controllo di congestione

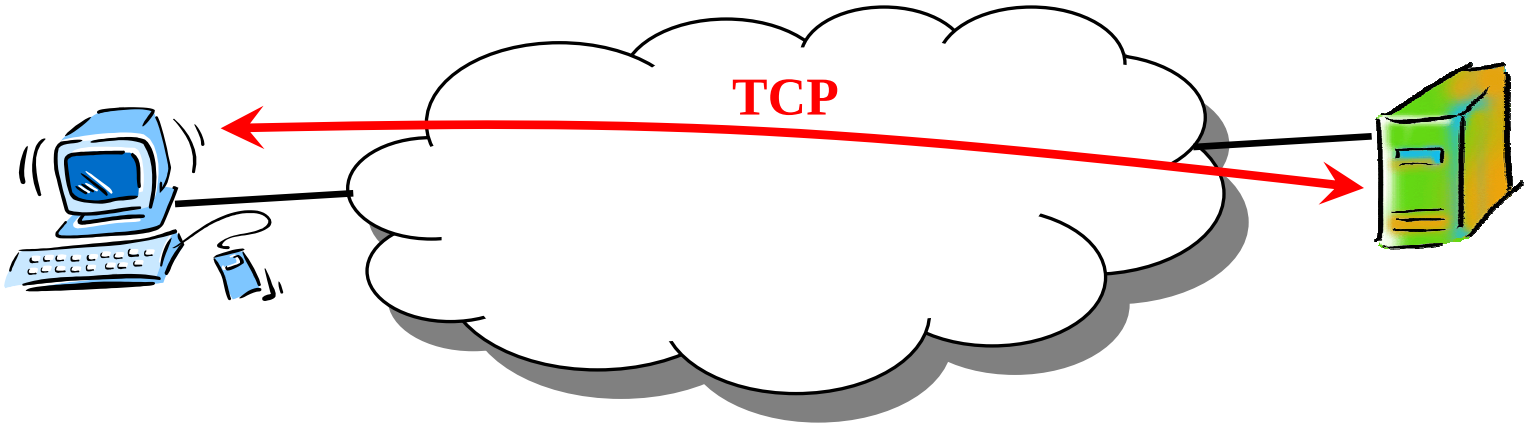
- Il modo più naturale per controllare il ritmo di immissione in rete dei dati per il TCP è quello di regolare la finestra di trasmissione
- Il trasmettitore mantiene una Congestion Window (CWND) che varia in base agli eventi che osserva (ricezione ACK, timeout)
- il trasmettitore non può trasmettere più del minimo tra RCVWND e CWND



Come regolare la CWND?
Come sapere della congestione?

Controllo di congestione

- L'idea base del controllo di congestione del TCP è quello di interpretare la perdita di un segmento, segnalata dallo scadere di un timeout di ritrasmissione, come un evento di congestione
- La reazione ad un evento di congestione è quella di ridurre la finestra (CWND)



Slow Start & Congestion Avoidance

- Il valore della finestra CWND viene aggiornato dal trasmettitore TCP in base ad un algoritmo
- il modo in cui avviene l'aggiornamento dipende dalla fase (o stato) in cui si trova il trasmettitore
- esistono due fasi fondamentali:
  - *Slow Start*
  - *Congestion Avoidance*
- La variabile SSTHRESH è mantenuta al trasmettitore per distinguere le due fasi:

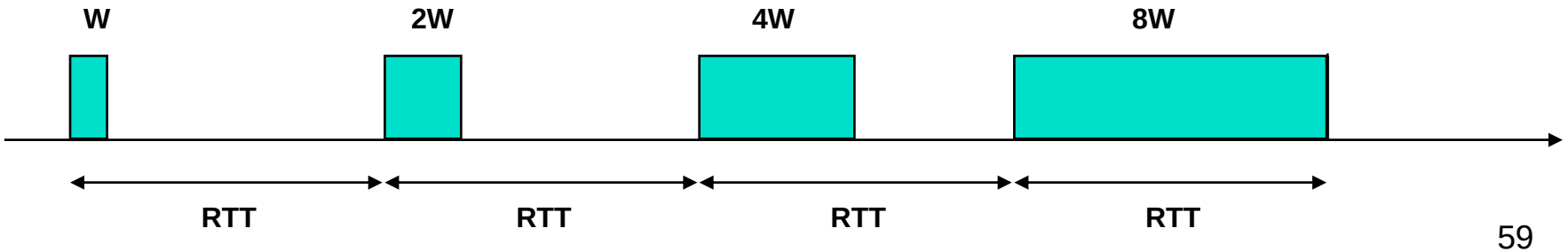
→ se CWND < SSTHRESH si è in Slow Start

→ se CWND > SSTHRESH si è in Congestion Avoidance

Slow Start

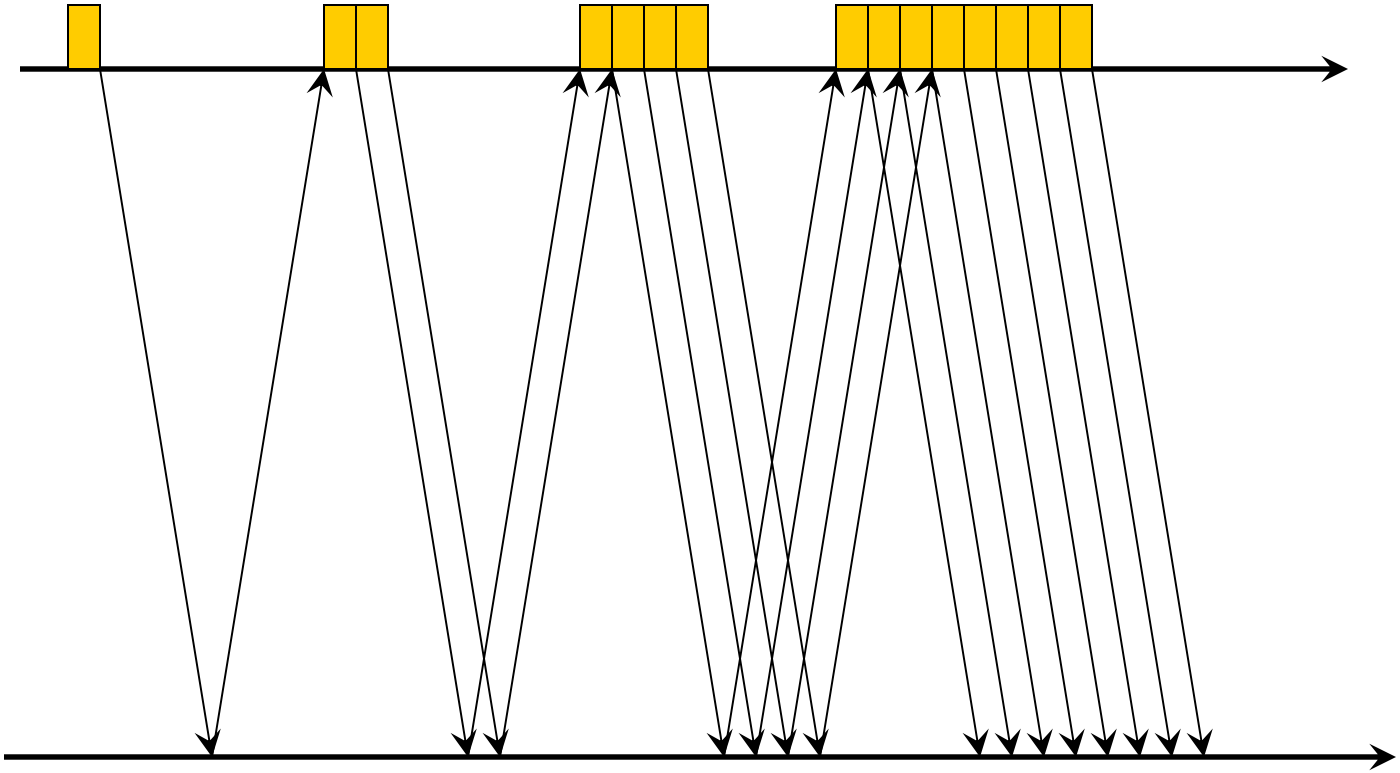
- All'inizio, il trasmettitore pone la CWND a 1 segmento (MSS) e la SSTHRESH ad un valore di default molto elevato
- Essendo CWND < SSTHRESH si parte in Slow Start
- *In Slow Start:*
  - *la CWND viene incrementata di 1 per ogni ACK ricevuto*

- Si invia un segmento e dopo RTT si riceve l'ACK, si pone CWND a 2 e si inviano 2 segmenti, si ricevono 2 ACK, si pone CWND a 4 e si inviano 4 segmenti, ...



Slow Start

- **Al contrario di quanto il nome faccia credere l'incremento della finestra avviene in modo esponenziale (raddoppia ogni RTT)**



Slow Start

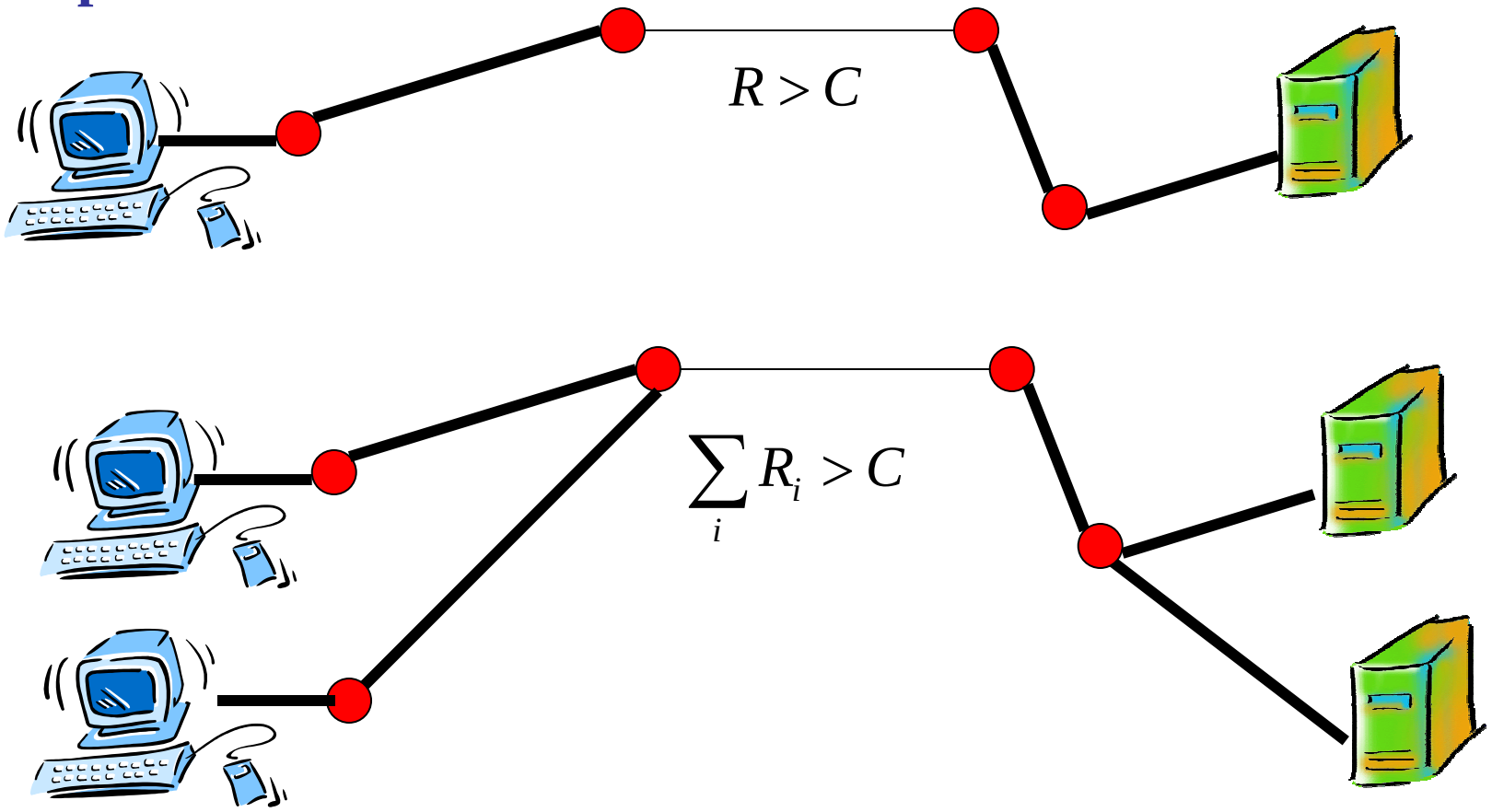
- **L'incremento può andare avanti fino al primo evento di congestione o fino a che CWND < SSTHRESH**
- **Insieme alla finestra aumenta il ritmo (o rate) di trasmissione che può essere stimato come:**

$$R = \frac{CWND}{RTT} \quad [\text{bit/s}]$$

- **avendo espresso la CWND in bit e il RTT in secondi e nell'ipotesi che RCVWND > CWND.**
- **Un evento di congestione si verifica quando il ritmo di trasmissione porta in congestione un link sul percorso in rete verso la destinazione**

Evento di Congestione

- **Un link è congestionato quando la somma dei ritmi di trasmissione dei flussi che lo attraversano è maggiore della sua capacità**



Evento di congestione

- **Reazione ad un evento di congestione (scatta un timeout di ritrasmissione):**
  - **il TCP reagisce ponendo SSTHRESH uguale alla metà dei "byte in volo" (byte trasmessi ma non riscontrati); più precisamente**

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

  - **e ponendo CWND a 1**

- **Si noti che di solito i "byte in volo" sono pari a CWND**

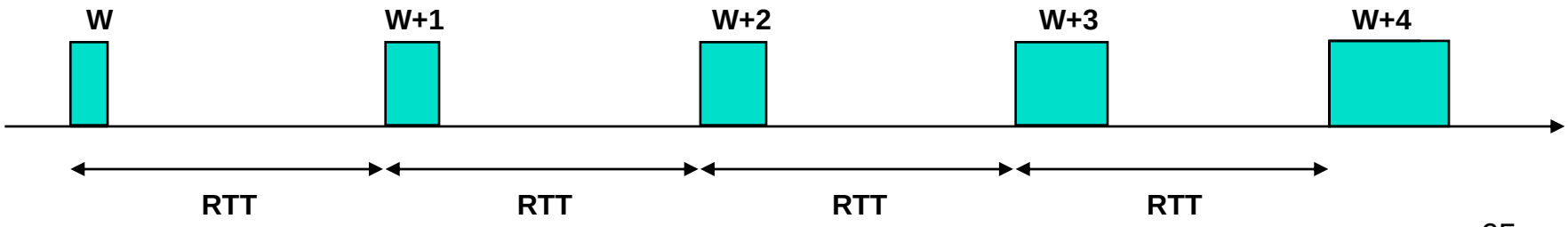
Evento di congestione

- Come risultato:
  - CWND (pari ad 1 MSS) è ora minore di SSTHRESH e si entra nella fase di Slow Start
  - il trasmettitore invia un segmento e la sua CWND è incrementata di 1 ad ogni ACK
- Il trasmettitore ritrasmette tutti i segmenti a partire da quello per cui il timeout è scaduto (politica go-back-N)

- Il valore a cui è posta la SSTHRESH è una stima della finestra ottimale che eviterebbe futuri eventi di congestione

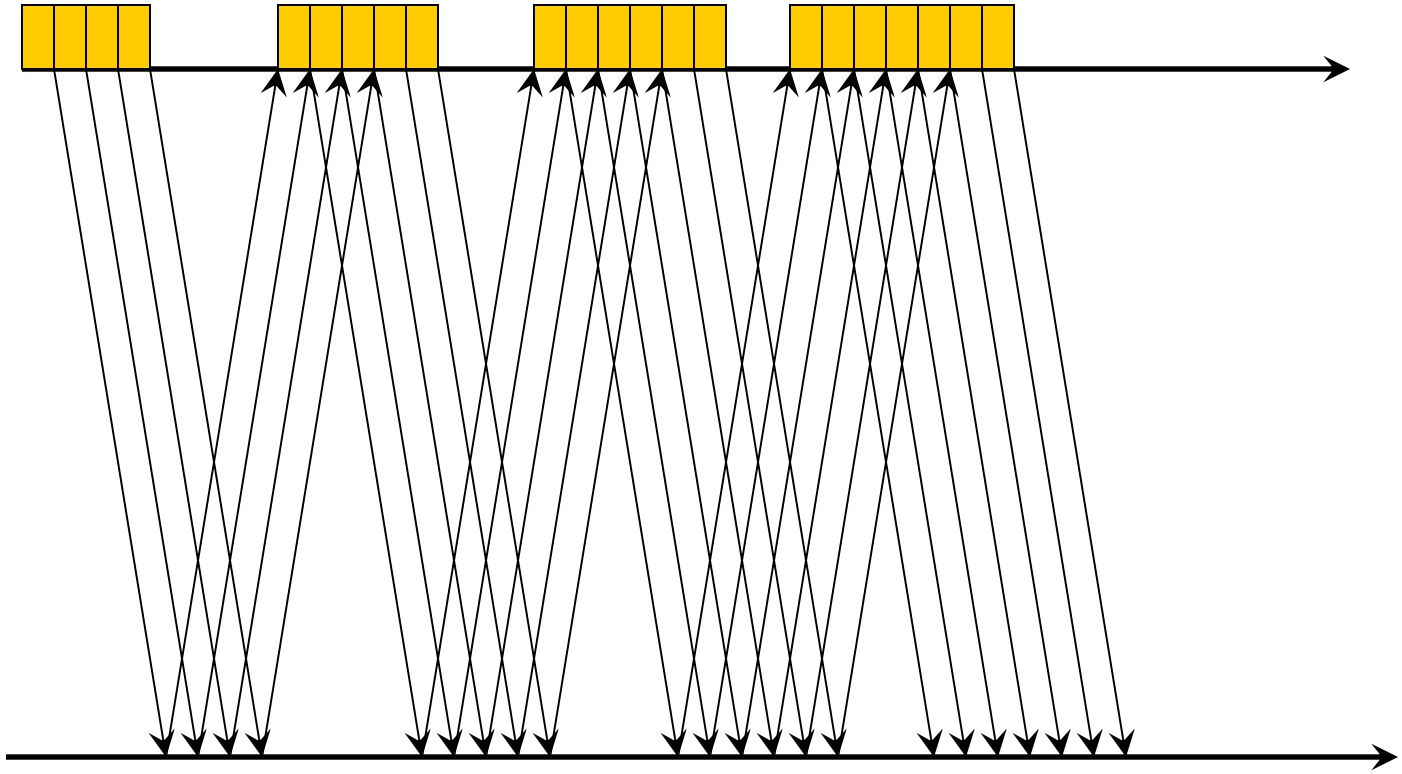
Congestion Avoidance

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

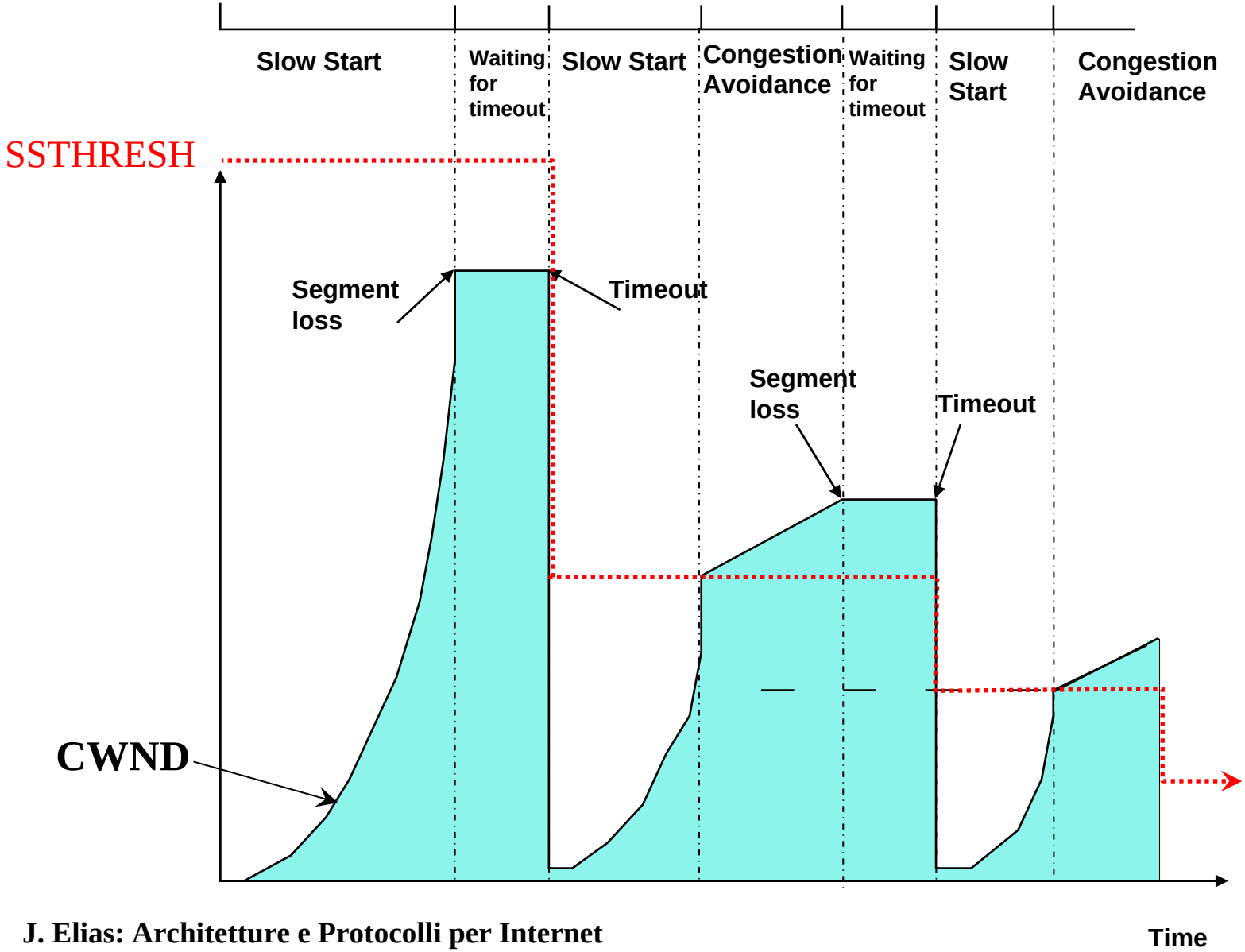


Congestion Avoidance

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



Esempio di funzionamento



Fast Retransmit e Fast Recovery

- Algoritmi implementati nella versione TCP nota come TCP Reno
- ACK duplicati:
  - Se il TCP ricevente riceve pacchetti fuori sequenza (diversi da quello atteso) invia immediatamente un ACK con il AN contenente il segmento atteso
- Gli ACK duplicati possono essere causati da perdite di pacchetti
- I meccanismi di Fast Retransmit e Fast Recovery cercano di recuperare velocemente queste perdite

Fast Retransmit e Fast Recovery

1. **Alla ricezione del 3° ACK consecutivo duplicato (con lo stesso AN): si pone**

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

2. **Viene ritrasmesso il pacchetto indicato dall'AN**
3. **Si pone la** $CWND = SSTHRESH + 3 \cdot MSS$
4. **Per ogni ulteriore ACK duplicato ricevuto la CWND viene incrementata di 1**
5. **Vengono trasmessi nuovi segmenti se consentito dai valori di CWND e RWND**
6. **Appena arriva un ACK che riscontra nuovi dati si esce dalla fase di fast recovery e si pone di nuovo**

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

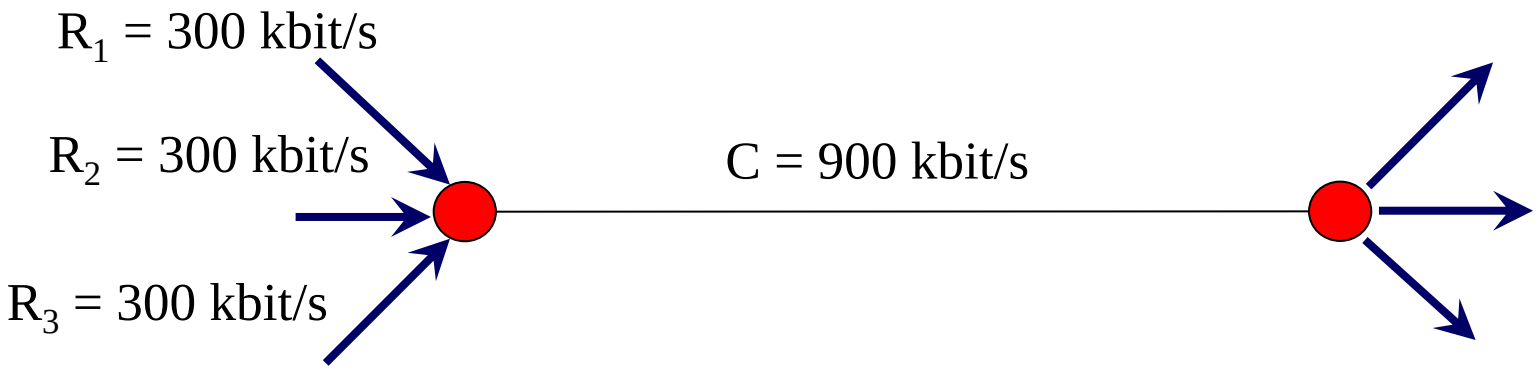
Fast Retransmit e Fast Recovery

- **Logica:**
  - **Se arrivano ACK duplicati un pacchetto sarà andato perso**
  - **Se arrivano ACK duplicati vuol dire che i pacchetti successivi a quello perso sono arrivati (niente congestione)**
  - **Se non c'è congestione si può incrementare la CWND del numero di pacchetti sicuramente arrivati al ricevitore**
- **Problemi:**
  - **Se ci sono perdite multiple nella finestra di trasmissione tipicamente non si riescono a recuperare**

Condivisione equa delle risorse

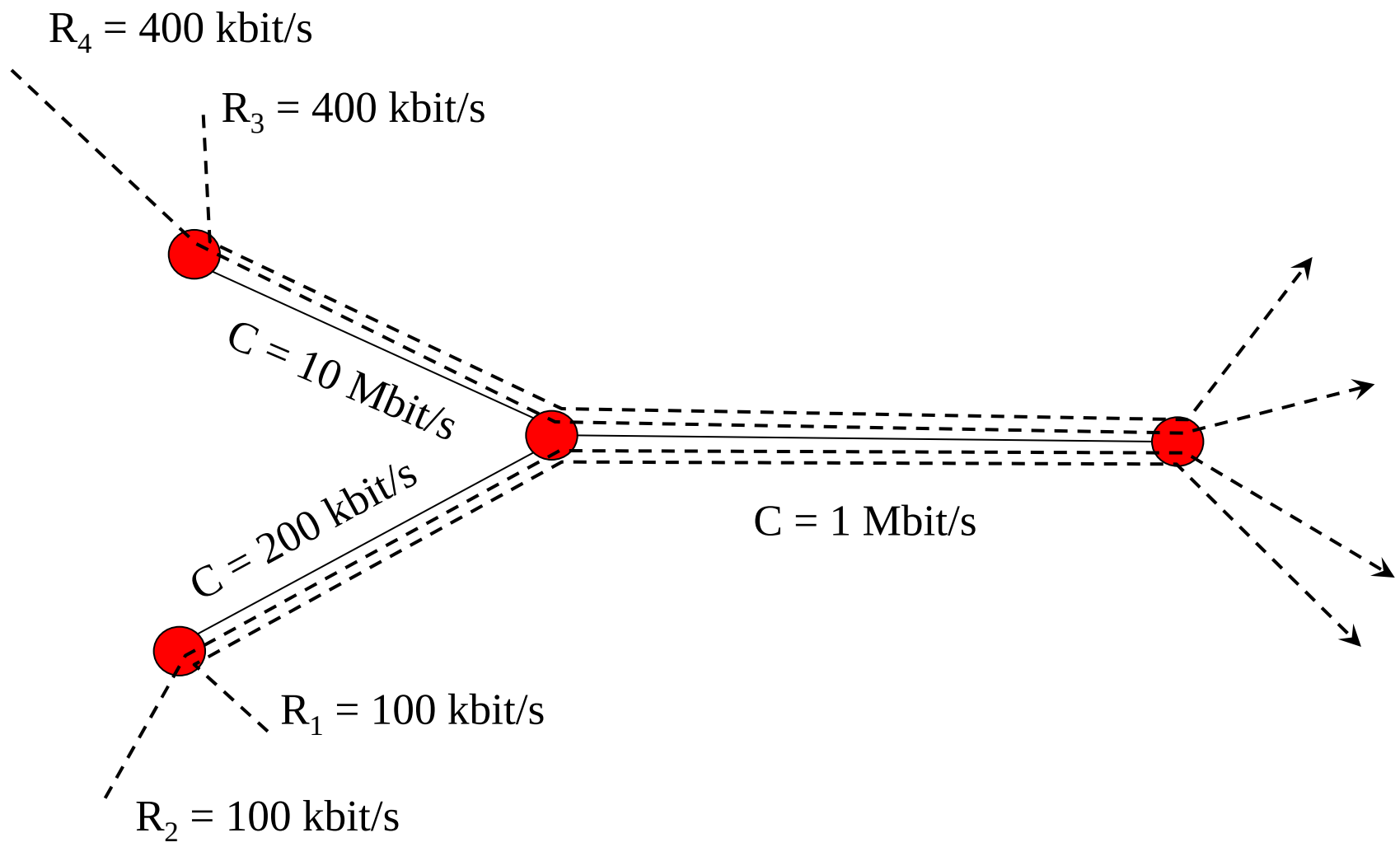
- Si può far vedere che in condizioni ideali il meccanismo di controllo del TCP è in grado
  - di limitare la congestione in rete
  - consentire di dividere in modo equo la capacità dei link tra i diversi flussi
- Le condizioni ideali sono alterate tra l'altro da
  - differenti RTT per i diversi flussi
  - buffer nei nodi minori del prodotto banda-ritardo

Condivisione equa delle risorse



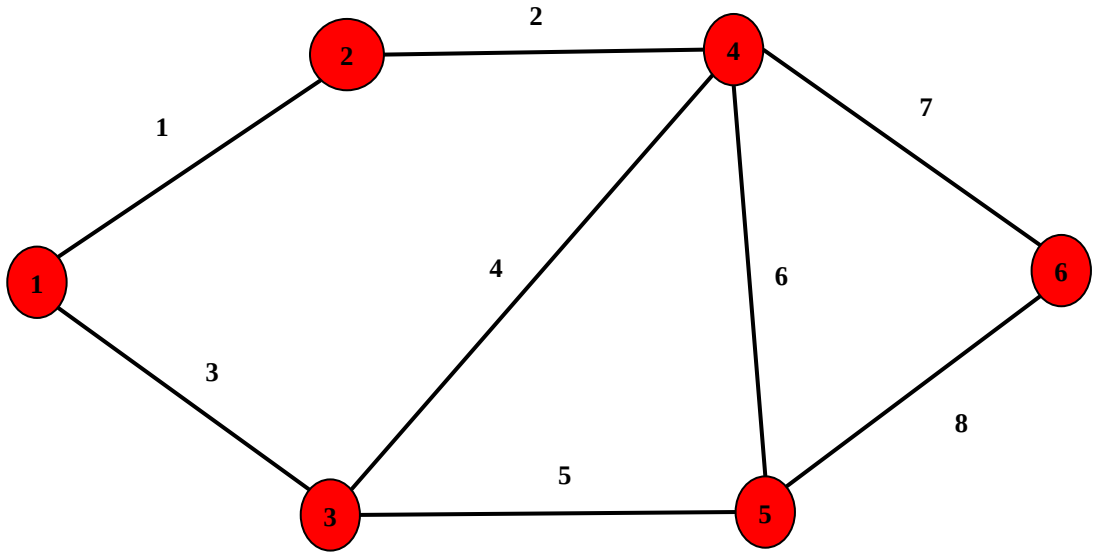
- **I valori dei rate indicati sono solo valori medi e valgono solo in condizioni ideali**
- **il ritmo di trasmissione in realtà cambia sempre e in condizioni non ideali la condivisione può non essere equa**

Condivisione equa delle risorse



Calcolo del fair-share

- **Algoritmo di calcolo fair share f_{ij}^{z} per ogni flusso *z* tra i nodi sorgente-destinazione *(i,j):***
  - **Sia noto il numero di flussi n_{ij} tra ogni coppia sorgente-destinazione *(i,j)***
  - **Sia noto l'instradamento per ogni flusso relativo alla coppia *(i,j)***

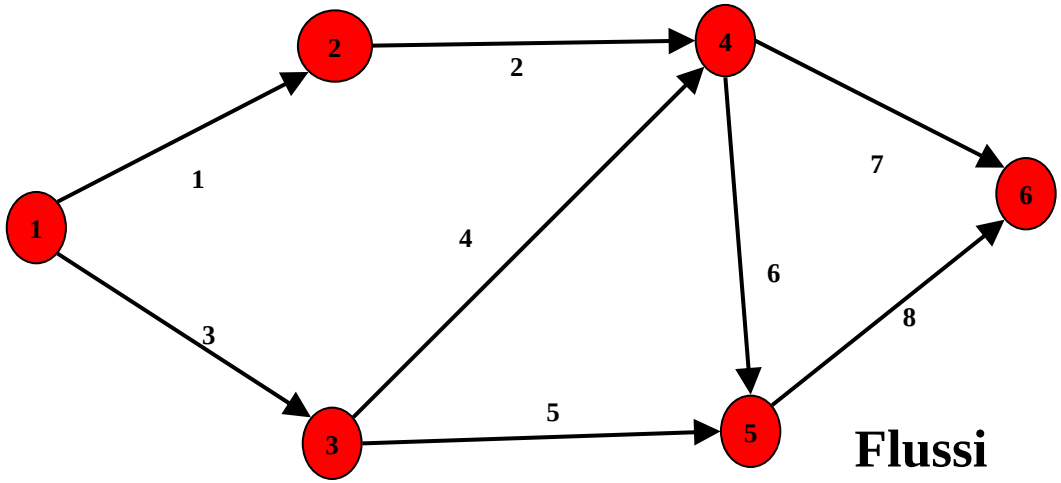


i	*j*	n_{ij}	*percorso*
1	6	3	1,2,7
1	4	3	3,4
1	5	4	3,5
3	6	2	4,7
3	5	5	4,6
2	5	3	2,6

Calcolo del fair-share

1. Si ponga inizialmente $f_{ij}^{z} = 0 \ \forall \ (i,j,z)$
2. Si elimini dall'insieme L degli archi quelli aventi il numero di flussi che lo attraversano n_k pari a zero
3. Per ogni link $k \in L$ si calcoli il rapporto $F_k = C_k / n_k$, dove C_k è la capacità del link
4. Sia $\alpha \mid F_\alpha = min_k(F_k)$
5. Si assegni $f_{ij}^{z} = F_\alpha \ \forall \ i,j,z : (i,j) \in L_\alpha$ dove L_α è l'insieme dei flussi (i,j) che attraversano il link α
6. Si assegni
$$C_k = C_k - \sum_{(i,j)\in L_\alpha} \sum_{z} f_{ij}^{z}$$
$$n_k = n_k - \sum_{(i,j)\in (L_\alpha \cup L_k)} n_{ij}$$
7. Si elimini dall'insieme L l'arco α e tutti quelli aventi n_k pari a zero
8. Se L è vuoto STOP. Altrimenti ripeti da 3.

Calcolo del fair-share: esempio



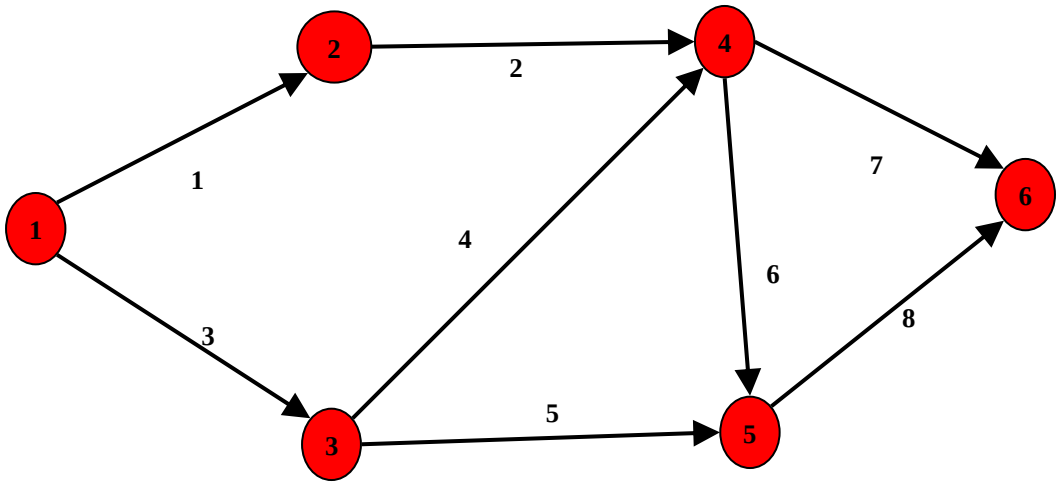
Flussi

i	*j*	n_{ij}	*percorso*
1	6	3	1,2,7
1	4	3	3,4
1	5	4	3,5
3	6	2	4,7
3	5	5	4,6
2	5	3	2,6

Capacità

link	C_k
1	9
2	6
3	7
4	15
5	12
6	2
7	5
8	5

Calcolo del fair-share: esempio



Flussi

i	*j*	n_{ij}	*percorso*
1	6	3	1,2,7
1	4	3	3,4
1	5	4	3,5
3	6	2	4,7
3	5	5	4,6
2	5	3	2,6

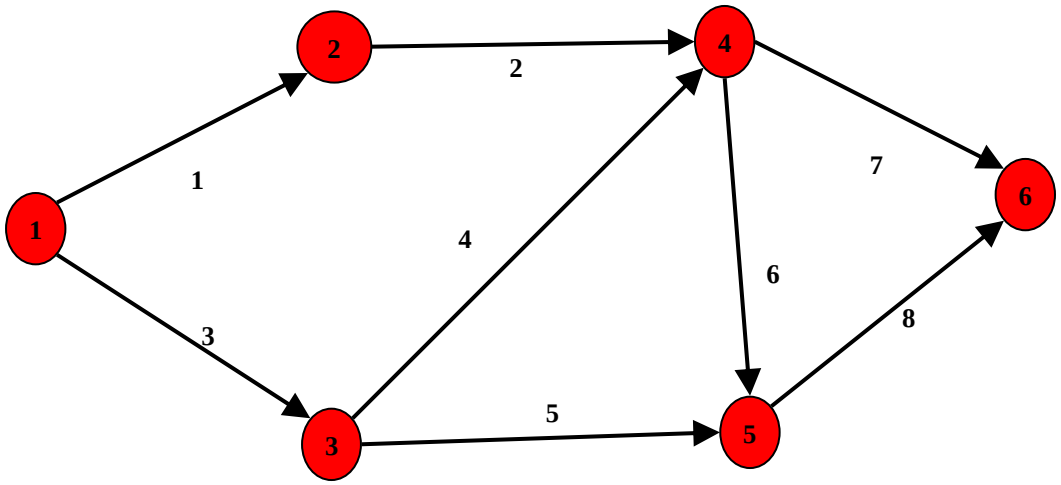
Step 1

link	C_k	n_k	F_k
1	9	3	3
2	6	6	1
3	7	7	1
4	15	10	1.5
5	12	4	3
6	2	8	0.25
7	5	5	1
8	5	0	--

i	*j*	n_{ij}	f_{ij}^{k}
1	6	3	0
1	4	3	0
1	5	4	0
3	6	2	0
3	5	5	0.25
2	5	3	0.25

link	C_k	n_k
1	9	3
2	5.25	3
3	7	7
4	13.75	5
5	12	4
6	0	0
7	5	5
8	5	0

Calcolo del fair-share: esempio



Flussi

i	*j*	n_{ij}	*percorso*
1	6	3	1,2,7
1	4	3	3,4
1	5	4	3,5
3	6	2	4,7
3	5	5	4,6
2	5	3	2,6

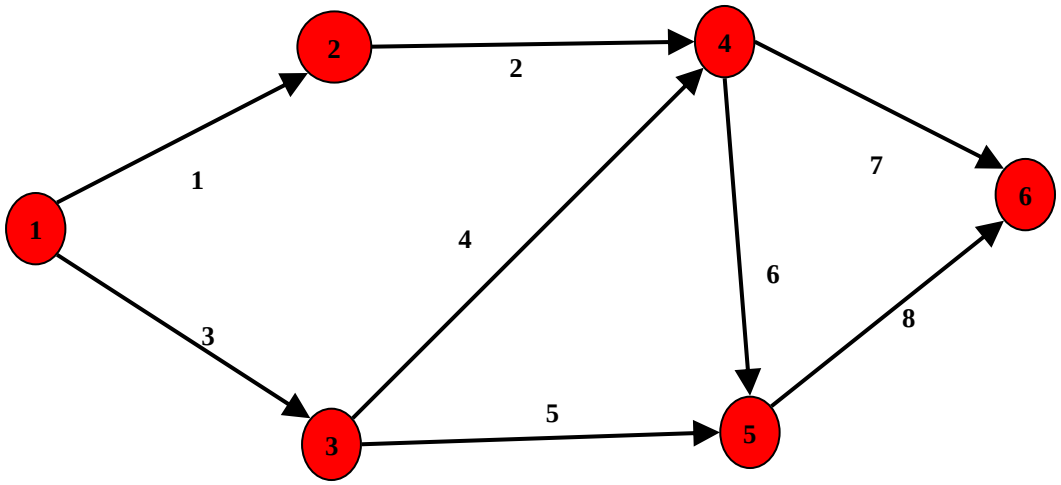
Step 2

link	C_k	n_k	F_k
1	9	3	3
2	5.25	3	1.75
3	7	7	1
4	13.75	5	2.75
5	12	4	3
6	0	0	--
7	5	5	1
8	5	0	--

i	*j*	n_{ij}	f_{ij}^{z}
1	6	3	1
1	4	3	0
1	5	4	0
3	6	2	1
3	5	5	0.25
2	5	3	0.25

link	C_k	n_k
1	6	0
2	2.25	0
3	7	7
4	11.75	3
5	12	4
6	0	0
7	0	0
8	5	0

Calcolo del fair-share: esempio



Flussi

i	*j*	n_{ij}	*percorso*
1	6	3	1,2,7
1	4	3	3,4
1	5	4	3,5
3	6	2	4,7
3	5	5	4,6
2	5	3	2,6

Step 3

link	C_k	n_k	F_k
1	6	0	--
2	2.25	0	--
3	7	7	1
4	11.75	3	3.92
5	12	4	3
6	0	0	--
7	0	0	--
8	5	0	--

i	*j*	n_{ij}	f_{ij}^{z}
1	6	3	1
1	4	3	1
1	5	4	1
3	6	2	1
3	5	5	0.25
2	5	3	0.25

link	C_k	n_k
1	6	0
2	2.25	0
3	0	0
4	8.75	0
5	8	0
6	0	0
7	0	0
8	5	0

Approfondimenti

- D.E. Comer, Internetworking with TCP/IP, capitoli interenti UDP e TCP
- RFC 768 (User Datagram Protocol)
- RFC 793 (TCP), primo RFC sul TCP
- RFC 2581
- RFC 2582
- RFC 2988 (Computing TCP's Retr. Timer)