È il livello che si occupa dei servizi in esecuzione su ogni host, estende il servizio di consegna. I due protocolli principali sono:

- TCP → Transfer Control Protocol
- UDP → User Datagram Protocol Le funzionalità minime che deve implementare un protocollo di livello 4 sono:
- **Multiplazione** e **Demultiplazione** dei messaggi tra processi
- rilevamento dell'errore

> il controllo dell'errore del protocollo IP è fatto solo sul suo header e non sul payload, quindi il protocollo IP non prevede un eventuale rilevamento dell'errore

UDP soddisfa solamente le due funzionalità base richieste per essere un protocollo di livello 4, mentre TCP implica anche la trasmissione *affidabile* e la *gestione della congestione di rete*.

multiplazione e demultiplazione

UDP e TCP attuano la multiplazione e la demultiplazione includendo due campi speciali nel header:

- n° porta mittente
- n° porta destinatario

Paradigma Client-Server

Chiamiamo **server** il processo applicativo che per primo si mette in ascolto, mentre chiamiamo **client** quello che inizializza la connessione.

1. SERVER si mette in ascolto (Apertura passiva)
2. CLIEN cerca di inizializzare la comunicazione con un server (Apertura attiva)

> il server quando si mette in ascolto può fare binding con una porta, ovvero dice al sistema operativo a quale porta il processo è collegato.

> Lo fa anche il Client per ricevere risposte dal server

Numeri di porta

Nell'header occupano `2 byte`, quindi c'è la disponibilità di 65536 porte, da 0 a 55535, suddivise in 3 categorie:

- `0 - 1023` WELL KNOWN PORTS → queste sono associati, a livello standard, a protocolli specifici, spesso utilizzabili e assegnabili solamente con privilegi
- `1024 - 49151` REGISTERED PORTS → uso più libero, anche queste possono essere usate da altri protocolli
- `49152 - 65535` DYNAMIC or PRIVATE PORTS → dette porte alte, ad uso privato e libero

Un processo client deve sapere preventivamente la porta di destinazione, il fatto che siano standard serve ad evitare la gestione della porta in fase di connessione. Il client ha una porta "meno importante" perché il server conoscerà dinamicamente la porta del client. Ogni volta che vediamo un pacchetto che fa uso di un applicativo livello 4 quindi vedremo

```
PORTA DESTINAZIONE | PORTA SORGENTE | TIPO DI PROTOCOLLO
```

> Usiamo il termine **flusso di pacchetti** per identificare pacchetti che appartengono a una stessa comunicazione.

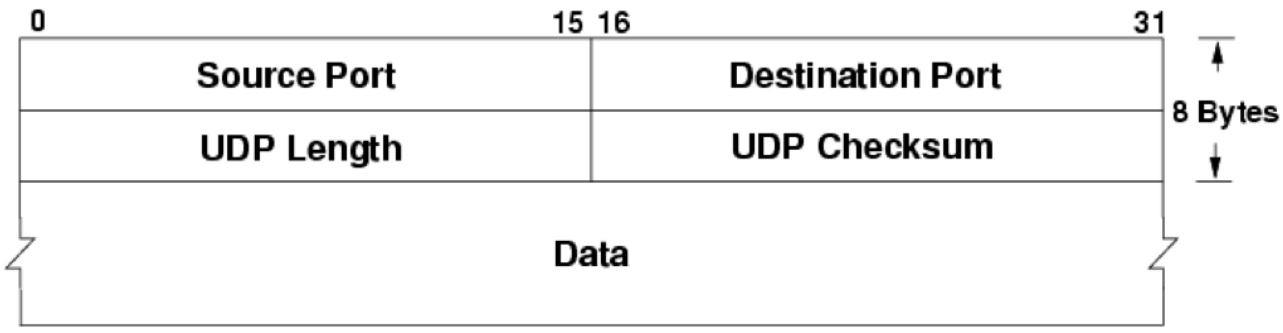
Protocollo UDP

Protocollo di trasporto leggero e "antico", avente le seguenti caratteristiche:

1. Usa le porte
2. Rileva l'errore
3. Servizio di consegna non garantito, è best-effort i pacchetti possono essere duplicati o persi
4. Servizio senza connessione

Header

è grande `8 byte`

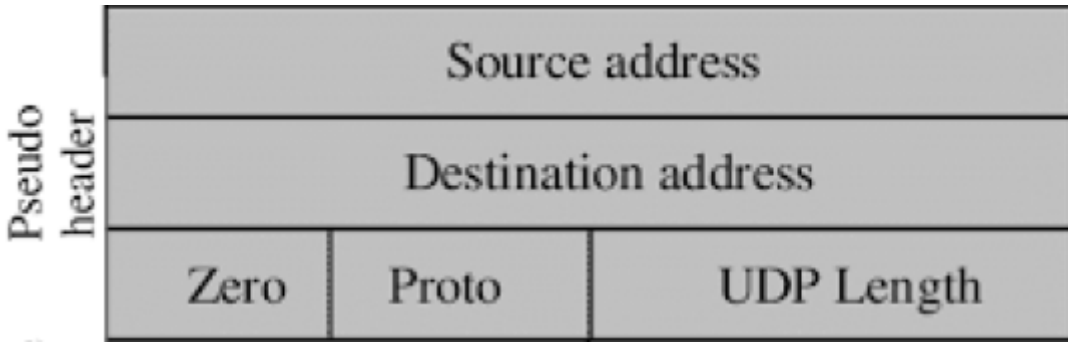


Checksum

calcolato su 3 informazioni:

1. pseudo header ip
2. header UDP
3. payload UDP

Lo pseudo Header IP è formato nel seguente modo



- proto → protocollo dell'header IP
- lunghezza → stesso del pacchetto UDP, utilizzato per facilitare il calcolo del checksum, deve essere un multiplo di 16bit L'algoritmo è una somma binaria: ne viene calcolato uno e trasmesso, all'arrivo viene ricalcolato e se risultano uguali vuol dire che non è stato presente nessun errore, altrimenti viene segnalato.

TCP

È un protocollo affidabile e ha un paradigma di comunicazione:

- orientato alla connessione
- orientato allo **stream** di dati

Affidabilità

Come avviene la trasmissione affidabile di dati su una infrastruttura inaffidabile? Creiamo una sorta di "canale virtuale", dove controlliamo e gestiamo eventuali errori. Ha la capacità di **rimediare** tramite la ritrasmissione del pacchetto.

La rilevazione dell'errore avviene tramite checksum, ma questo non ci permette di rilevare un pacchetto perso (perdite), un pacchetto duplicato (duplicati) e che la consegna avvenga in ordine. Questi errori vengono rilevati tramite altri meccanismi.

Ci offre un canale "perfetto" ma ha un costo e può risultare, a seconda del contesto, pesante. Se ci serve un protocollo intermedio si crea un protocollo a partire da UDP

Rilevare (e rimediare) alla Perdita di pacchetti

- Ogni trasmissione andata a buon fine viene notificata al mittente (**acknowledged**) dall'host ricevente. Nel caso in cui il pacchetto ricevuto sia errato, l'host ricevente non invia l'ACK, stessa cosa accade nel caso di un pacchetto duplicato → il pacchetto ha lo stesso numero di uno già ricevuto, non invio l'ACK
- Se l'host mittente non riceve un acknowledgement entro un intervallo di tempo predefinito (time-out), il mittente ritrasmette i dati

Questa logica si dice di acknowledgement positivo (viene inviato l'ACK solo in caso di trasmissione corretta). Esistono protocolli che inviano anche il NACK, implementarlo comporta costi. Il **time-out** deve essere scelto in maniera "intelligente", per questo il dimensionamento è fatto in maniera adattiva.

Orientato alla connessione

Il TCP ha 3 fasi principali:

1. Apertura della connessione
2. utilizzo
3. chiusura della connessione L'apertura della connessione prevede l'esecuzione prevede l'esecuzione di vari controlli: la disponibilità dell'altro partecipante e stabilire uno stato END-TO-END, ovvero allocare (non a livello di rete) risorse.

TCP implementa la logica di segmentazione del flusso di dati in pacchetti:

- l flusso dati viene separato in tanti **segmenti**
- Viene chiamato **segmento** il blocco di dati che TCP invia a IP per costruire il pacchetto Il TCP cerca di minimizzare la divisione dei pacchetti:
- cerca di non far frammentare ulteriormente i segmenti TCP

Affidabilità

- Le caratteristiche di affidabilità dipendono dal **paradigma di comunicazione**.
- il tcp essendo orientato alla connessione e orientato a flusso (di byte) deve prevenire l'alterazione dell'intero flusso
- L'apertura e la chiusura della connessione devono essere affidabili
- Tutti i segmenti devono arrivare a destinazione Altro modo in cui si garantisce affidabilità della frammentazione è attraverso l'identificazione univoca di ogni segmento (numeri di sequenza - **sequence number**) che permette al destinatario di:
- identificare i pacchetti e scartare i duplicati
- riordinare i pacchetti prima di inoltrarli

Funzionalità

- **trasferimento con BUFFER** gestione dei dati tramite buffer gestito dal sistema operativo (anche dal livello 4 o dalle applicazioni)
- **connessione full duplex** trasferimento contemporaneo. Una volta instaurata la connessione client e server sono alla pari
- **controllo di flusso e congestione**: regola la velocità di invio dei dati scambiati (throughput) rispetto alle capacità degli end (**Flusso**) e della rete (**Congestione**)

Non garantisce

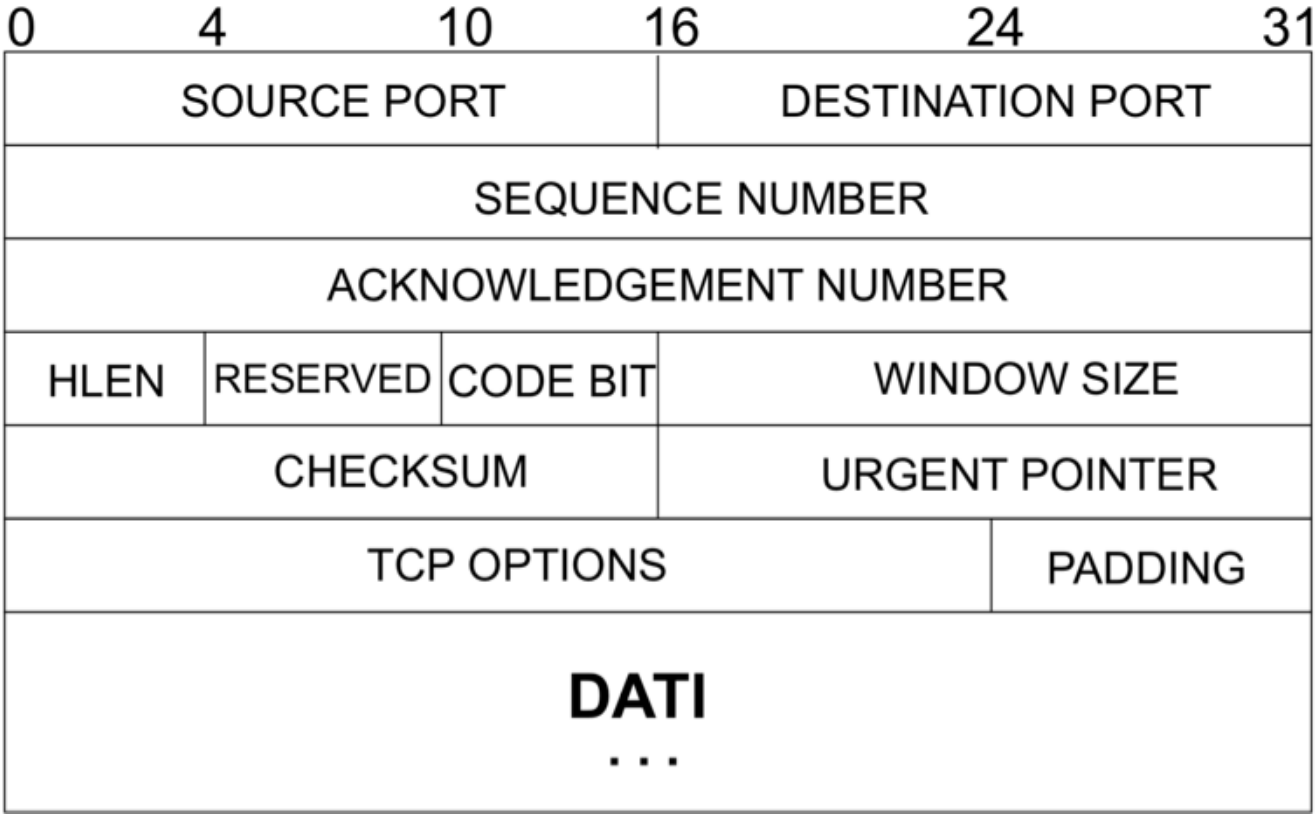
- comunicazione in tempo reale
- garanzia di disponibilità di banda tra mittente e destinatario
- Multicast affidabile

Problematiche

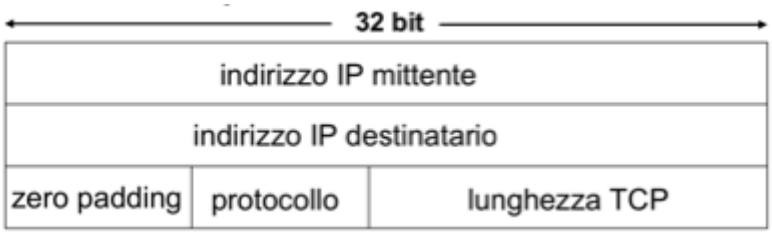
- Eterogeneità degli host e delle reti
- possibilità di ritardi nella rete

Segmento TCP

Come tutti i pacchetti è formato da un Header e da un Payload.



- Source/Destination port → Sono uguali al protocollo UDP
- Checksum → Uguale all'UDP
- **Sequence Number** (32 bit) --> numero di sequenza relativo al flusso
- **Acknowledgment Number** (32 bit) --> ACK riferito ad un numero di sequenza del flusso di byte che si sta ricevendo, questo campo c'è perchè l'ACK può anche avere dei dati, non da solo da pacchetto di conferma, questo prende il nome di *piggybacking*
- **HLEN** (4bit) --> lunghezza dell'header TCP (in multipli di 32 bit) ->20 byte come minimo
- **Reserved** (4 bit) --> per scopi ad uso futuro
- **Code BIT** (6 bit) --> definisce lo scopo del contenuto del segmento, esistono diversi "flag" che se settati indicano lo scopo del frammento:
  - URG --> urgent - inviare subito i dati
  - ACK
  - PSH -->il destinatario invia i dati subito al processo applicativo, senza aspettare che il buffer si riempia
  - SYN, FIN, RST --> usati per instaurare e per la chiusura/interruzione della connessione
- **Window Size** (16 bit) --> dimensione della finestra di ricezione (serve per il controllo di flusso)
- **Urgent Pointer** (16 bit) --> Se il bit URG è settato allora c'è la posizione dell'ultimo byte trattato in modo urgente / al primo non urgente del payload
- **TCP Option** --> pseudo header TCP più segmento TCP



Dati Urgent

Servono per trasportare caratteri speciali (come sequenze di tasti tipo CTRL+C ecc...) così che siano recapitati immediatamente, scavalcando lo stream

Opzione MMS

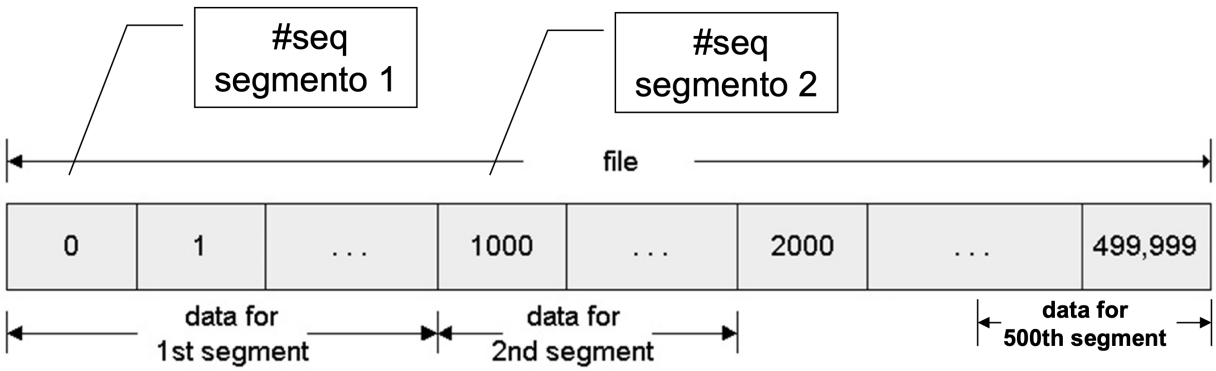
Le TCP OPTIONS consentono di negozionare il **Maximum Segment Size** per:

- garantire che il segmento entri nei rispettivi buffer
- evitare il più possibile la frammentazione H2N
- sfruttare al meglio la banda In caso di MMS troppo grande --> elevati rischi di frammentazione nell'attraversamento dei livelli dello stack sottostante

> Il valore di default è di 536 byte

Sequence Number

Non è un contatore, ma una posizione in una memoria virtuale. Non partono da 0, il punto di partenza è generato in maniera pseudo-casuale ed è chiamato **ISN** (initial Sequence Number). I successivi Sequence Number sono offset del primo byte del flusso dal mittente.



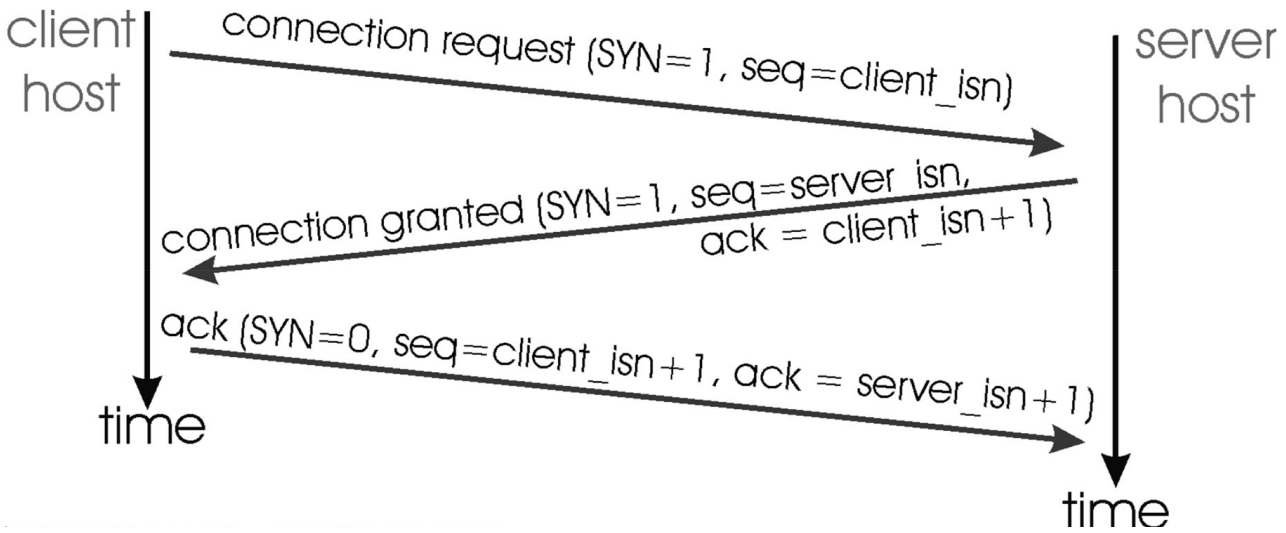
- L'intervallo dei numeri di sequenza (32bit) garantisce che lo stesso numero non sia riutilizzato prima di qualche ora.
- Grazie al TTL dei pacchetti IP, segmenti con lo stesso numero non coesisteranno sulla rete ACKNUMBER non è uguale al sequence number del pacchetto ricevuto, ma è uguale al primo byte che ci aspettiamo di ricevere

Piggy Backing

Ogni ACK può trasferire dei dati --> ci sono standard che definiscono quanto al massimo si può aspettare prima di provare a inviare ACK + DATI e non solo ACK

Instauramento e chiusura di una connessione TCP

Three way Handshake



Siamo nel paradigma client server:

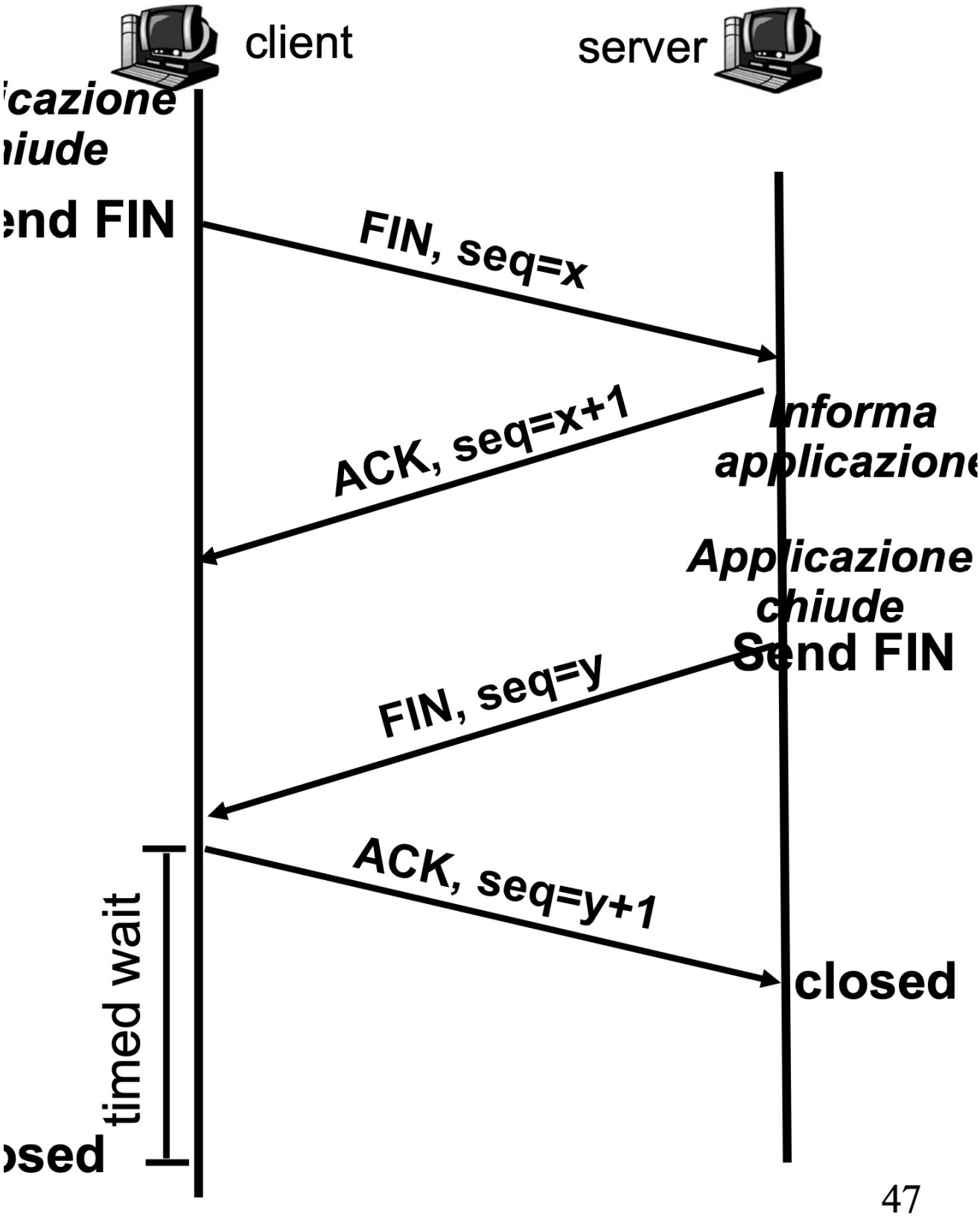
1. il client invia un pacchetto con la porta e l'ip del destinatario con settato il bit SYN e con il sequence-number uguale all'ISN
2. Se il server accetto manda al mittente un pacchetto di risposta con il bit SYN settato, il sequence-number uguale al ISN del server, e come ACK il client-ISN + 1
3. Il client invia alla ricevuta della risposta un pacchetto con il sequence-number il suo ISN + 1, ACK uguale a ISN del server + 1 e con bit SYN non settato

A questo punto la connessione TCP è instaurata. Alla fine si da per scontato che siano stati allocati le risorse necessarie. Tra le informazioni aggiuntive vengono anche indicati:

- MMS (anche il server lo comunica nella fase di risposta) --> al terzo pacchetto inviato si conosce già qual'è il minimo fra i due scambiati ed è già stato scelto
- Window Size (rispettiva prima del client e poi del server) --> ovvero il massimo numero di byte che client/server sono in grado di ricevere nel proprio buffer, necessario per la regolazione del flusso dello stream di byte

Chiusura della connessione Polite

Essendo una connessione full-duplex va chiusa da entrambe le parti:



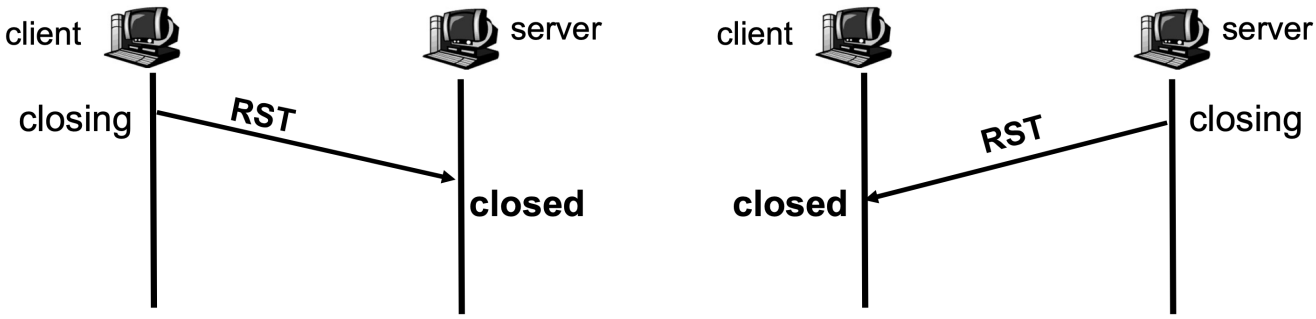
1. il client invia un segmento controllo con il bit FIN=1 al server
2. il server riceve FIN e invia ACK
3. il server chiude la connessione lato client-server ed invia FIN=1 al client
4. il client riceve il segmento FIN=1 e invia ACK
5. il server riceve ACK
6. il client attende il timeout dell'ACK inviato, allo scadere anche la connessione lato server-client viene chiusa Il Time Wait serve per verificare che l'ACK sia stato

conseganto con successo (in caso contrario il server invia nuovamente il pacchetto di FIN). Si sceglie una unità di tempo di 2 $\times$ MSL ovvero 2 volte il tempo di vita massimo di un frammento TCP all'interno della rete prima di essere scartato

Chiusura Unpolite della connessione

Normalmente la connessione viene chiusa in maniera Polite, ma talvolta si verificano delle condizione che portano ad interrompere la connessione in modo "brusco" o dal lato server o dal lato client. Il TCP offre un meccanismo per la chiusura rapida, il bit RST (**reset**).

- L'host decide il reset e pone il campo RST=1
- L'altro nodo chiude immediatamente la connessione
- Vengono rilasciate tutte le risorse utilizzate dalla connessione



Affidabilità del protocollo TCP

Per rendere affidabile il TCP si utilizzano 3 diversi meccanismi:

1. Acknowledgment
2. Time-out
3. Ritrasmissione Ogni trasmissione andata a buon fine viene notificata dall'host ricevente (ACK). Se l'host mittente non riceve un ACK entro il **time-out**, il mittente ritrasmette i dati. Questo è uno scenario denominato **Stop & Wait**

Stima del Time-out

Il Time-out deve essere maggiore del **Round Trip Time** (RTT) ovvero quanto ci impiega il pacchetto ad andare e tornare dal mittente-destinatario-mittente. Questo valore, per vari motivi, non è fisso (i cambiamenti spesso sono dovuti a fluttuazioni della rete o a cambiamenti di route): a livello ideale il time-out non ha limiti superiori, ovviamente però maggiore è più la qualità della connessione risulta scarsa.

> inizialmente era $TimeOut = \beta \times RTT_{medio}$ dove $\beta = 2$

Il calcolo quindi è sperimentale:

- **SempleRTT** --> misura del tempo trascorso dalla trasmissione del segmento alla ricezione del suo ACK (ignorando ritrasmissioni)
- **EstimatedRTT** --> media pesata per stimare RTT al tempo t

$$EstimatedRTT(t) = (1-x) \times EstimatedRTT(t-1) + x \times SampleRTT(t)$$![[Screenshot 2024-02-12 alle 1

Alla fine il tempo di time-out viene scelto con il valore del RTT stimato più un margine di errore

$$Timeout(t) = EstimatedRTT(t) + 4 \times Deviation(t)$$

dove

$$Devation(t) = (1-x) * Devation(t-1) + x \times |SampleRTT(t) - EstimatedRTT(t)|$$

Prestazioni del protocollo

- Round Trip Time (RTT) --> tempo di andata-ritorno
- **Tempo di propagazione** --> $\frac{RTT}{2}$
- **Rate** --> trasmission Rate (throughput) = capacità di trasmissione della rete espressa in Kbps ecc..
- **L(ptk)** --> lunghezza del pacchetto [Kbit, ... , Gbit]
- **Tempo di trasmissione** -> $\frac{L(ptk)}{Rate}$
- **Tempo di trasferimento pacchetto** -> $[\frac{RTT_{ptk}}{2} + \frac{L(ptk)}{Rate}] + [\frac{RTT_{ack}}{2}]$

Utilizzano un algoritmo di stop & wait le prestazioni risultano scarse. Per incrementare basta che si mandino contemporaneamente più segmenti (utilizzano una tecnica di pipelining) senza superare il throughput. Quello che fa il TCP è capire quanti segmenti e quanto grandi inviarli prima di aspettare un ACK.

> La dimensione del segmento è calibrata in base al PATH-MTU

Sliding Window

Per implementare un algoritmo di pipelining servono 3 requisiti:

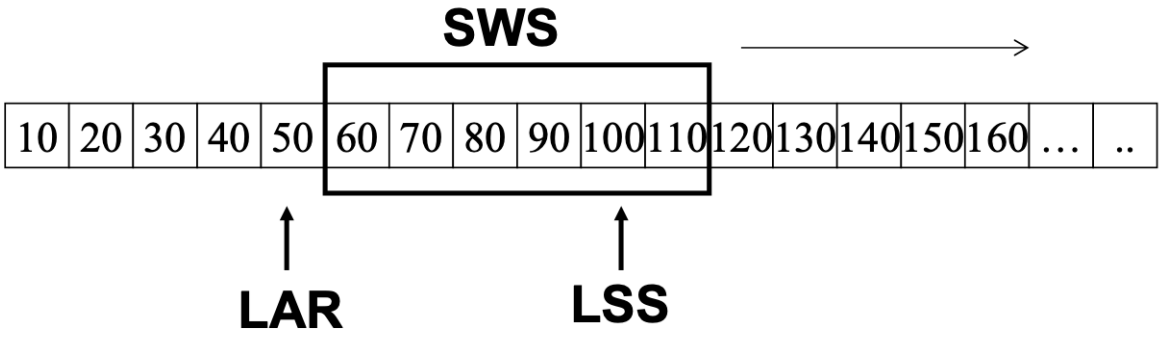
1. Buffer lato mittente --> per mantenere i dati inviate e di cui non ha ancora ricevuto l'ack
2. Buffer lato destinatario --> per mantenere le sequenze di dati dove non tutti i dati sono arrivati o sono arrivati correttamente
3. **Finestra di scorrimento** (Sliding window) --> denota il numero massimo di dati che il mittente può inviare senza aver ricevuto un ACK dal destinatario

Mittente

Per gestire la sliding window il mittente utilizza 3 variabili:

- Dimensione della finestra di invio SWS (Sender windows Size): indica il limite superiore peri l numero di segmenti che il mittente può inviare in pipeline senza aver ricevuto un ACK
- Numero di sequenza dell'ultima conferma ricevuta LAR (Last ACK recived)
- Numero di sequenza dell'ultimo segmento inviato LSS (Last segment sent) Queste tre variabili devono soddisfare la seguente relazione:

$$LSS - LAR \leq SWS$$



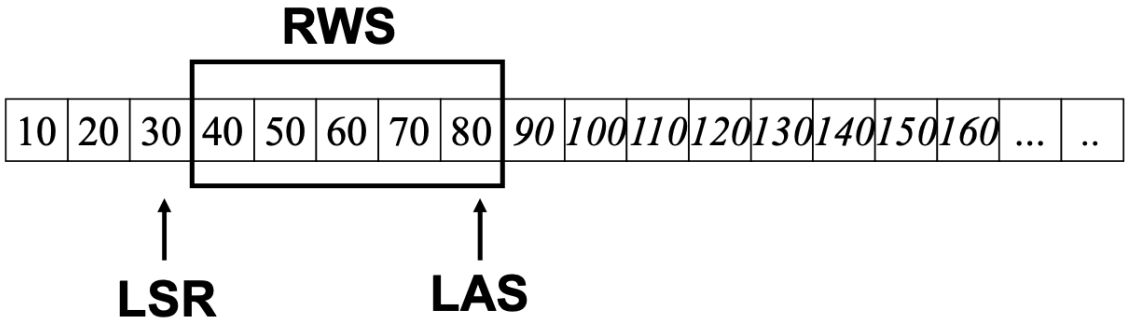
- Si possono inviare tutti i segmenti dentro la window size.
- Quando arriva un ACK il mittente sposta LAR verso destra
- Il time-out è relativo a ciascun segmento, non a tutta la window

Destinatario

I destinatari che gestiscono una slinding window utilizzano 3 variabili:

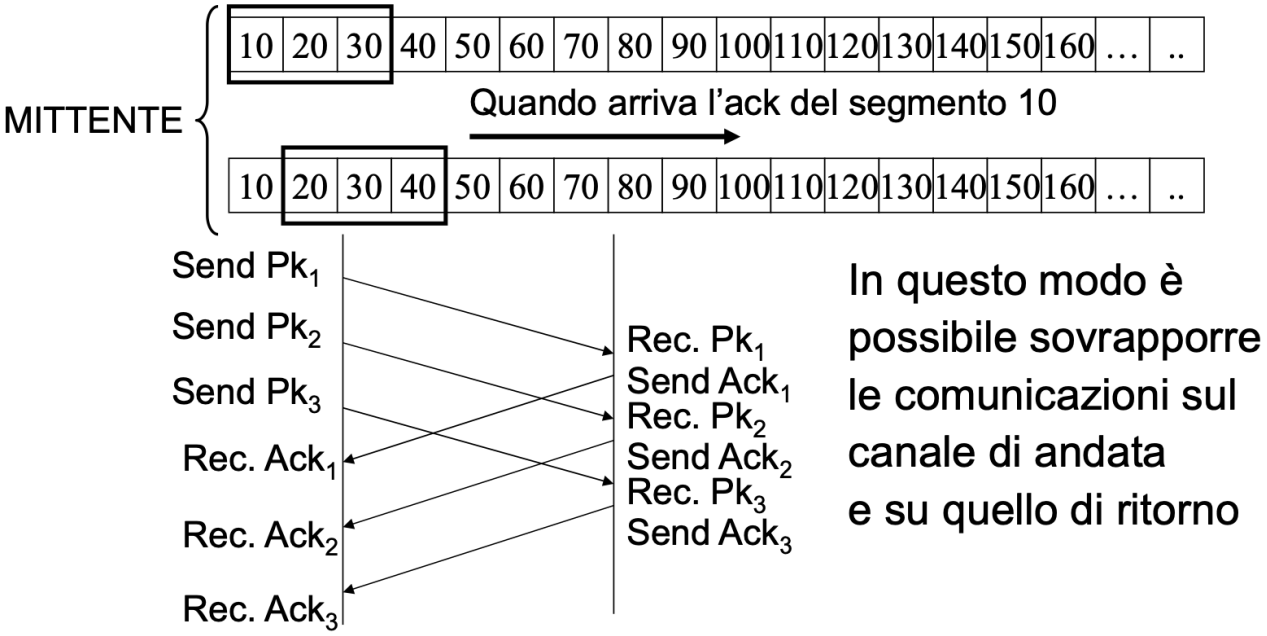
- Dimensione della finestra di ricezione `RWS` (Recive window size): indica il limite superiore dei segmenti "fuori frequenza" che il destinatario può accettare
- Numero di sequenza del segmento accettabile più elevato `LAS` (Largest Acceptable Segment)
- Numero di sequenza dell'ultimo segmento ricevuto "in sequenza" `LSR` (Last segment Recived) Queste tre variabili devono soddisfare la seguente relazione

$$LAS - LSR \leq RWS$$



Quando arriva un segmento con numero di sequenza `NUM_SEG` , il destinatario agisce come segue:

- se `NUM_SEG ≤ LSR` (pacchetto duplicato) oppure (per errore) `NUM_SEG > LAS`, significa che il segmento si trova al di fuori della finestra utile del destinatario e viene scartato
- se `LSR < NUM_SEG ≤ LAS`, il segmento si trova all'interno della finestra del destinatario e viene inserito nel buffer



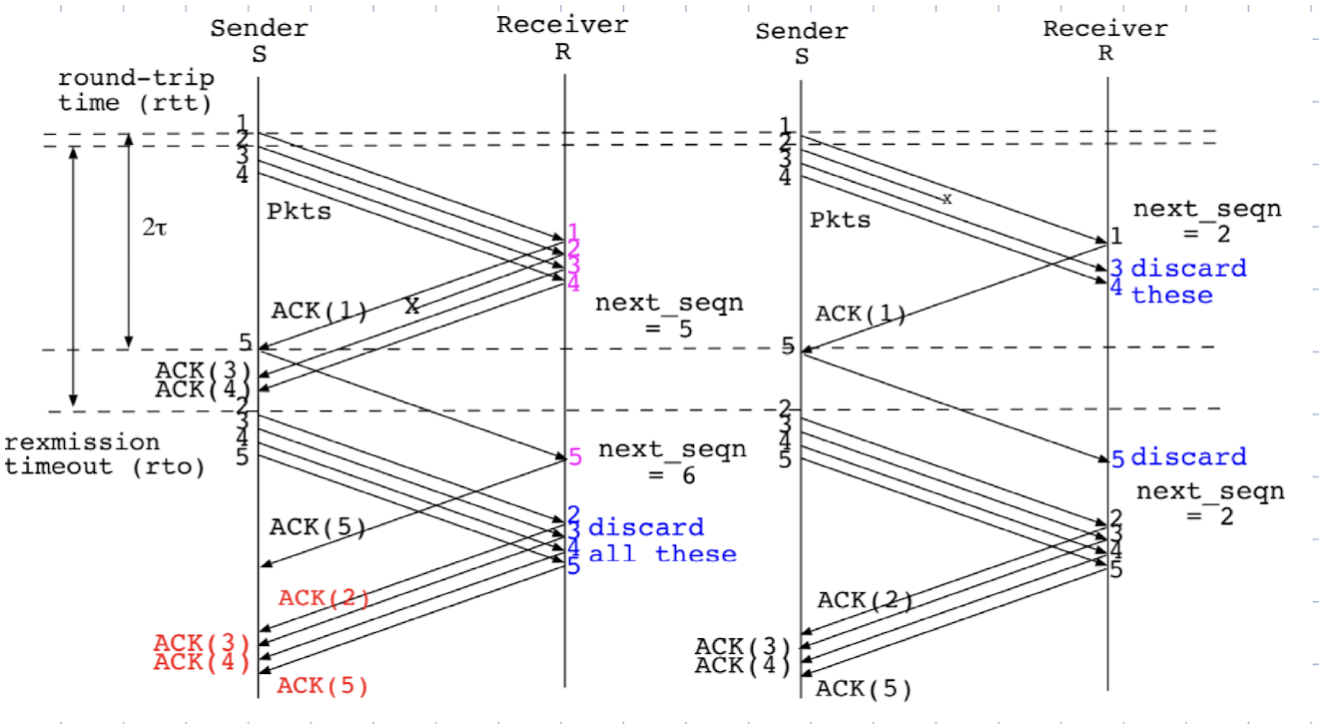
> La dimensione della window Size non è fissa

La dimensione del buffer del mittente deve essere ≤ della capacità di rete ≤ buffer destinatario

Go-back-N

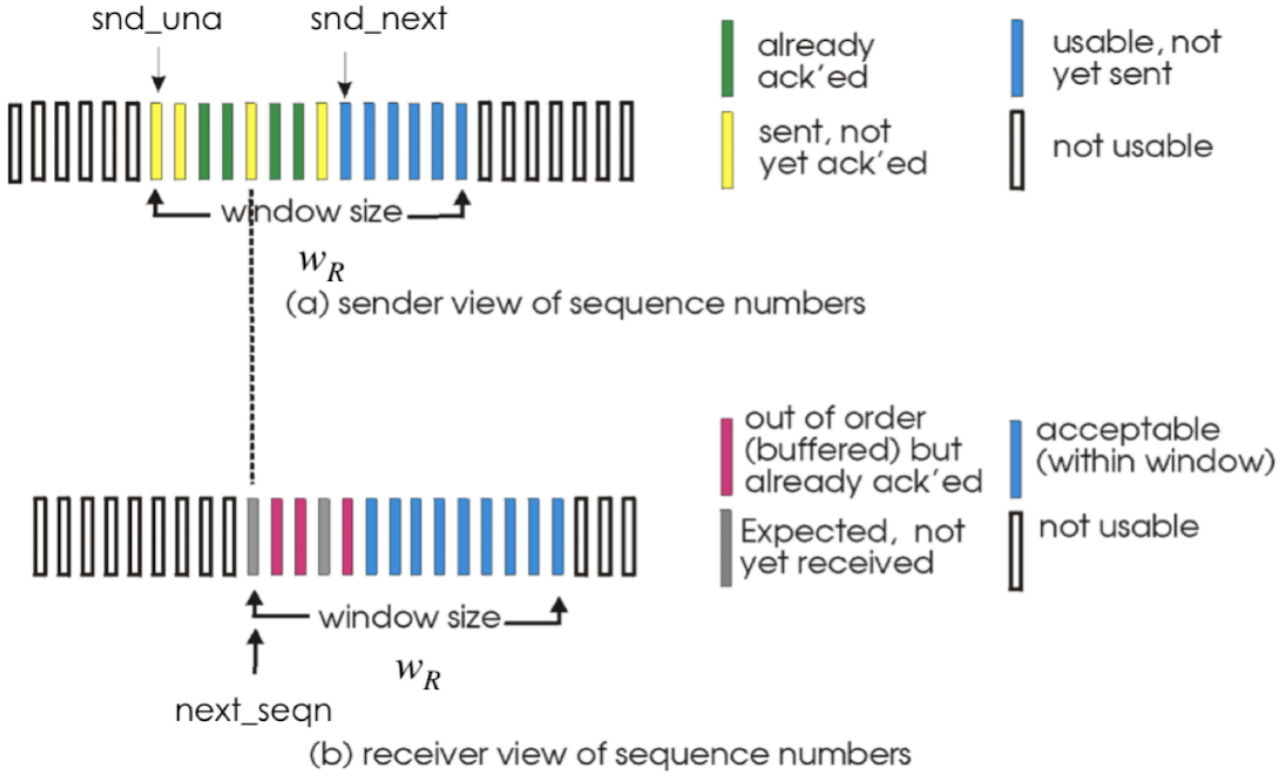
Questo è un algoritmo per la ritrasmissione di pacchetti quando si fa uso di sliding window: **Ipotesi**: finestra di massimo N segmenti

- Il destinatario manda l'ACK dei soli pacchetti ordinati --> l'ultimo pacchetto in ordine ricevuto (si fa uso anche dell'ACK comulativo --> se invio ACK 4 allora i pacchetti prima sono intrensicamente arrivati correttamente)

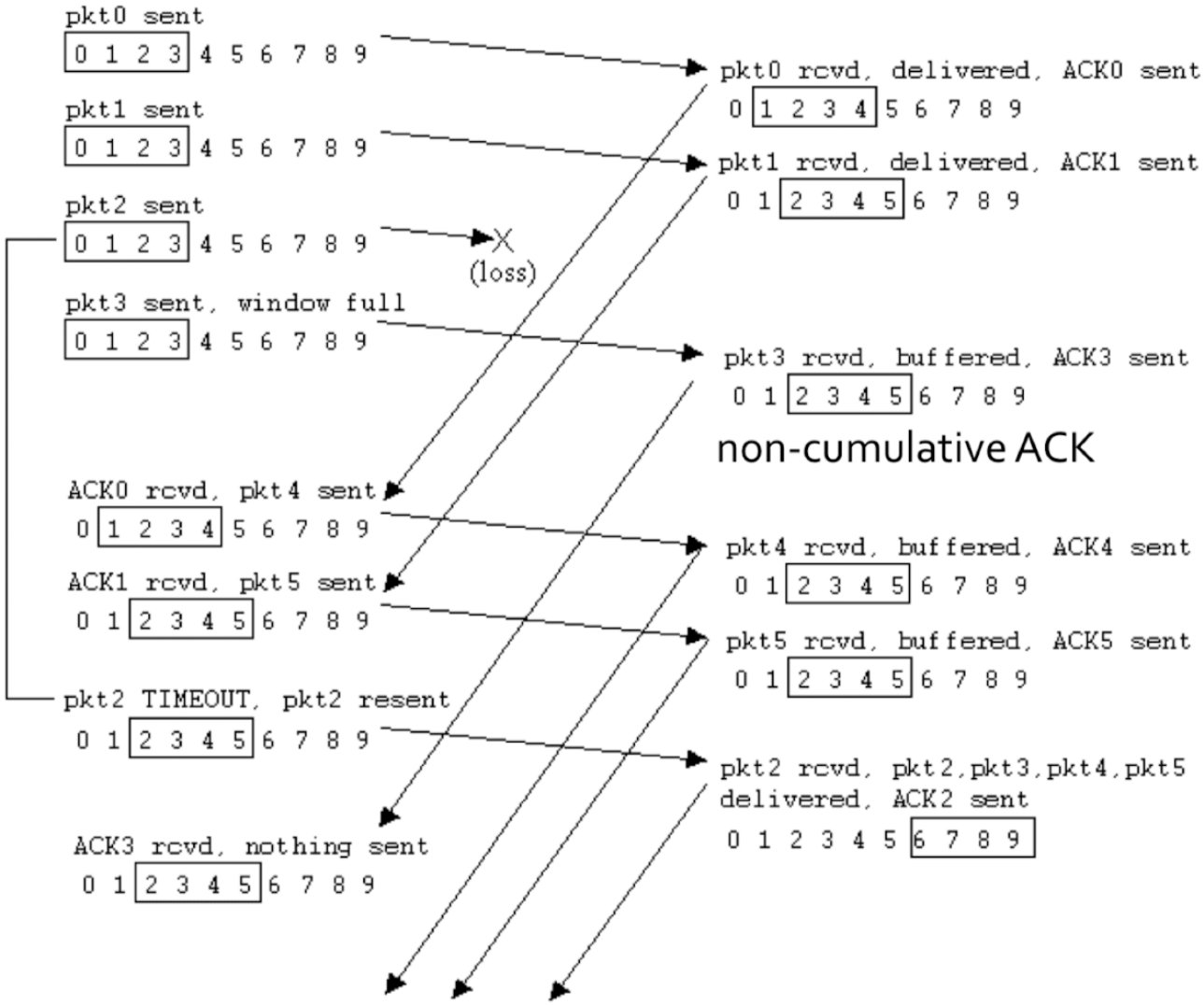


Ritrasmissione Selettiva

il mittente riceve una ACK selettiva, quando scade il time-out viene inviato nuovamente solo quel pacchetto. L'ack selettivo viene inviato per ogni pacchetto ricevuto (anche non in ordine)



> con l'ack comulativo c'è meno probabilità di errore in caso di perdita del'ack



Sliding Window nel TCP

Attualmente utilizza un ACK ibrido --> usa un ACK accumulativo: accetta pacchetti guori ordine ma non ne modifica la ricezione

La window size non è fissa:

- Congestion Window: dimensione della finestra di congestione, derivate dalle capacità della rete
- Flow Window: dimensione della finistra di flusso derivante dalla dimensione del buffer di destinazione La dimensione effettiva è quindi la minima tra la Congestion window e la Flow window

Il tcp è fairness --> non genera overhead inutile. In caso di connessioni multiple (e senza altri colli di bottiglia) e di risorse condivise, le bande utili saranno condivise equamente.

Controllo di Flusso

Funziona sulla base di informazioni inviate esplicitamente attraverso il campo header denominato Window Size:

- nel caso di invio dell'ACK il destinatario indica nel campo window-size lo spazio restante (=libero) del buffer di destinazione --> questa dimensione è usata dal

mittente per ridimensionare la sua window in modo da non far scartare pacchetti
- il buffer si può riempire perchè l'applicazione non è detto che consumi i dati in arrivo, questo perché il TCP è asincrono
  - in questo caso si manda uno 0 e si smettono i inviare dati utili. Viene inviato un pacchetto periodicamente di **POLLING** con cui il destinatario potrà segnalare nuovamente la dimensione del buffer (per ricominciare a inviare dati)

Controllo della congestione

Gli effetti della congestione sono:

- perdita di pacchetti (buffer dei router pieni)
- pacchetti scartati

La congestione generalmente viene gestita in due modi:

- **Controllo congestione assistito** --> nel tos è presente un bit ECN che i router mandano in caso di congestione
- **Controllo congestione end-to-end** --> la rete non indica il suo livello di congestione, ma viene regolato dal mittente e dal destinatario

Soluzione del TCP

1. Controllo della congestione end-to-end
2. slow start --> cominciamo piano ed aumentiamo
3. AIMD --> Additive-Increase, Multiplies-Divide

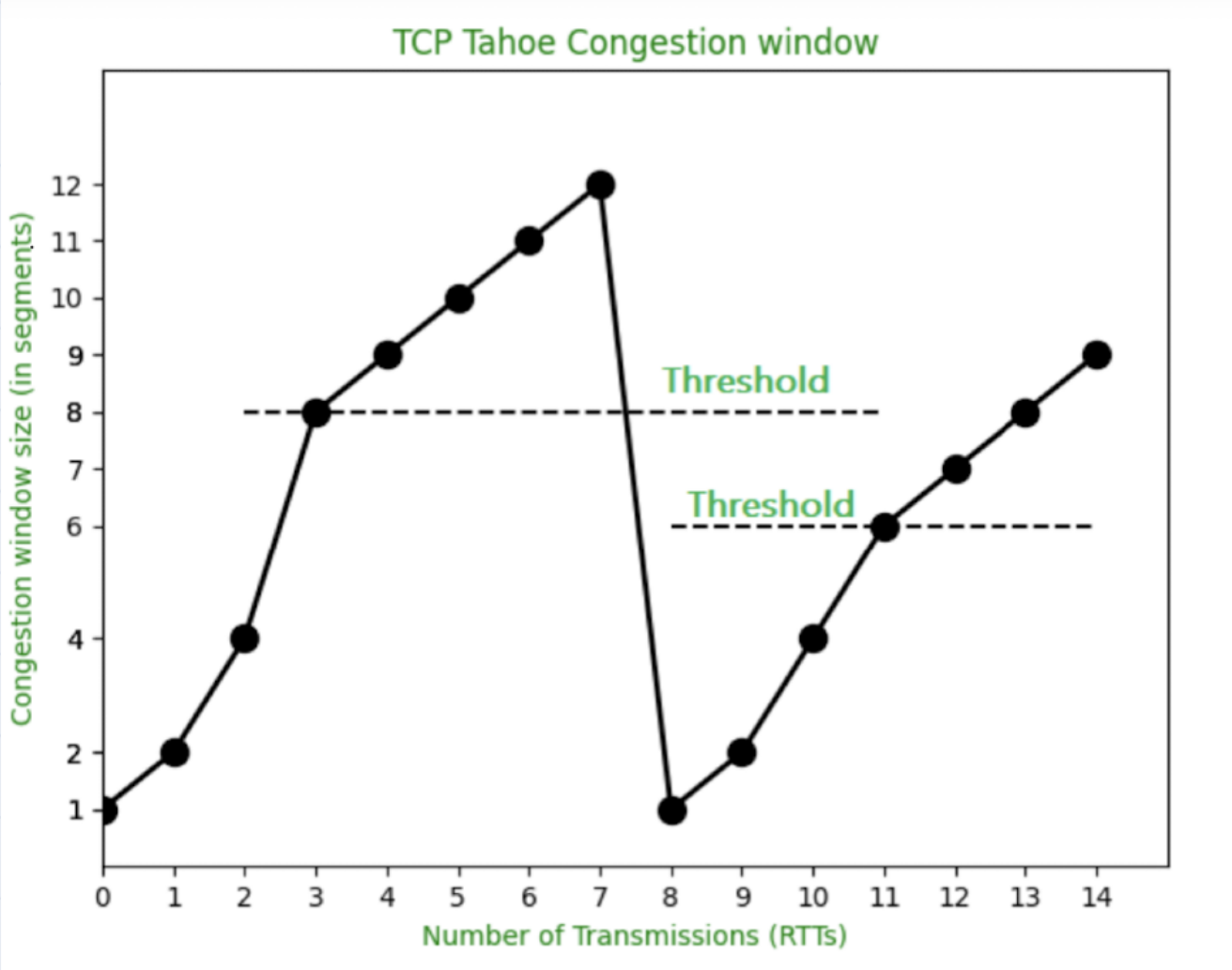
> Solitamente nel TCP si inizia esponenzialmente e pian piano si aumenta linearmente

Slow start

```
Inizializa CW=1
for (ogni segmento ACK)
       CW = CW * 2
until (TIMEOUT or (CW>threshold))
```

Algoritmo di Tahoe

```
if not (perdita):
       ogni w segmenti ACK:
              CW ++
else
       threshold = congestion_window / 2
       CW = 1
```



In alcune versioni del TCP al posto del NACK si fa utilizzo dell'ACK duplicato: Nel caso di un triplo ACK si considera come un NACK e si procede con il reinvio

> i timeout e i 3-ACK vengono usati per ridimensionare la finestra di invio, ma il 3-ACK spesso è gestito come meno grave rispetto ad un TIMEOUT

ad esempio nel TCP-RENO: con 3-ACK la CW viene dimezzata, mentre con un time-out viene rimessa a 1