

Reti di telecomunicazioni

saverio

giugno 2025

Contents

1	Livello 4 trasporto TCP/UDP	2
1.1	Nozioni preliminari	2
1.1.1	Sock e ports	3
1.2	UDP - datagram	4
1.2.1	Calcolo checksum	6
1.3	TCP - segmento	7
1.3.1	Apertura connessione client-server (3way handshake) . . .	9
1.3.2	Chiusura connessione	10
1.3.3	TCP client - sequenza di stati?	12
1.3.4	TCP server - sequenza di stati?	12
1.3.5	Trasmissione dei segmenti - sliding window	12
1.3.6	Finestre di trasmissione e ricezione	13
1.3.7	Controllo di flusso	14
1.3.8	Perdita di segmenti	14

Chapter 1

Livello 4 trasporto TCP/UDP

1.1 Nozioni preliminari

UDP(Used Datagram Protocol) e TCP(Transmission Control Protocol) sono i protocolli di trasporto(livello 4) più diffusi. Principali differenze:

- TCP è orientato alla connessione, UDP no (vuol dire che non è necessario stabilire una connessione prima di inviare i dati, cosa invece necessaria con TCP)
- TCP è un protocollo di trasporto affidabile, UDP è non affidabile(quindi TCP garantisce che i dati arrivino a destinazione e nell'ordine corretto, mentre UDP non lo fa)

I pacchetti(PDU) TCP sono chiamati segmenti, mentre i pacchetti UDP sono chiamati datagrammi.

Funzioni principali UDP Svolge l'unico compito di incapsulare i dati dell'applicazione in un pacchetto UDP, aggiungendo le informazioni necessarie per la consegna(Multiplexing). Non fornisce alcuna garanzia di consegna o ordine dei pacchetti.

Funzioni principali TCP

- Controllo flusso end to end
- Controllo congestione end to end
- Ritrasmette le PDU perse o corrotte
- Riordina i segmenti ricevuti in ordine corretto

Il TCP numera i singoli byte che arrivano dal livello applicativo, e non i segmenti; è fondamentale poichè dal livello 3(IP) non arrivano i byte in ordine, ma i pacchetti possono arrivare in ordine sparso.

1.1.1 Sock e ports

Dei concetti fondamentali per il funzionamento di TCP e UDP sono i socket e le porte.

- Socket: è un punto finale di una connessione di rete, rappresentato da:
 - Indirizzo IP: identifica un host sulla rete
 - Porta: identifica un'applicazione in esecuzione su quell'host

I socket sono utilizzati per inviare e ricevere dati tra applicazioni su host diversi.

- Porta: è un numero che identifica un'applicazione in esecuzione su un host. Le porte sono numerate da 0 a 65535 e sono suddivise in tre categorie: well-known ports, registered ports e dynamic/private ports.

Il numero di porta è un numero a 16 bit che identifica un'applicazione in esecuzione su un host. Quando un client invia una richiesta HTTP a un server web, utilizza la porta 80 per comunicare con il server. Il server ascolta sulla porta 80 e risponde alla richiesta del client.

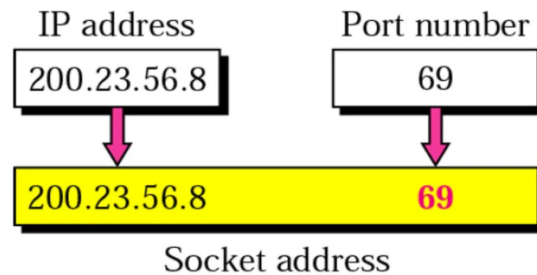


Figure 1.1: Relazione tra socket, indirizzo IP e porta

Le porte sono numerate da 0 a 65535 e sono suddivise in tre categorie:

- Well-known ports: da 0 a 1023, utilizzate da applicazioni di sistema e protocolli standard (es. HTTP su porta 80, HTTPS su porta 443, DNS(traduce il nome "simbolico" del servizio web per il web server) su porta 53)
- Registered ports: da 1024 a 49151, utilizzate da applicazioni registrate presso l'IANA
- Dynamic/Private ports: da 49152 a 65535, utilizzate per connessioni temporanee e dinamiche

Tramite i numeri di porta è possibile identificare a livello 4 con quale applicazione sto inviando/ricevendo dati

Come identificare un flusso dati in internet?

- Numero di porta locale
 - Indirizzo IP host mittente
 - Numero di porta remota
 - Indirizzo IP host destinatario
 - Tipologia di Protocollo di Trasporto
- } **IP flow**

Figure 1.2: Identificazione di un flusso dati tramite indirizzi IP e numeri di porta (tuple a 4 campi)

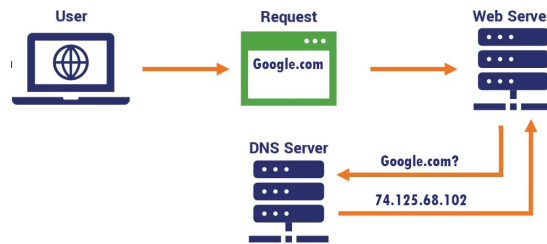


Figure 1.3: Esempio di funzionamento del DNS: risoluzione di un nome simbolico in indirizzo IP, tramite DNS server

1.2 UDP - datagram

è il più veloce del TCP poichè non esegue molte funzioni come l'ordinamento dei pacchetti i pacchetti sono da 32 bit, ossia 4 byte:

- datagram: 16 bit
- pseudoheader: 16 bit

Si occupa di fare multiplexing(trasmissione) e demultiplexing(ricezione) dei dati, ovvero incapsula i dati dell'applicazione in un pacchetto UDP e aggiunge le informazioni necessarie per la consegna. Viene usato soprattutto per applicazioni in tempo reale, come servizi streaming live.

Struttura del datagram UDP

- Source port(indirizzo IP del mittente): 16 bit
- Destination port(indirizzo IP del destinatario): 16 bit

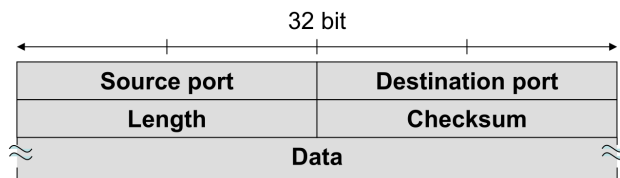


Figure 1.4: Struttura del datagram UDP

- Length (dimensione del datagram): 16 bit, quindi può rappresentare valori da 0 a $2^{16} - 1 = 65535$ bytes, ma il campo include anche l'header, quindi la dimensione massima di un datagram UDP è 65535 byte.
- Checksum: 16 bit, utilizzato per verificare l'integrità dei dati del datagram. Il checksum è calcolato su un pseudoheader che include informazioni sull'indirizzo IP del mittente e del destinatario, oltre alla lunghezza del datagram e al protocollo utilizzato (UDP in questo caso). Il pseudoheader non viene trasmesso, ma viene utilizzato solo per il calcolo del checksum.
- Data: dimensione variabile, contiene i dati dell'applicazione

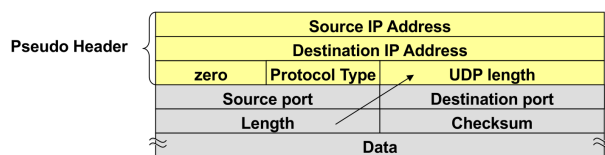


Figure 1.5: Pseudoheader UDP utilizzato per il calcolo del checksum

Nello pseudoheader sono presenti informazioni come il campo Protocol Type viene preso direttamente dal livello 3(protocollo IP).

Il checksum è la somma di controllo, è una tecnica di verifica veloce di controllo errore, non è affidabile al 100%, è comunque ottimo perchè è veloce

- Se il checksum è 0, significa che non ci sono errori, non affidabile.
- Se il checksum è diverso da 0, significa che ci sono errori, sicuramente(condizione necessaria).

1.2.1 Calcolo checksum

Il valore del checksum prima della trasmissione è impostato a 0.

In trasmissione In trasmissione vengono sommate in binario tutte le righe (mezzo righe da 16 bit) del datagram, faccio il complemento ad 1 (NOT) di questo risultato. Questo è il checksum inviato nel datagram di trasmissione.

In ricezione In ricezione viene ricevuto il datagram, con all'interno il checksum calcolato precedentemente in trasmissione.

Viene calcolato nuovamente il checksum del datagram ricevuto, sommando in binario tutte le righe (mezzo righe da 16 bit) del datagram, e sommandolo al checksum ricevuto.

Se il risultato è diverso da 0 significa che i due valori calcolati in ricezione e trasmissione sono differenti, perciò c'è un errore.

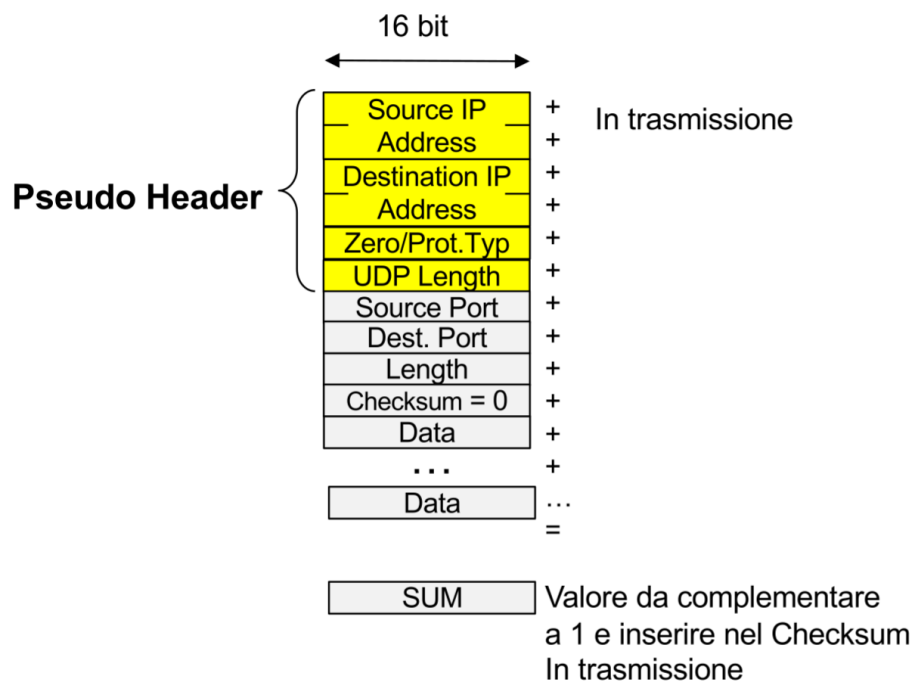


Figure 1.6: Esempio di calcolo del checksum UDP

1.3 TCP - segmento

Nei processi TCP viene utilizzata una coppia di socket per identificare un flusso di dati. Ogni socket è identificato da una tupla a 4 campi:

- Indirizzo IP del mittente
- Porta del mittente
- Indirizzo IP del destinatario
- Porta del destinatario

La connessione è full duplex → comunicazione affidabile, se qualcosa va storto chiede la ritrasmissione del pacchetto errato.

Il tcp è un protocollo greedy, ingordo, tende a prendere tutta la banda di cui necessita; questo protocollo non evita le congestioni ma le provoca, però quando le provoca cerca di risolverle.

L'affidabilità di TCP sta nei riscontri cumulativi che gli permettono di gestire le seguenti problematiche:

- Controllo di congestione e2e: regolazione del rate di trasmissione così da utilizzare completamente la banda disponibile evitando che la rete collassi
- Controllo del flusso e2e: regolarizza il rate di trasmissione per evitare la saturazione del buffer di ricezione

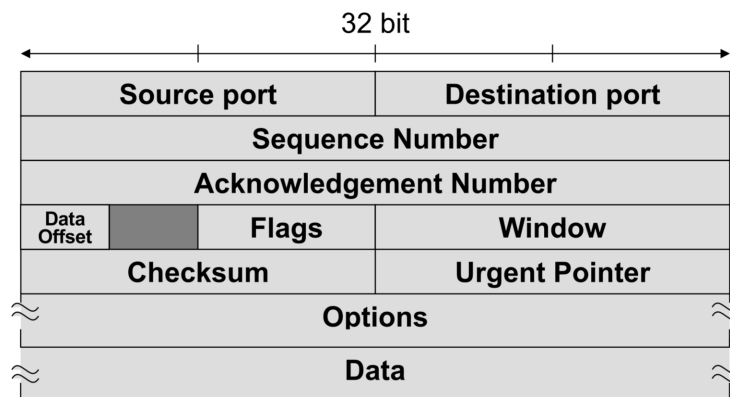


Figure 1.7: Struttura del segmento TCP

I pacchetti in tcp vengono chiamati segmenti, così strutturati:

- source port/destination port: 16 bit ciascuna e identificano il mittente e il destinatario del segmento

- sequence number: 32 bit, numero di sequenza del primo byte del segmento, utilizzato per ordinare i segmenti ricevuti ; i byte del livello applicativo (che costituiranno il payload dei diversi segmenti trasmessi) sono numerati in modo continuo partendo da un valore casuale(zero relativo).

Nel campo Seq. Number si inserisce poi il numero di sequenza (di questa numerazione continua) relativo al primo byte contenuto nel campo Data

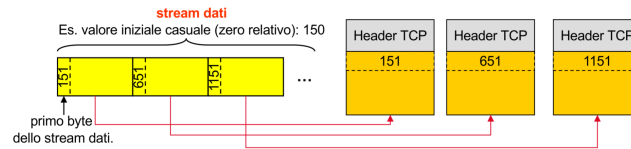


Figure 1.8: Esempio di utilizzo del campo Sequence Number nei segmenti TCP

- acknowledgment number: 32 bit, nel campo ack.number si inserisce il numero di sequenza del primo byte che ci si aspetta di ricevere (quindi il numero di sequenza del primo byte che non è stato ricevuto), si usa per capire fino a che punto il segmento è stato ricevuto correttamente. Così facendo so anche da che punto riprendere la trasmissione dei dati.(è utile a dare un riscontro di ciò che mi sta inviando il mittente, tenendolo aggiornato sulla correttezza di ciò che mi ha inviato, connessione full duplex appunto); può essere anche un campo vuoto, 0.
- data offset: 4 bit, indica la lunghezza dell'header del TCP
- flags: 6 bit, sono dei flag di controllo che indicano lo stato della connessione TCP:
 - URG: indica se il segmento contiene dati urgenti.
 - ACK: acknowledgment number
 - PSH: push, richiesta di invio immediato dei dati al livello applicativo
 - RST, SYN, FIN: reset, synchronize, finish della connessione
- window: è un campo utilizzato nella gestione del controllo di flusso, indica la dimensione della finestra di ricezione, ovvero la quantità di dati che il mittente può inviare prima di ricevere un acknowledgment dal destinatario. La dimensione della finestra può variare durante la connessione in base alla disponibilità di buffer del destinatario.
- urgent pointer: indica i dati che il ricevitore deve elaborare per prima
- checksum: è come in UDP
- options: almeno 32 bit, è opzionale, può contenere info di configurazione del segmento TCP

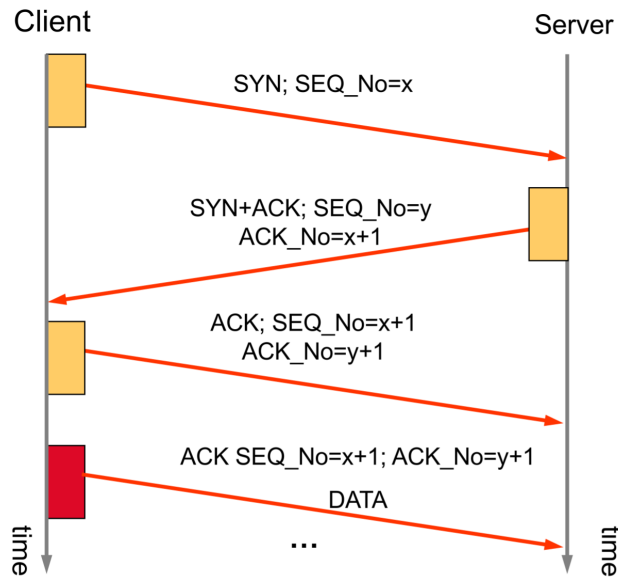


Figure 1.9: Three-way handshake: stabilimento della connessione TCP tra client e server

1.3.1 Apertura connessione client-server (3way handshake)

La connessione viene sempre aperta dal client.

Quando ricevo un segmento ho bisogno di sapere in che modo sta contando i byte il mittente;

Il client sceglie lo zero relativo, ossia il numero di sequenza(sequence number scelto casualmente, $SEQ.NUMBER = x$) da cui partire per numerare i byte che invierà al server, inviando un segmento vuoto in cui setta il flag $SYN = 1$, ($ACK = 0$ all'inizio della comunicazione) così da iniziare la comunicazione.

A questo punto il server risponde alla richiesta del client, inviando:

- un segmento vuoto con $SYN = 1$ e $ACK = 1$ (questo flag a 1 fa capire al client che il campo ACKNUMBER è stato modificato), in cui il numero di sequenza è $SEQ.NUMBER = y$ (per i dati che il server invierà, non dimenticare che la comunicazione è full duplex) e il numero di ack è $ACK.NUMBER = x + 1$ (il server ha ricevuto la richiesta del client, quindi il numero di ack è incrementato di 1)
- un segmento con i dati richiesti dal client, in cui il numero di sequenza è $SEQ.NUMBER = y + 1$ e il numero di ack è $ACK.NUMBER = x + 1$

Per terminare la "conversazione" il client invia un segmento(vuoto, ossia senza dati) con il flag SYN a 0, e conferma la corretta ricezione del messaggio precedente (y) con l'ACK.NUMBER ricevuto dal server più di 1(y + 1).

Quello in rosso nell'immagine è un ulteriore segmento, uguale al precedente, ma "pieno", ossia con i dati da inviare.

1.3.2 Chiusura connessione

La chiusura della connessione può essere avviata da uno dei due host, il client o il server (tipicamente la chiude il client).

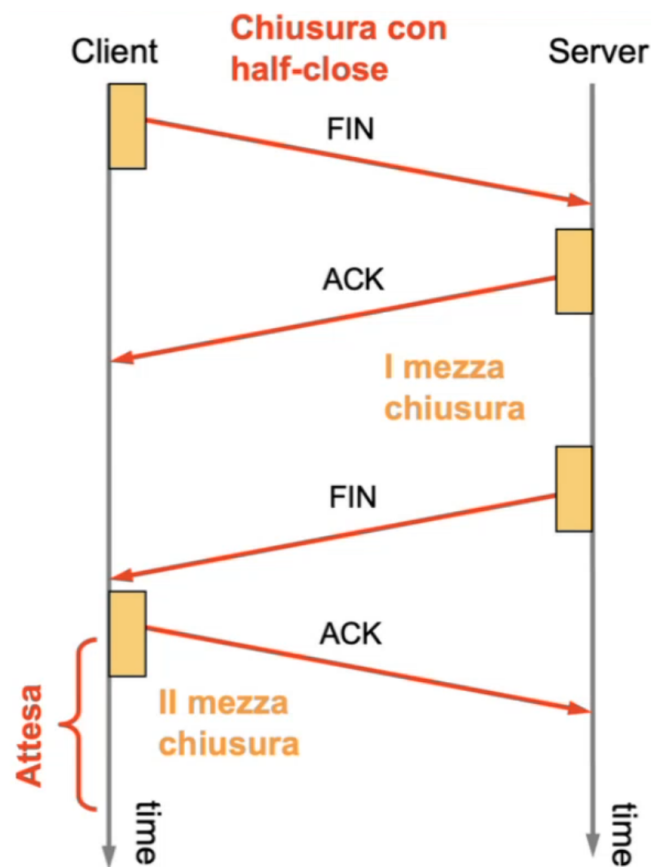


Figure 1.10: Chiusura della connessione TCP tramite half-close

Chiusura con half-close La chiusura della connessione avviene tramite due mezze chiusure:

- Il client invia un segmento con il flag FIN a 1, per indicare che non invierà più dati al server. (I mezza chiusura)
- Il server continua ad inviare i dati finché non termina e invia un segmento con flag FIN settato. (II mezza chiusura)

- Il server invia un segmento con il flag FIN a 1 e il flag ACK a 1, per indicare che non invierà più dati al client.
- Il client risponde con un segmento con il flag ACK a 1, per confermare la ricezione della richiesta di chiusura del server.

Chiusura con 3way handshake in risposta al segmento FIN inviato dal client, il server invia un unico segmento con FIN+ACK settati (e con eventualmente gli ultimi dati).

Questa soluzione non prevede che il server invii ulteriori dati e il client risponde con il segmento di ACK di chiusura della connessione.

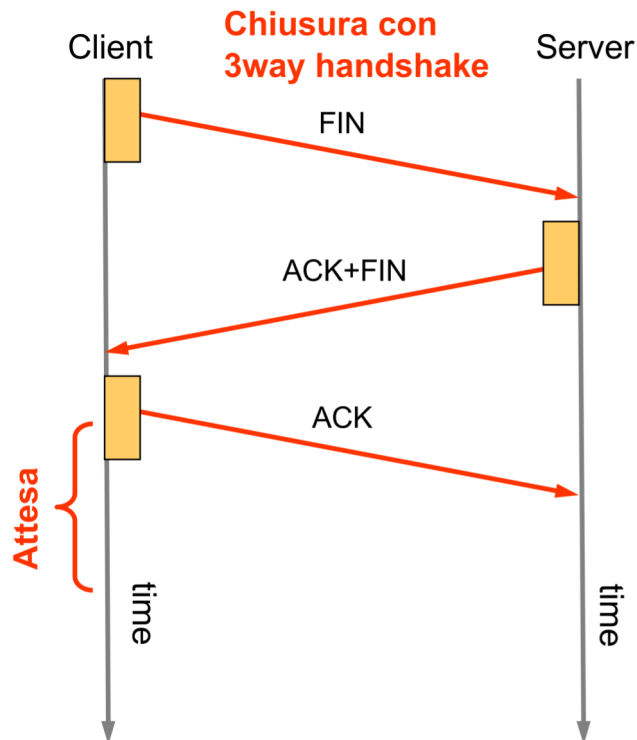


Figure 1.11: Chiusura della connessione TCP tramite 3-way handshake

TIMEWAIT in entrambe le tecniche (con half-close o 3way handshake) dopo il segmento di ACK si avvia un timer (Time Wait) prima della definitiva chiusura della connessione. Può succedere, infatti, che il segmento ACK vada perso e il server ritrasmetta il suo FIN. Il client deve quindi essere ancora in grado di poter ritrasmettere l'ACK finale.

1.3.3 TCP client - sequenza di stati?

1.3.4 TCP server - sequenza di stati?

1.3.5 Trasmissione dei segmenti - sliding window

Il TCP usa un meccanismo di trasmissione dei segmenti basato su finestre scorrevoli, finestra di trasmissione e finestra di ricezione; in base alla loro larghezza si possono inviare più o meno bytes.

Attua "self-clocking", ossia il mittente regola la velocità di invio dei segmenti in base alla velocità di ricezione del destinatario. Non c'è una velocità fissa delle sliding windows.

Round trip time - RTT

Quando invia un segmento, il mittente deve attendere un certo tempo prima di ricevere un (riscontro)acknowledgment dal destinatario. Questo tempo è chiamato round trip time (RTT) e rappresenta il tempo necessario per inviare un segmento e ricevere l'ACK.

Data W , dimensione della finestra di trasmissione, e RTT, il tempo di round trip, la velocità media di trasmissione dei segmenti è data da:

$$\text{rate medio} = \frac{W}{RTT} \quad (1.1)$$

Come migliorare questo rate?

Si può agire solo sulla dimensione della finestra W , non sul RTT.

1.3.6 Finestre di trasmissione e ricezione

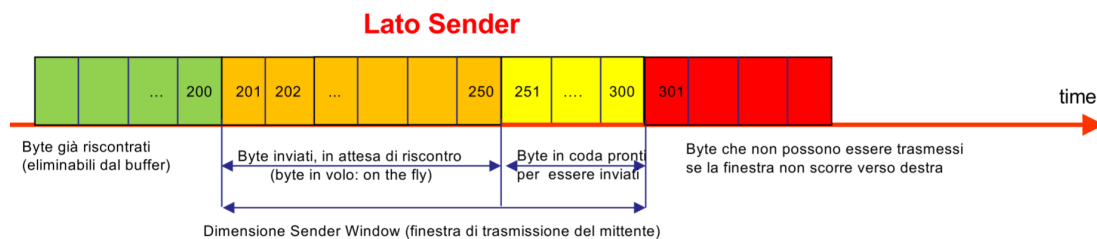


Figure 1.12: Esempio di finestra di trasmissione

Sono gli ACK ricevuti che fanno scorrere la finestra. Se ricevo l'ACK 240 allora la finestra scorre fino a 240 (riscontri cumulativi, non devo dire quelli 201, 202 ecc..., ma mi basta 240 per scorrere anche tutti i byte precedenti a 240).

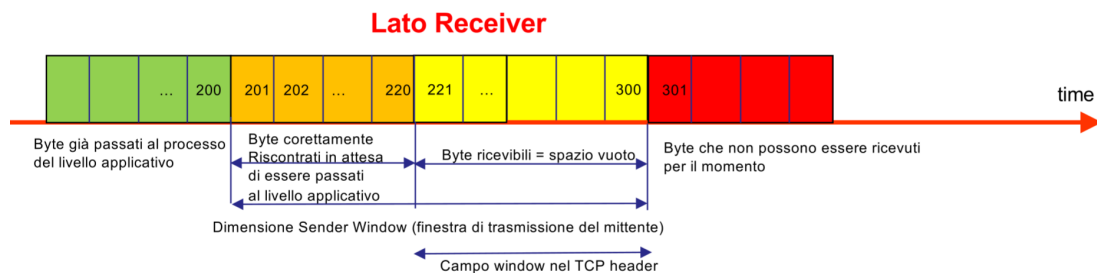


Figure 1.13: Esempio di finestra di ricezione

1.3.7 Controllo di flusso

Come gestisce il TCP il flusso dei dati?

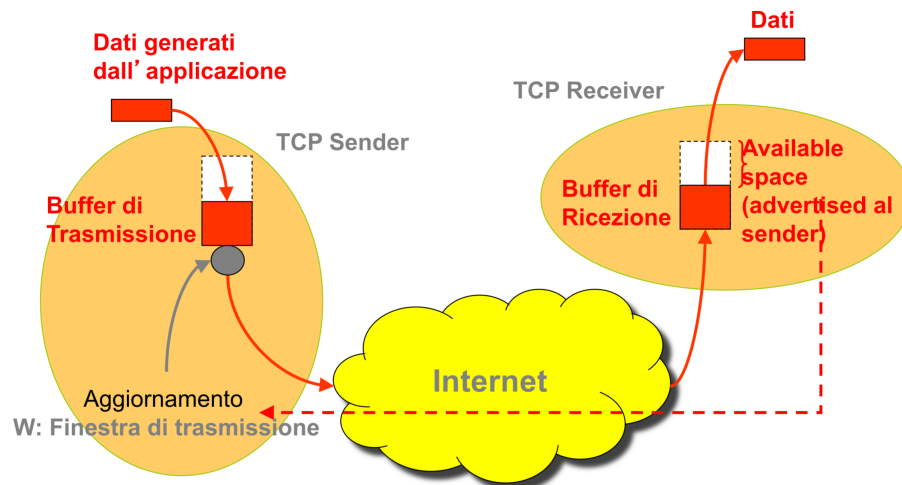


Figure 1.14: Esempio di flusso TCP

IMPORTANTE: ciò che è presente nel buffer di ricezione è stato riscontrato dal mittente.

Il receiver comunica con il sender la dimensione del buffer di ricezione, in modo che il mittente possa regolare la velocità di invio dei segmenti, quindi regolare la finestra di trasmissione.

1.3.8 Perdita di segmenti

Si assume che un segmento sia stato perso se si verificano: 3 dupack, RTO.

Retransmission Time Out - RTO

Per ogni segmento inviato, viene avviato un RTO, ossia un timer che scade dopo un certo intervallo di tempo. Se il timer scade prima di ricevere l'ACK, il mittente ritrasmette il segmento.

3 Dupack

Finestra: $W=3$
Rate medio = $3/RTT$

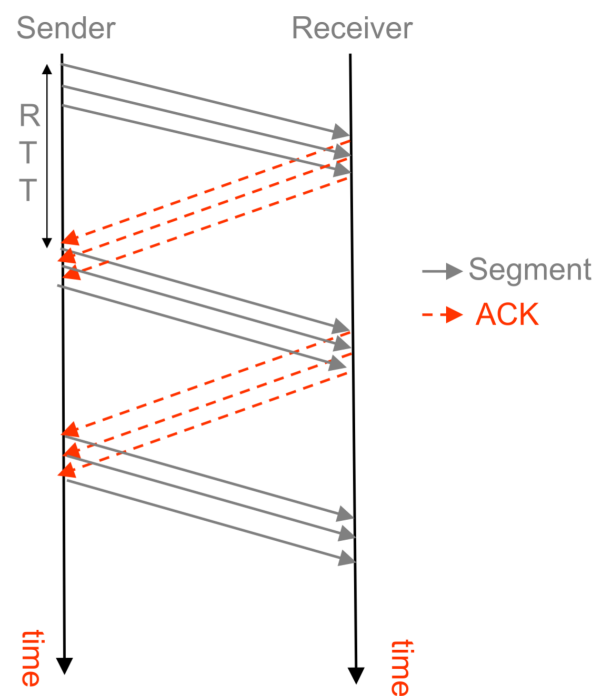


Figure 1.15: Esempio di sliding window e gestione del Retransmission Time Out