Azzolini Riccardo 2020-11-12

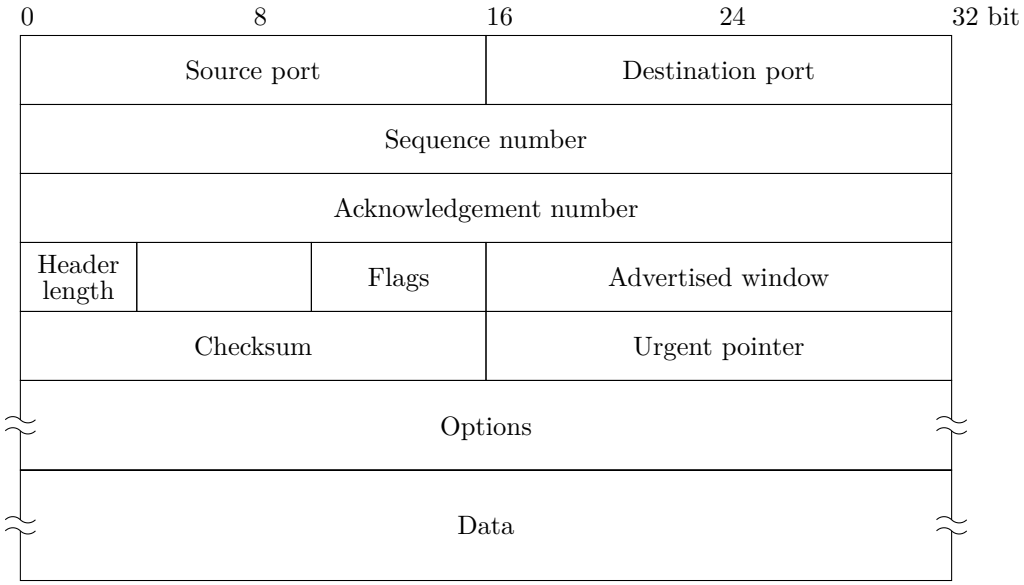
Transmission Control Protocol

1 Transmission Control Protocol

Transmission Control Protocol, **TCP**, è un protocollo di trasporto che offre un servizio connection-oriented affidabile, con comunicazione full-duplex, ordine di consegna garantito, conferma dell'avvenuta ricezione, controllo di flusso e controllo di congestione.

2 Struttura del segmento

Il pacchetto TCP, chiamato anche **segmento**, è composto dai seguenti campi:

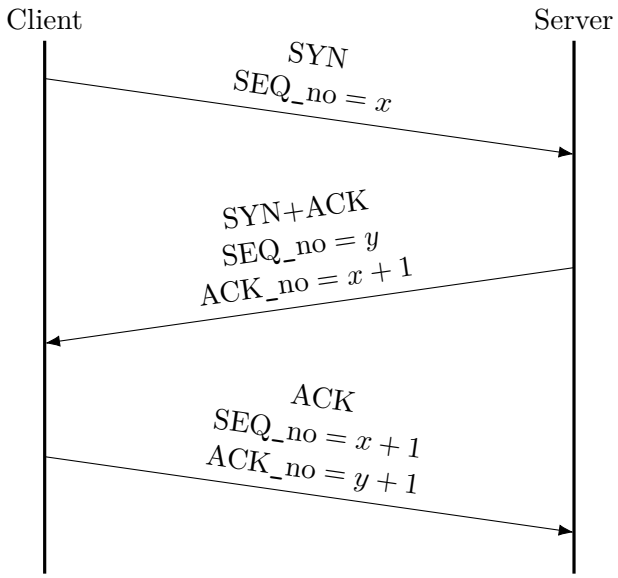


- **Source** e **destination port**: i numeri di porta associati, rispettivamente, alle applicazioni mittente e destinataria del pacchetto.
- **Sequence number**: il numero di sequenza associato al primo byte contenuto nel campo *data*. Questo campo viene usato dal destinatario per capire se ci sono segmenti persi o fuori ordine.

- **Acknowledgement number**: conferma la ricezione dei segmenti da parte del destinatario, indicando il numero di sequenza del byte successivo che il destinatario si aspetta di ricevere. Ad esempio, un acknowledgement number pari a 201 significa che il destinatario ha ricevuto tutti i byte fino al numero 200, e si aspetta quindi di ricevere altri byte a partire dal numero 201.
- **Header length**: la lunghezza dell'header, espressa in parole da 32 bit.
- Il campo **flags** contiene 6 bit, ciascuno dei quali ha un proprio significato:
  - **URG** indica se il segmento contiene dati urgenti;
  - **ACK** indica se il segmento contiene informazioni di riscontro (conferma) di ricezione;
  - **PSH** indica se il ricevitore deve passare immediatamente i dati al livello superiore (il livello applicativo);
  - **RST**, **SYN** e **FIN** sono usati per stabilire e interrompere la connessione.
- **Advertised window**: indica lo spazio disponibile nel buffer del ricevitore (misurato in byte). Quest'informazione è usata per il controllo di flusso.
- **Checksum**: un campo per il controllo degli errori.
- **Urgent pointer**: indica dove finiscono i dati urgenti, espresso come offset in byte rispetto al primo byte di dati del segmento.
- **Options**: dei campi opzionali.
- **Data**: la SDU di livello di trasporto, che corrisponde alla PDU di livello applicativo.

3 Creazione e rilascio della connessione

Siccome TCP è connection-oriented, prima che possa avvenire lo scambio di dati tra un client e un server è necessaria una fase di creazione (setup) della connessione. Questa avviene secondo un protocollo chiamato **three-way handshake**:

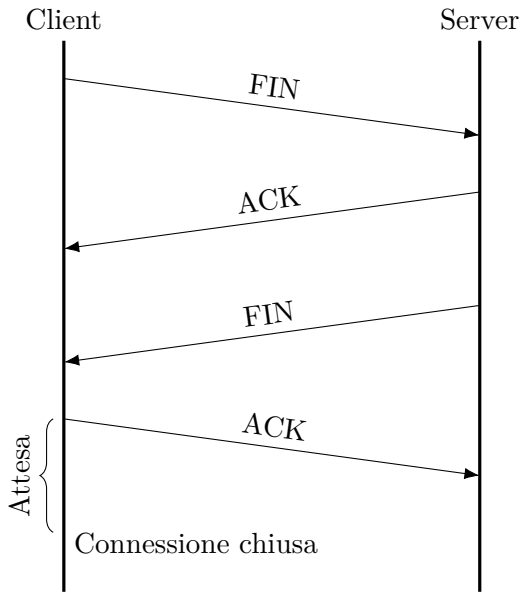


1. Il client manda al server una richiesta di connessione, inviando un pacchetto con il flag SYN (sincronizzazione) e un qualche sequence number x (ad esempio $x = 100$), dal quale il client inizierà a contare i byte inviati.

2. Il server risponde con un pacchetto che ha ancora il flag SYN (anche il server si deve sincronizzare), ma in più ha anche il flag ACK, perché conferma la ricezione del pacchetto precedente. Il sequence number è un qualche valore y (ad esempio $y = 1000$), dal quale il server inizierà a contare i byte inviati (il conto del server è indipendente da quello del client). Invece, l'acknowledgement number è $x + 1$ (ad esempio 101): esso conferma la ricezione del pacchetto con sequence number x, e indica che il server si aspetta di ricevere un pacchetto successivo con sequence number $x + 1$.

3. Infine, il client conferma la ricezione del SYN+ACK dal server, rispondendo con un pacchetto che ha solo il flag ACK. Il sequence number di questa risposta è $x+1$ (ad esempio 101), mentre l'acknowledgement number è $y + 1$ (ad esempio 1001), per confermare la ricezione del pacchetto con sequence number y.

Quando poi la comunicazione tra client e server termina, bisogna rilasciare la connessione (per liberare le risorse allocate). Ciò avviene mediante lo scambio di quattro segmenti:

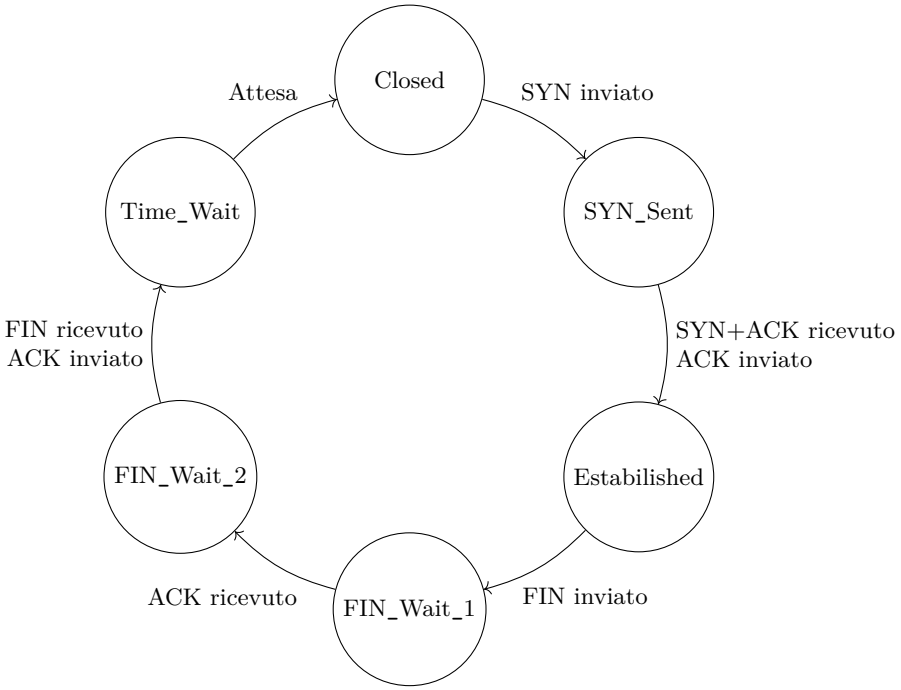


1. Il client manda al server un pacchetto con il flag FIN.
2. Il server risponde con un ACK, per confermare la ricezione del FIN.
3. Il server manda a sua volta un FIN al client.
4. Il client conferma la ricezione mediante un ultimo ACK, e infine, dopo un'attesa di durata prefissata, considera chiusa la connessione.

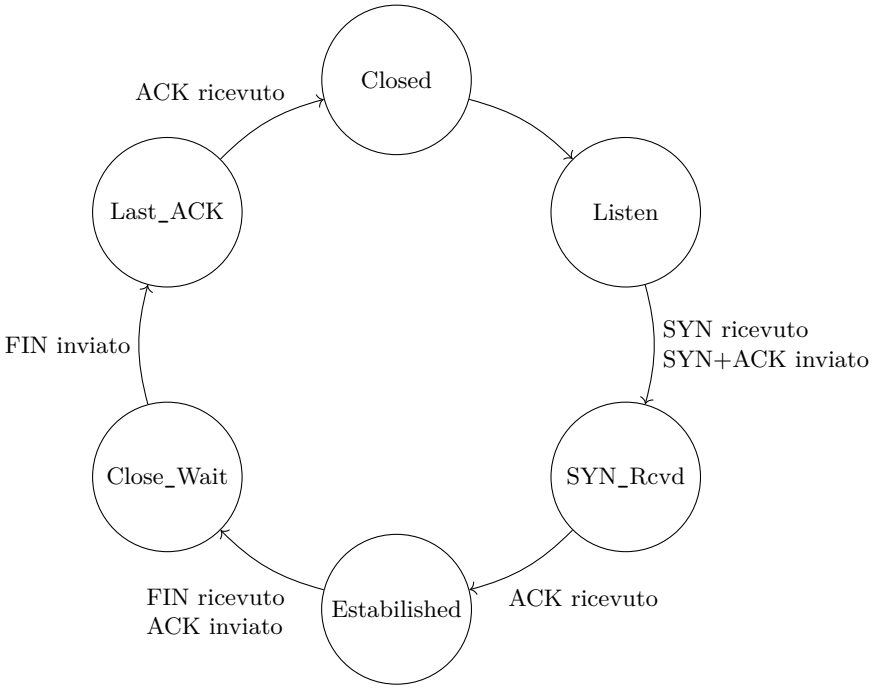
3.1 Stati di client e server

Le stesse informazioni sulle procedure di creazione e rilascio della connessione che sono mostrate nei diagrammi appena visti possono anche essere rappresentate sotto forma di automi a stati finiti.

La tipica sequenza di stati di un client TCP è:



Invece, la tipica sequenza di stati di un server TCP è:

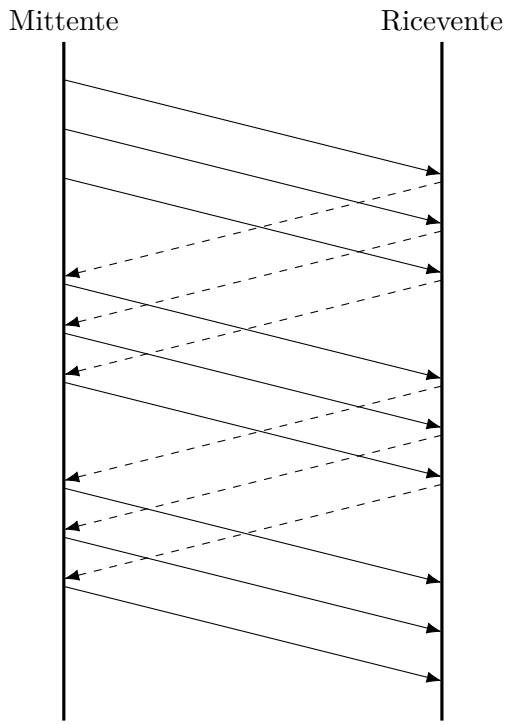


4 Sliding window

TCP garantisce l'affidabilità della comunicazione ritrasmettendo i pacchetti per cui non arriva un riscontro di ricezione (ACK). Intuitivamente, il modo più semplice di gestire la comunicazione sarebbe allora inviare un pacchetto, aspettare il riscontro di ricezione, poi inviare il successivo, e così via. Così facendo, però, si introdurrebbe un grosso delay per ogni pacchetto inviato.

Per ridurre questo delay, si usa invece un meccanismo chiamato **sliding window**: esso *stabilisce il rate di trasmissione nella comunicazione TCP, ossia il numero di pacchetti che si possono trasmettere senza attendere immediatamente il riscontro di ciascuno di essi.*

Con una sliding window di ampiezza W, il trasmittente invia W pacchetti "di fila", e solo dopo averne terminato l'invio si ferma per aspettarne i riscontri. Inoltre, ogni eventuale riscontro ricevuto già durante l'invio dei W pacchetti permette l'invio di un ulteriore pacchetto senza bisogno di aspettare. In altre parole, il parametro W è il massimo numero di pacchetti che possono essere contemporaneamente "in volo", cioè trasmessi ma non riscontrati. Ad esempio, con $W = 3$:



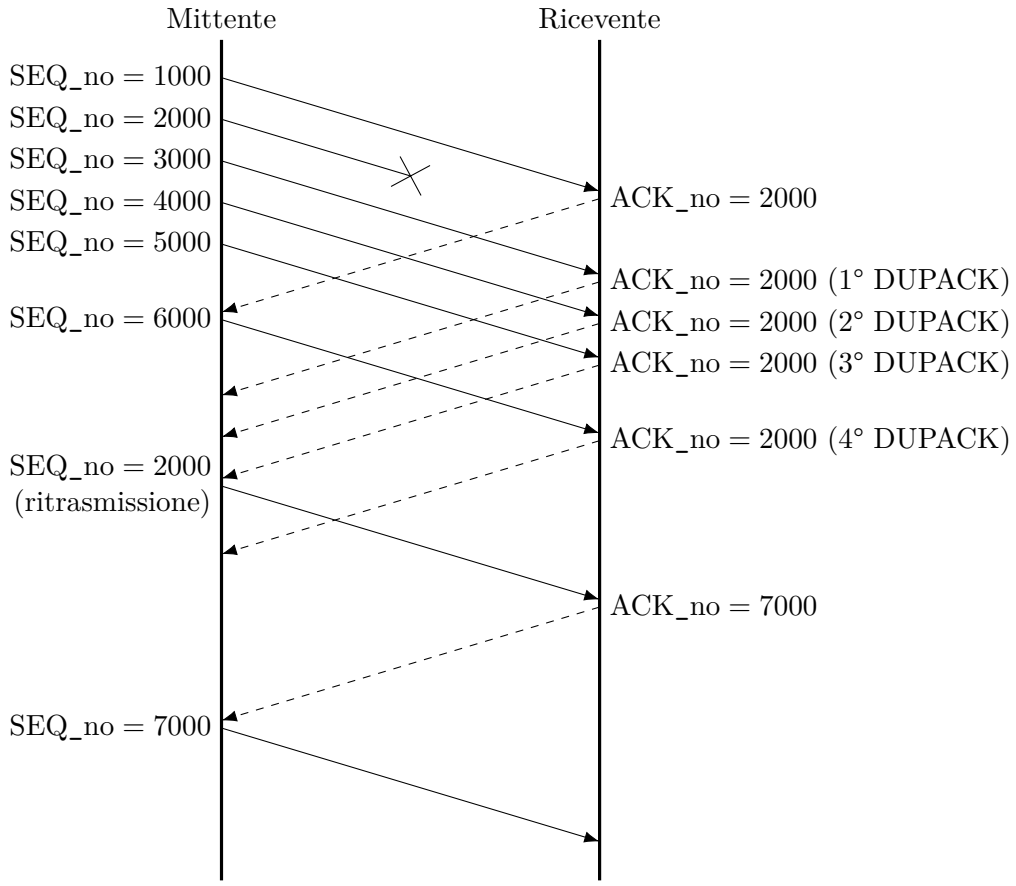
(in questa figura, le linee continue rappresentano i segmenti dati, mentre le linee tratteggiate indicano gli ACK).

5 Ritrasmissione dei segmenti

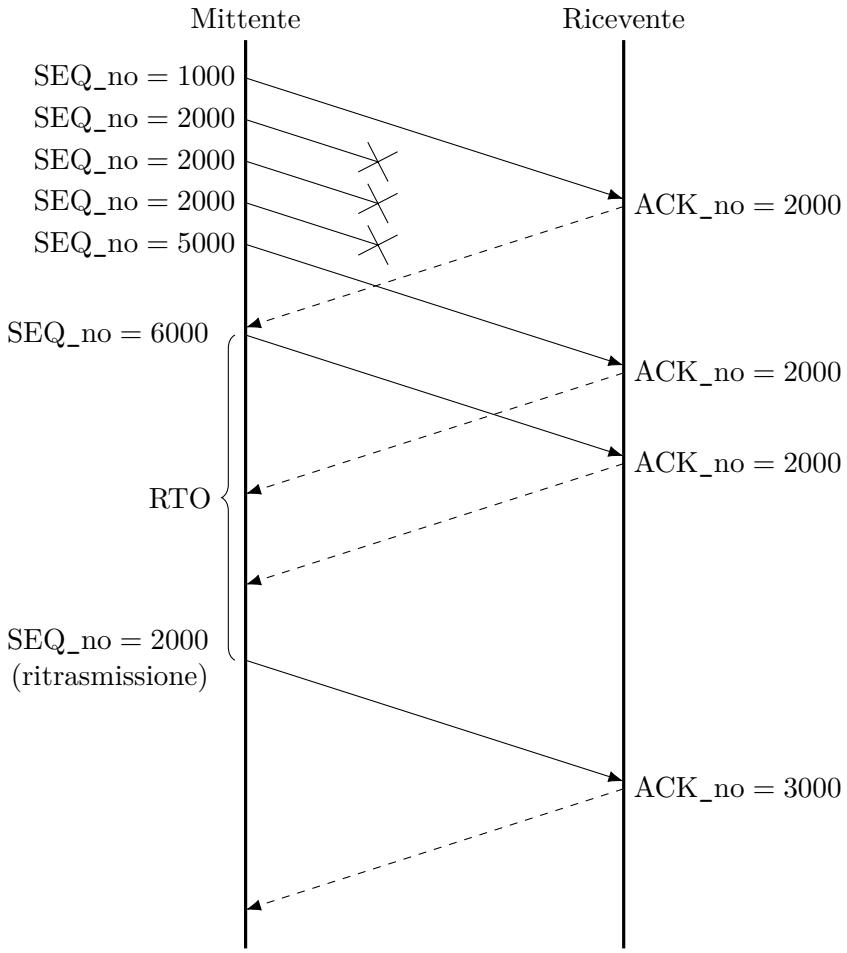
Quando un segmento con sequence number x viene perso, il mittente non riceve mai dal destinatario un corrispondente riscontro di ricezione, cioè una risposta con acknowledgement number maggiore di x (ad esempio $x+1$). Piuttosto, se giungono a destinazione pacchetti successivi a quello perso, il destinatario continua a mandare risposte di ACK con acknowledgement number x, poiché si aspetta ancora di ricevere il pacchetto con sequence number x. Questi ACK con acknowledgement number uguale a quello di ACK precedenti prendono il nome di **ACK duplicati** (DUPACK).

Il mittente suppone che un pacchetto sia andato perso, e quindi lo ritrasmette, se osserva una delle seguenti due condizioni:

- riceve 3 ACK duplicati (3 DUPACK);



- non riceve l'ACK per un pacchetto entro un tempo limite chiamato **retransmission timeout** (RTO).



Il retransmission timeout garantisce la ritrasmissione anche nei casi in cui:

- il destinatario riceve meno di tre pacchetti successivi al primo perso, quindi non invia tre ACK duplicati;
- alcuni degli ACK duplicati si perdono.

Quando finalmente il destinatario riceve un pacchetto ritrasmesso, lo mette in ordine rispetto a eventuali pacchetti successivi già ricevuti (invece di scartare questi ultimi), e richiede al mittente (tramite l'acknowledgement number) il pacchetto successivo a tutti questi.

6 Controllo di flusso

Quando il destinatario riceve un segmento TCP, prima di passare i dati al livello applicativo li inserisce temporaneamente in un buffer di ricezione. Tale buffer ha una capienza

limitata: quando è pieno, eventuali nuovi segmenti in arrivo dovrebbero per forza essere scartati, e sarebbe poi necessario che questi vengano ritrasmessi, nonostante avessero già raggiunto con successo il destinatario. Ciò comporterebbe un notevole spreco di risorse della rete.

Per evitare questa situazione, TCP implementa il **controllo di flusso**, che *regola il rate di trasmissione dei segmenti al fine di non saturare il buffer di ricezione*. Per fare ciò, il mittente utilizza come limite superiore all'ampiezza W della sliding window lo spazio libero nel buffer del destinatario, che conosce perché questo è indicato nel campo *advertised window* (*awnd*) dei segmenti di ACK. Così, quando il destinatario segnala di avere poco spazio libero nel buffer, il mittente può trasmettere solo pochi pacchetti, mentre quando il destinatario ha più spazio il mittente può aumentare il rate di trasmissione.

7 Controllo di congestione

Il rate di trasmissione dei segmenti TCP deve tenere conto non solo della capacità del destinatario, ma anche dello *stato di congestione* della rete che collega i due host comunicanti, per evitare di sovraccaricarla, provocando un *collasso di rete*. Il meccanismo che regola il rate di trasmissione dei segmenti in base alla banda disponibile, al fine di utilizzarla pienamente ma evitare collassi di rete, è chiamato **controllo di congestione**.

Il parametro che "cattura" lo stato di congestione della rete è chiamato **congestion window** (*cwnd*). Come la advertised window, esso funge da limite massimo alla dimensione W della sliding window. Allora, complessivamente, W è determinata dal *minimo* di questi due parametri:

$$W = \min(awnd, cwnd)$$

Siccome la banda disponibile non è nota a priori e varia nel tempo, il valore di *cwnd* viene calcolato e aggiornato dinamicamente per mezzo di un opportuno algoritmo di controllo di congestione.

Esistono moltissimi diversi algoritmi di controllo di congestione, ma l'idea di base è il paradigma **Additive Increase Multiplicative Decrease** (AIMD), che utilizza due fasi per la regolazione dinamica di *cwnd*:

- la fase **Additive Increase** incrementa progressivamente *cwnd*, finché non si verifica un *episodio di congestione*, che può essere la ricezione di 3 ACK duplicati o lo scadere del retransmission timeout (cioè coincide con le circostanze in cui si ritrasmette un segmento);
- la fase **Multiplicative Decrease** riduce drasticamente il *cwnd* in seguito a un episodio di congestione, per scongiurare il collasso della rete.

7.1 TCP Tahoe

Un semplice algoritmo di controllo di congestione è **TCP Tahoe**. Esso prevede l'incremento di *cwnd* a ogni ACK ricevuto, secondo due fasi chiamate **slow start** e **congestion avoidance**, separate da una soglia *ssthresh*:

$$cwnd = \begin{cases} cwnd + 1 & \text{se } cwnd < ssthresh \text{ (slow start)} \\ cwnd + \dfrac{1}{cwnd} & \text{se } cwnd \geq ssthresh \text{ (congestion avoidance)} \end{cases}$$

Quando invece si verifica un episodio di congestione, sia esso la ricezione di tre ACK duplicati o lo scadere dell'RTO, la soglia di slow start viene reimpostata a

$$ssthresh = \frac{\text{cwnd}}{2}$$

e poi la finestra di congestione viene riportata al minimo:

$$cwnd = 1$$

7.2 TCP Reno

Un altro algoritmo di controllo di congestione è **TCP Reno**. Esso funziona in gran parte come TCP Tahoe, ma tratta in modo diverso i due tipi di episodio di congestione:

- In caso di ricezione di 3 ACK duplicati, si esegue una **fast recovery**:

  $$ssthresh = \frac{\text{cwnd}}{2}$$
  $$cwnd = ssthresh$$

  In pratica, dopo aver ricalcolato *ssthresh*, la finestra di congestione viene impostata direttamente a tale soglia, saltando così la fase di slow start e passando subito alla congestion avoidance.

- Lo scadere dell'RTO viene gestito esattamente come nel TCP Tahoe:

  $$ssthresh = \frac{\text{cwnd}}{2}$$
  $$cwnd = 1$$