Vediamo altri dettagli del protocollo TCP. Il primo dettaglio che discutiamo è il timeout per il ri-trasporto dei dati persi. Tale timeout è chiamato RTO (retransmission time out).

Il funzionamento di questo timeout è che quando si stabilisce una connessione di tipo Stream col protocollo TCP c’è un tempo massimo d’attesa per ricevere l’acknowledgement (dopo aver inviato un messaggio). Se il timeout scade prima che sia arrivato un acknowledgement si assume che i dati inviati siano stati persi e dunque i dati vengono rinviati.

È abbastanza critico scegliere un valore sensato per il timeout: Se per il timeout scegliamo un valore troppo basso non diamo tempo all’acknowledgement di arrivare e rinviamo dati che in realtà sono arrivati (intasando il buffer di ricezione con copie degli stessi dati). Se scegliamo invece un valore troppo alto dal punto di vista della connessione non è problematico, ma può essere fastidioso per il mittente che deve aspettare lunghe quantità di tempo prima di poter procedere ad inviare altri dati.  
Per scegliere il valore da attribuire al timeout si può assumere che la connessione funzioni e misurare il roundtrip time (RTT), cioè la differenza di tempo tra l’invio di un messaggio e la ricezione dell’acknowledgement. A quel punto, una volta ricevuto il primo acknowledgement (in genere durante il three way handshake), basta impostare il RTO (retransmission time out) a poco più dell’RTT (potrebbero esserci dei leggeri rallentamenti nella rete).

L’RFC che spiega l’impostazione dell’RTO è il 6298 e spiega di fare in questo modo. Tutte le volte che una trasmissione va a buon fine, viene misurato il tempo passato tra l’istante di invio del messaggio e l’istante di ricezione dell’acknowledgement. Dopodiché, si raccolgono tutti i campioni relativi a tutti i messaggi inviati e si fa una statistica utilizzando due formule: una per il round trip time stimato e un’altra per la differenza tra il valore stimato e il valore effettivo relativo all’ultima trasmissione.

SRTT = (1 – alpha) \* SRTT + alpha \* R, dove alpha è un coefficiente minore di 1 (e R è l’ultimo round trip time registrato) -> l’idea è di fare una media pesata tra l’ultimo valore e il valor medio precedente (il calcolo è detto “a media mobile”).  
Il valore suggerito per il parametro alpha, secondo l’RFC è 1/8 (0.125).

A cosa serve una media mobile (e perché non usiamo una media normale)? Serve perché la rete può non essere stabile nella velocità di trasmissione delle informazioni (potremmo partire coi dispositivi scarichi e mentre si caricano la loro velocità aumenta, oppure potremmo avere un server vuoto che si riempie man mano, rallentando la velocità di risposta ai client). Con questa media pesata, che si basa sempre sui nuovi valori di RTT, non ci vuole molto perché l’RTO di un dispositivo si adatti alla velocità della connessione.

Oltre alla stima del valor medio abbiamo anche la stima delle variabilità:

RTT VAR = (1 - beta) \* RTT VAR + beta \* |SRTT - R|

In questo caso, il coefficiente suggerito dall’RFC 6298 è ¼ (0.25). Dunque, lo smorzamento sulla variabilità è più rapido dello smorzamento sul valor medio.

I valori di inizializzazione di RTO, SRTT e di RTT VAR sono (secondo l’RFC):

per RTO, il valore costante (più volte cambiato nel tempo per tenere conto della velocità delle reti) di 1 secondo. -> questo è il valore del timeout per la richiesta Syn (la prima misura di RTT la si otterrà con la risposta syn / ack).

Per SRTT, il valore R della prima misura quando si ottiene la risposta syn / ack   
(SRTT = R).

Per RTT VAR, il valore R della prima misura quando si ottiene la risposta syn / ack, diviso per due   
(RTT VAR = R / 2).

Ad ogni nuova misura, si ricalcola l’RTT VAR prima dell’SRTT (in modo da poter avere il valore vecchi di SRTT per il calcolo di RTT VAR).  
dopo l’inizializzazione, il Timeout viene definito come SRTT + max(G, K \* RTT VAR), dove secondo l’RFC K = 4 e G è la granularità del clock con cui misuriamo il roundtrip time. La Granularità è il valore minimo che il clock può misurare (può essere per esempio 1ms): vedremo in dettaglio questa cosa quando parleremo del protocollo di sincronizzazione degli orologi, NTP.

Se si arriva alla ritrasmissione, non si usa più un eventuale acknowledgement ricevuto per ricalcolare l’SRTT (e dunque l’RTO) (perché non sappiamo se l’acknowledgement è relativo alla trasmissione o alla ritrasmissione); in tal caso, il Timeout (RTO) si “limita” a raddoppiare (L’UNICA eccezione a questo è quando si deve ancora stabilire la connessione, in tal caso se l’RTO è ancora a 1s e viene perso il syn/ack, questo viene portato a 3s).

Nell’implementazione può essere previsto un valore massimo di RTO, oltre il quale il timeout non può essere più raddoppiato, e se c’è, non può essere minore di 60 secondi.

C’è un altro timeout che se la connessione è libera per troppo tempo scatta e fa interrompere la connessione. La pratica di inviare dati solo per resettare questo timer anche se non sono necessari è chiamata “Keep Alive” (e serve a tener viva la connessione; nota: potrei essermi confuso e Keep Alive POTREBBE essere invece il nome del timer, tbh non ricordo).

Un altro aspetto abbastanza interessante e complicato del protocollo TCP è il protocollo di controllo di congestione. Immaginiamo di avere i nostri soliti due host Client e Server, connessi tramite la rete internet (costituita da una serie di router). La possibilità di congestione deriva dalla condivisione di alcuni router all’interno della rete tra tanti utenti diversi (che si vogliono scambiare messaggi). Quindi la strada che deve percorrere il datagramma del nostro mittente potrebbe essere in parte sovrapposta alla strada che deve percorrere il datagramma di qualcun altro (o più altri).  
Può quindi succedere che dove si concentrano i flussi di traffico dei dati provenienti da tanti utenti diversi non ci sia abbastanza spazio in memoria: la memoria dei router è infatti composta da buffer ma di dimensione limitata, e che quindi possono essere saturati.  
Nel caso in cui i buffer dei router siano saturi, i messaggi in arrivo vengono scartati. Il protocollo TCP implementa già un controllo di flusso che è basato sullo scambio della recieve window (in modo che il mittente non mandi messaggi più grandi dello spazio nel buffer di ricezione del destinatario): sarebbe bello avere ciò anche per i router intermedi, ma ciò richiederebbe di implementare il protocollo di congestione al livello 3 (IP), dove i router sono visibili. Poiché il protocollo IP è stato implementato senza di esso, il protocollo di congestione è stato implementato al livello del TCP: è una pezza però, perché può solo stimare la dimensione dei buffer dei router intermedi e la porzione che hanno libera (tramite la cooperazione tra host mittente e host destinatario).

Mentre nel caso del controllo di flusso abbiamo il campo esplicito, che presenta un valore esatto, nell’header TCP (la recieve window), nel caso del controllo di congestione il router incriminato non può essere visibile al livello 4, quindi viene fatta una stima che viene inserita nella “Congestion window”. Per effettuare la stima vengono usati due o tre algoritmi diversi da parte del mittente.  
Il primo algoritmo utilizzato viene chiamato “slow start”; come suggerisce il nome, se si ha una certa quantità di dati da inviare si prende solo una piccola parte (il valore esatto viene scritto nella congestion window), si inviano quelli e si aspetta l’acknowledgement. Se tutto va tranquillamente (e quindi il messaggio non va perso), si raddoppia la quantità di informazioni che vengono inviate col secondo messaggio. Se va tutto bene anche col secondo messaggio si raddoppia ulteriormente il valore della congestion window e si invia la quantità di dati corrispondente. Il processo di raddoppiamento continua finché il valore nella congestion window non supera quello nella recieve window o finchè non arriva un acknowledgement (o ne arriva uno parziale); in questo caso vorrebbe dire che qualche datagramma è andato perso e si può ipotizzare di aver saturato il buffer di uno dei router che il messaggio deve attraversare.  
Se mi trovo nel secondo caso l’host smette di usare l’algoritmo slow start e passa all’algoritmo “fast recovery”. Si parte memorizzando qual è stato l’ultimo valore della congestion window per il quale non si ha avuto perdite (la potenza di 2 precedente a quella attuale) e si reinserisce nella congestion window il minimo valore “possibile” (quello con cui lo slow start era partito). Dopodiché, si ripete la stessa cosa che fa l’algoritmo slow start (raddoppiando il valore della congestion window ad ogni acknowledgement), ma se si ritorna al valore che è stato memorizzato si procede con una crescita lineare (aumentando di 1 unità per volta, che potrebbe essere 1 kb?). Quando si viene bruciati di nuovo, si ritorna da capo (ripartendo dalla singola unità e procedendo con l’algoritmo fast recovery).

Un’altra possibilità che viene interpretata come congestione è l’eventualità di acknowledgement ripetuti. Supponiamo di avere inviato tanti datagrammi, ma uno di essi (in mezzo agli altri) viene perso. Quando il ricevente si rende conto di avere un buco nei datagrammi ricevuti, perché ha inviato gli acknowledgement degli altri fino a quello, questo manda un acknowledgement ripetuto (in particolare, ripete l’acnowledgement di quello precedente). Anche questo caso viene interpretato come congestione e porta al “reset” degli algoritmi.

Il protocollo TCP implementa quindi un protocollo di controllo di flusso “fatto bene” e un protocollo di controllo di congestione che però non può essere perfetto, perché gli algoritmi usati si possono basare solo sui fallimenti già avuti in precedenza nell’invio di informazioni e può al massimo effettuare una stima (non essendo le dimensioni dei buffer dei router visibili al livello 4).   
Questi due protocolli di controllo si combinano nello stabilire il limite massimo delle informazioni inviabili nella rete in un singolo datagramma.

C’è da dire che la slow start limita la velocità della banda di comunicazione se si vogliono inviare piccole quantità di dati: infatti i pochi byte inviati all’inizio vengono compensati dalle grosse quantità di byte che vengono inviate dopo pochi secondi, ma ciò non ha senso se i dati da inviare non sono tanti in primo luogo. Anche per questo motivo la connessione TCP risulta conveniente per l’invio di grandi quantità di dati, mentre il protocollo UDP resta più conveniente per messaggi di dimensioni relativamente piccole.