Protocollo TCP - 2

mercoledì 16 agosto 2023 18:05

TCP, come RDT, utilizza un meccanismo di timeout per cui vengono recuperati i segmenti persi...

Come stabiliamo la durata di un timeout?

Questo dovrebbe essere più grande del **SampleRTT** di un dato segmento, è la quantità di tempo che intercorre tra l'istante di invio del segmento a quello di ricezione del suo ack.

Viene misurato il SampleRTT di OGNI singolo segmento (esclusi quelli ritrasmessi); il valore risultate cambia oltre che da segmento a segmento anche in base alla configurazione dei router e dal diverso carico sui sistemi periferici.

Di conseguenza non possiamo affidarci ai singoli valori, bensì effettuiamo una stima, EstimatedRTT, calcolando la media di tutti SampleRTT presenti, questa stima viene aggiornata quando si ottiene un nuovo valore SampleRTT nel seguente modo:

$$EstimatedRTT = (1-\alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

Combinazione ponderata del suo vecchio valore e del nuovo valore di SampleRTT

peso che gli assegno nel calcolo dei valori

Bisogna definire il timeout a partire da questa media, aggiungendogli una sorta di margine di sicurezza che dipende dalla variabilità dei ritardi.

Se la rete è stabile, ovvero le code nei router hanno una stabilità e ci sono ritardi stabili, vorrei che il mio timeout fosse poco più grande della mia stima, più i valori si distano di poco meglio è.

Quando aumenta il traffico, crescono le dimensioni delle code, la mia media stimata e la variabilità dei ritardi.

Si definisce la **variazione** di quanto il nuovo valore SampleRTT si discosta dalla stima EstimatedRTT:

discostamento tra i Sample precedenti e i nuovi

$$DevRTT = (1-\beta) \times DevRTT + \beta \cdot |SampleRTT-EstimatedRTT|$$

Il valore suggerito per β è 0,25

Più il valore è piccolo più le fluttuazioni sono limitate, viceversa se il valore è grande ci sono notevoli fluttuazioni tra i valori SampleRTT.

Ricapitolando l'intervallo di timeout non deve quindi essere inferiore a quello di Estimated RTT ma neanche troppo maggiori, altrimenti comporterebbe gravi ritardi sul trasferimento dei dati...Soluzione: impostiamo il timeout a EstimatedRTT + un margine grande quanto è la fluttuazione nei valori di SampleRTT:

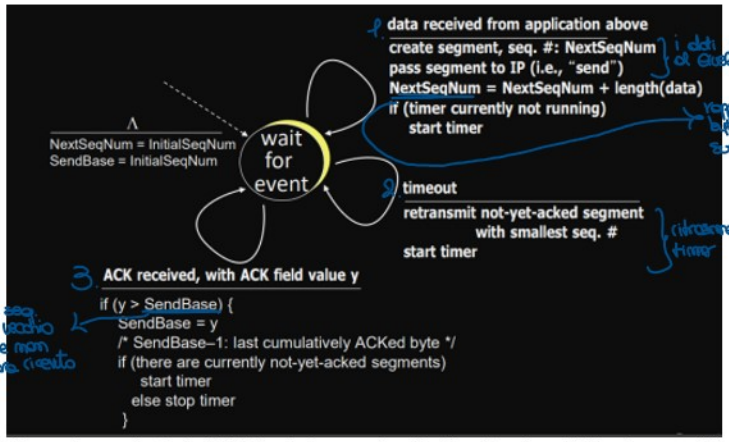
$$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT.$$

Inoltre quando si verifica un timeout, *Timeout Interval* viene raddoppiato per evitare un timeout prematuro riferito a un segmento successivo per cui si riceverà presto un ACK.

Appena viene ricevuto un segmento *EstimatedRTT* viene aggiornato, di conseguenza anche *TimeoutInterval* viene ricalcolato secondo la formula precedente.

Le procedure per la gestione dei timer TCP utilizzano un solo timer di ritrasmissione, anche in presenza di più segmenti trasmessi, In pratica il timer è associato al più vecchio segmento che non ha ancora ricevuto ACK...

Dato mittente



i dati vengono impacchettati e passati al livello di rete

rappresenta il numero del primo byte di dati del segmento nel suo flusso

ritrasmetto e avvio timer

num di seq del più vecchio byte che non ha ancora ricevuto un ack

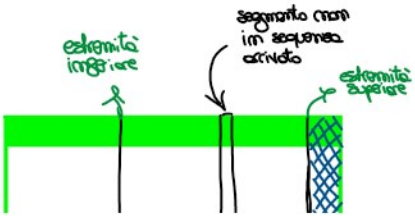
Ci sono 3 eventi principali: (1)Ev. dati provenienti dall'applicazione, (2)Ev. timeout, (3) Ev. ricezione del segmento di acknowlegment con un valore valido nel campo.

TCP ricevente: generazione di ACK

In risposta dei seguenti eventi, il ricevente TCP effettua azioni diverse:

Evento	Azione
(1) Arrivo di un segmento in ordine con un sequence number x previsto *(per tutti i dati fino a sequence number x sono già stati inviati i rispettivi ACK)*.	(1) Viene mandato un ACK ritardato, bisogna attendere fino a 500ms per il segmento successivo; se non ce n'è si invia l'ACK.
(2) Arrivo di un segmento in ordine con sequence numberx # previsto. Un altro segmento ha un ACK in attesa.	(2) Viene inviato immediatamente un ACK , corrispondente all'ultimo segmento arrivato, in risposta.
(3) Arrivo di un segmento fuori ordine, con sequence number più alto di quello previsto: GAP(buco nella finestra) rilevato.	(3) Viene inviato immediatamente un ACK duplicato indicando sequence number x del prossimo byte atteso.
(4) Arrivo di un segmento che riempie parzialmente o completamente il GAP.	(4) Viene inviato immediatamente l'ACK corrispondente, a patto che il segmento si trovi

Nota: TCP cerca di ottimizzare: quando arriva un segmento, non manda subito il riscontro! Il ragionamento fatto da TCP è: se arriva un segmento, è probabile che faccia parte di una sequenza, quindi attendo un certo tempo (tipicamente 500 millisecondi), se arriva il secondo segmento, dato che i ***riscontri sono cumulativi,*** *riscontra solo il secondo, automaticamente anche il primo*

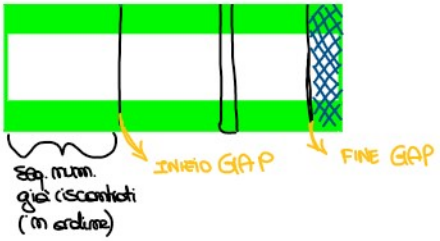


finestra) rilevato.

(4) Arrivo di un segmento che riempie parzialmente o completamente il GAP.

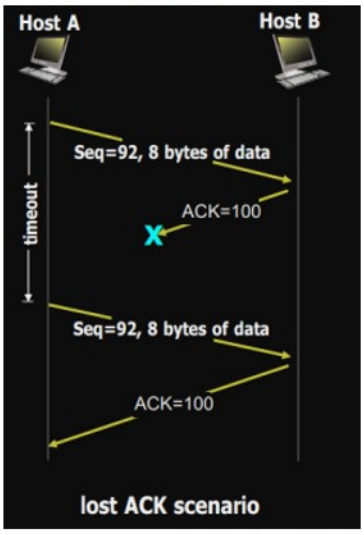
atteso.

(4) Viene inviato immediatamente l'ACK corrispondente, a patto che il segmento si trovi all'estremità inferiore del GAP.



Ora analizziamo nel dettaglio gli scenari possibili....

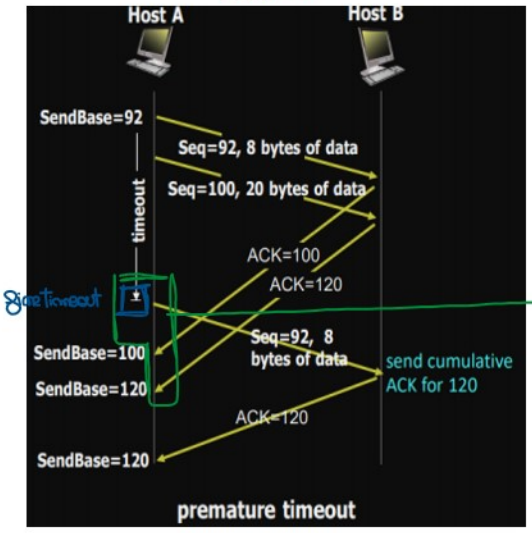
Scenario 1



ACK 100 → seq number + lunghezza dati

L'Host B riceve la ritrasmissione, rileva dal numero di sequenza che il segmento contiene dati che sono già stati ricevuti, quindi scarta i dati del segmento ritrasmesso e invia l'ACK di dove è arrivato.

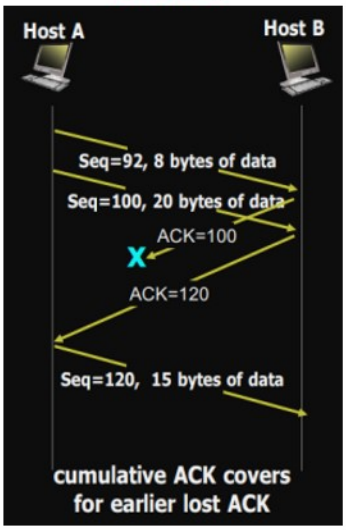
Scenario 2



gli ACK arrivano dall'host A dopo il timeout, è come se non fossero mai arrivati

Fino a quando l'ACK del secondo segmento non arriva prima del nuovo timeout, il secondo segmento non verrà ritrasmesso.

Scenario 3



Appena prima dell'evento di timeout, l'Host A riceve un acknowledgment con numero 120; quindi l'host A è a conoscenza che l'host B ha ricevuto tutto fino al byte 119, di conseguenza l'host A non rispedisce nessuno dei due segmenti, *anche se non ha ricevuto l'ACK dello scorso segmento...*

Introduciamo ora un problema: se la rete diventa rapidamente congestionata e non aggiorno abbastanza velocemente il timeout, questo scatterà in continuazione provocando una ritrasmissione eccessiva di pacchetti non necessaria...

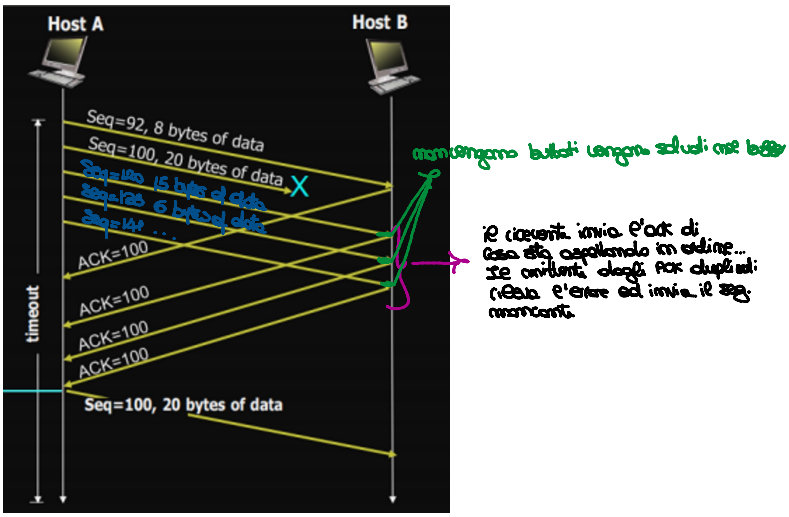
Varianti utilizzate nella maggior parte delle implementazioni TCP: Raddoppio dell'intervallo di timeout -->

Algoritmo di Karn

Di base la lunghezza dell'intervallo di timeout, dopo la scadenza di un timer, viene derivata dagli ultimi EstimatedRTT e DevRTT; con questa variante abbiamo che in tutti i casi in cui si verifica un timeout, TCP ritrasmette il segmento con il più basso numero di sequenza che non ha ancora ricevuto acknowledgment.
Ogni volta che questo si verifica, viene impostato il successivo intervallo di timeout al doppio del valore precedente.
Gli intervalli quindi cresceranno esponenzialmente ad ogni ritrasmissione! Viene stabilito per questo una soglia tale per cui una volta raggiunta, TCP dichiara morta la connessione e dealloca tutte le risorse.

Varianti utilizzate nella maggior parte delle implementazioni TCP: Ritrasmissione rapida

Quando si smarrisce un segmento, il lungo periodo di timeout impone al mittente di ritardare il nuovo invio del pacchetto perso, aumentando il ritardo end-to-end, tuttavia il mittente può rilevare la perdita del segmento attraverso gli ACK duplicati relativi a un segmento il cui ACK è già stato ricevuto dal mittente, di conseguenza un secondo ACK duplicato inviato dal ricevente mi segnala un errore, ben prima che si verifichi l'evento di timeout.



TCP è un ibrido tra GBN e SR.

Controllo di flusso

Se l'applicazione (consumatore) è lenta nella lettura dei byte all'interno del buffer di ricezione, accade che il mittente (produttore) potrebbe mandare in overflow il buffer di ricezione del consumatore inviando troppi dati rapidamente.
TCP offre un servizio di controllo di flusso (flow-control-service) alle proprie applicazioni per evitare che il mittente saturi il buffer del ricevente; vi è pertanto un confronto tra la velocità di invio e quella di lettura.
Viene mantenuto lato mittente una variabile chiamata **finestra di ricezione** *(receive window)*, che fornisce, al mittente, un'indicazione dello spazio libero disponibile nel buffer del destinatario.
TCP è full-duplex, di conseguenza i due sistemi periferici che comunicano mantengono finestre di ricezione distinte.
Esempio sul controllo di flusso: l'host A deve inviare un file di grandi dimensioni all'host B…
B *alloca il suo buffer* di ricezione per la connessione, la cui dimensione è **RcvBuffer**; B non legge immediatamente i dati immessi nel buffer. Definiamo due variabili che serviranno per il controllo di flusso:

- **LastByteRead**: numero dell'ultimo byte letto (dal buffer) dal processo applicativo B;
- **LastByteRcvd:** numero dell'ultimo byte entrato nel buffer del processo applicativo B.

Facendo la differenza di questi si ottiene il numero di byte presenti nel buffer non ancora letti da B, per non mandare il buffer in overflow deve quindi valere:

$$LastByteRcvd - LastByteRead \leq RcvBuffer$$

Più la distanza tra l'ultimo byte ricevuto e l'ultimo byte letto aumenta, più il risultato della loro sottrazione aumenta…
La finestra di ricezione, indicata con rwnd, viene imposta sulla base della quantità disponibile di spazio del buffer:

$$rwnd = RcvBuffer - [LastByteRcv - LastByteRead]$$ *rwnd è dinamica, varia nel tempo.*

Quindi, l'host B comunica all'host A quanto spazio ha disponibile sul suo buffer della connessione *(il valore viene comunicato all'interno dell'intestazione del segmento nel campo "receive windows").*
L'host A, dalla sua parte, tiene traccia di due variabili specifiche per la connessione fra i due:

- LastByteSent, ultimo byte mandato;
- LastByteAcked, ultimo byte per cui si è ricevuto un ACK.

Facendo la differenza fra questi otteniamo la quantità di dati spediti da A per cui ancora non è stato ricevuto l'ACK, questo valore deve essere tale per tutta la durata della connessione:

$$LastByteSent - LastByteAcked \leq rwnd$$

La dimensione della finestra modula la velocità di trasmissione.
Problema: supponiamo che l'host B abbia riempito il suo buffer ed rwnd = 0; B svuota il buffer, solo che non possiamo comunicarlo ad A , che rimane quindi bloccato al vecchio valore in cui rwnd = 0 , finestra di ricezione piena, e non può mandare più dati --> *TCP permette l'inoltro di un segmento solo se contiene dati o si tratta di un segmento ACK..*
Soluzione: l'host A continua comunque ad inoltrare segmenti di prova, ovvero senza alcun payload, prima o poi il destinatario invierà l'ACK corrispondente a questo segmento di prova dove comunica che il buffer è nuovamente usufruibile.