Gestione delle connessioni TCP

A.A. 2005/2006

Walter Cerroni

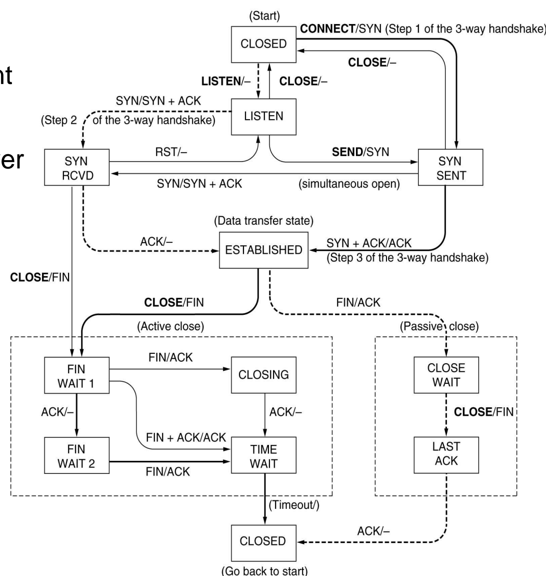
Le primitive Berkeley Socket

- Interfaccia TCP/applicazione tipica dei sistemi operativi
  - primitive eseguite dal processo server
    - **socket**: crea una nuova entità end-point (simile all'apertura di un file)
    - **bind**: assegna l'indirizzo IP di una interfaccia alla socket creata
    - **listen**: si mette in ascolto sulla socket creata, in attesa di connessioni da parte dei client (anche più di uno)
    - **accept**: accetta una richiesta di connessione creando un processo separato per gestirla (fork) e torna in ascolto sulla socket
    - **send/receive**: trasmette/riceve dati sulla connessione
    - **close**: chiude la connessione
  - primitive eseguite dal processo client
    - **socket**: crea una nuova entità end-point (simile all'apertura di un file)
    - **connect**: blocca il processo client e tenta di aprire una connessione verso il server; sblocca il client a connessione instaurata
    - **send/receive**: trasmette/riceve dati sulla connessione
    - **close**: chiude la connessione

La macchina a stati finiti del TCP

- Linee marcate
  - azioni tipiche di un client
- Linee tratteggiate
  - azioni tipiche di un server
- Linee leggere
  - eventi inusuali
- Transizioni
  - causa/effetto

Da A.S. Tanenbaum, “Reti di Calcolatori”



3

I time-out del TCP

- Al termine della chiusura attiva, il client
  - si ferma nello stato TIME-WAIT per un tempo pari a 2 MSL
  - dopodiché chiude definitivamente la connessione passando allo stato CLOSED
- Il TIME-WAIT vuole garantire l’estinzione di segmenti appartenenti a precedenti incarnazioni
  - dopo la chiusura una connessione potrebbe essere ricreata
  - la nuova incarnazione potrebbe finire per usare numeri già utilizzati dall’incarnazione precedente
- Oltre al Time-Out dello stato TIME-WAIT, il TCP mantiene attivi altri 3 contatori per ogni connessione
  - Retransmission Time-Out
  - Keepalive Timer
  - Persist Timer

4

Time out di trasmissione

- Ogni volta che si trasmette un segmento viene fatto partire il **Retransmission Time-Out** (**RTO**) e si attende il riscontro, che deve arrivare prima che RTO scada
  - il timeout deve essere dimensionato in relazione al tempo di andata e ritorno (**Round-Trip Time** o **RTT**)
  - RTT è una variabile aleatoria la cui varianza dipende dalle condizioni della rete
  - è molto difficile scegliere un corretto valore per il RTO
- Il timeout deve essere determinato in modo dinamico
  - l'idea di base è di mantenere aggiornato un valore medio del RTT e di calcolare il RTO in funzione di esso e della sua varianza

5

Algoritmo di Jacobson (RFC 2988)

- Si stima RTT tramite un campionamento continuo di quello che accade ad alcuni dei segmenti inviati
  - si spedisce un segmento, si memorizza l'istante di invio e lo si sottrae all'istante di arrivo dell'ACK relativo, misurando così un campione del RTT
  - una volta aggiornato il campione di RTT, si fa partire un'altra misura
- Si utilizzano i seguenti valori:
  - $sRTT_k$ è il k-mo campione del RTT
  - $eRTT_k$ è la stima del valore medio del RTT basato sulle misure dei campioni da 0 a k

$$eRTT_k = (1 - \alpha)\, eRTT_{k-1} + \alpha\, sRTT_k$$

- Valore tipico di $\alpha = 1/8$

6

Media pesata mobile esponenziale

- La procedura utilizzata per la stima di RTT viene detta **Exponential Weighted Moving Average**

$eRTT_0 = (1-\alpha)\ sRTT_0$

$eRTT_1 = \alpha\ (1-\alpha)\ sRTT_0 + (1-\alpha)\ sRTT_1$

$eRTT_2 = \alpha\ [\alpha\ (1-\alpha)\ sRTT_0 + (1-\alpha)\ sRTT_1] + (1-\alpha)\ sRTT_2$

…

$eRTT_k = (1-\alpha) \sum_{i=0}^{k} \alpha^{k-i}\ sRTT_i$

  - **pesata** perché con il parametro α si può decidere che peso dare ai campioni più recenti
  - **mobile** perché si modifica man mano che vengono misurati nuovi campioni
  - **esponenziale** perché il peso dei valori passati diminuisce in modo esponenziale a favore dei valori più recenti ($\alpha < 1$)

7

Definizione di RTO (RFC 2988)

- Utilizza $eRTT_k$ e la deviazione di $sRTT_k$ rispetto alla stima
  - si stima una specie di deviazione standard di RTT, detta deviazione media, con la formula

$$devRTT_k = (1 - \beta)\ devRTT_{k-1} + \beta\ |eRTT_k - sRTT_k|$$

  - valore tipico di $\beta = 1/4$
  - quindi si pone

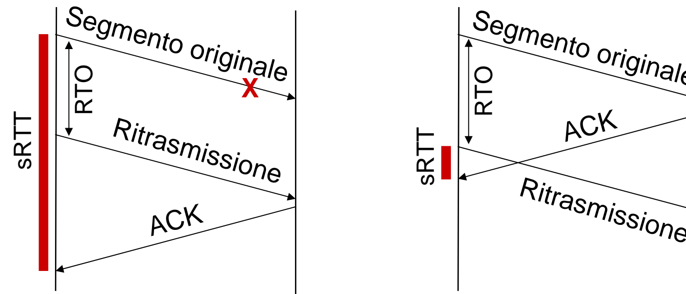
$$RTO_k = eRTT_k + 4\ devRTT_k$$

  - se le fluttuazioni sono piccole RTO $\cong$ eRTT
  - se le fluttuazioni sono grandi RTO > eRTT in modo consistente

8

Ambiguità di sRTT in caso di ritrasmissione

- In caso di ritrasmissione c'è ambiguità nella valutazione di sRTT

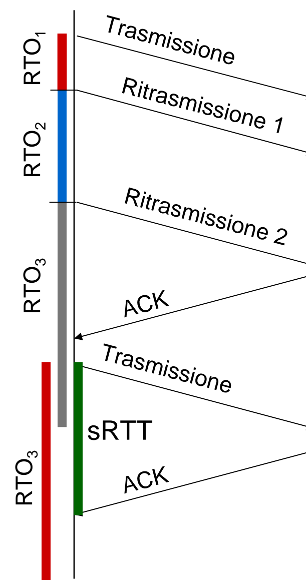


- Algoritmo **Karn-Partridge**
  - in caso di ritrasmissione non viene misurato sRTT
  - si riprende con la misura al prossimo segmento confermato senza ritrasmissione

9

Back-off esponenziale

- Dopo una ritrasmissione
  - il valore corrente di RTO potrebbe essere troppo basso e causare ritrasmissione
  - può avere senso dare un tempo maggiore al ricevitore
- Ad ogni ritrasmissione il TCP raddoppia RTO fino al raggiungimento di un valore massimo



10

Keepalive Timer e Persist Timer

- Una connessione TCP attiva non implica un continuo trasferimento di dati fra sorgente e destinazione
  - es. lettura di una pagina web reperita da un server
  - una connessione può rimanere attiva ma dormiente teoricamente all'infinito

  → molte implementazioni del TCP prevedono l'invio di informazioni di refresh, per verificare che una connessione esistente sia effettivamente attiva, allo scadere del **Keepalive Timer**

- Sono possibili situazioni di stallo (deadlock) in cui ciascuno dei due interlocutori aspetta segmenti e/o riscontri da parte dell'altro

  → scaduto il **Persist Timer**, il mittente invia un segmento-sonda chiedendo conferma della dimensione della finestra

11

Dimensione variabile della finestra

- Il TCP prevede un meccanismo a finestra scorrevole di dimensione variabile per aumentare la flessibilità
  - adattamento alla velocità di elaborazione del ricevente
  - adattamento alle condizioni di traffico nella rete
- La dimensione della finestra viene messa a punto dinamicamente basandosi su:
  - valore comunicato dal ricevente (**Advertised Window** o **AW**)
    - AW indicata in modo esplicito al trasmettitore nell'intestazione TCP
  - stato di congestione della rete (**Congestion Window** o **CW**)
    - non c'è in genere comunicazione esplicita di CW
    - il TCP gli attribuisce un valore in base alla percezione di congestione che riceve dalla rete

w = min{AW,CW}

12

Pericolo di deadlock

Trasmittente
1. Invia segmenti
4. Riceve il segmento con AW = 0
5. Sospende l'invio dei dati

Ricevente (lento)
2. Il buffer di ricezione si riempie
3. Invia un segmento con AW = 0
6. Non ha altri segmenti da trasmettere

A questo punto il protocollo è in **deadlock**
- il trasmittente non può inviare dati poiché AW = 0
- il ricevente non ha dati da inviare quindi non ha modo di comunicare AW > 0

Soluzione: TCP prevede che sia sempre possibile inviare un segmento-sonda di 1 byte anche se AW = 0

13

Persist timer e window probe

- Il trasmettitore che riceve un ACK con ackN=X e AW = 0 fa partire il **Persist Timer** (PT)
  - PT = 1.5 sec per un normale collegamento LAN
- Quando PT scade, il trasmettitore invia un segmento-sonda di 1 byte (**window probe**)
  - seqN=X
- Il ricevitore deve rispondere
  - se invia ACK con ackN = X+1 e AW > 0
    - ha ricevuto correttamente il byte del segmento-sonda
    - la trasmissione riprende normalmente
  - se invia ACK con ackN = X e AW = 0
    - non ha spazio nel buffer di ricezione perciò non può ricevere il byte X
    - PT = 2PT e si ricomincia ad attendere
- Il massimo valore di PT viene fissato a 60 sec

14

Controllo della congestione

- TCP cerca di adattare la dimensione della finestra alle condizioni di congestione della rete

$$w = \min\{AW, CW\}$$

- Idea di base: se si verifica congestione in rete si rallenta la trasmissione
  - quando si verifica una perdita si riduce w
  - quando gli ACK arrivano correttamente w viene aumentata
- w può essere espressa
  - in byte (w)
  - in numero di segmenti (W): in questo caso si deve indicare quale lunghezza si assume per i segmenti
- Normalmente W viene misurata in segmenti di dimensione massima

$$w = W \times MSS$$

15

CW ideale

- Avendo a disposizione una banda B (byte/sec) il massimo **throughput** si ottiene quando il protocollo a finestra non limita la velocità di scambio dei dati

$$w_{id} = RTT \times B$$

- In questo caso si utilizza al 100% la capacità disponibile nella tratta trasmettitore-ricevitore
  - se $w < w_{id}$: si spreca banda
  - se $w > w_{id}$: è necessario accodare nei router intermedi
    - cresce il ritardo e potenzialmente anche la perdita
- Il massimo throughput (byte/sec) vale:

$$S = w/RTT$$

16

CW all’apertura e durante la connessione

- Al momento dell’instaurazione della connessione TCP la banda disponibile B è incognita
  - a quale valore si deve impostare CW?
- La banda disponibile B può cambiare durante la connessione
  - CW va adattata dinamicamente alla banda disponibile
- Definite due fasi di controllo della congestione che corrispondono a diverse dinamiche di CW
  - **Slow start**
    - per raggiungere velocemente un W prossimo a W_{id}
  - **Congestion avoidance**
    - per far sì che W sia il più prossimo possibile a W_{id} durante la connessione

17

Slow start

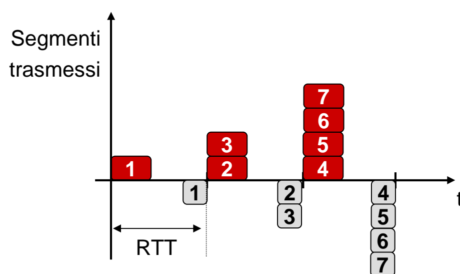
- All’inizio W = 1, cioè w = MSS
  - trasmesso il segmento, il mittente si ferma ed aspetta l’ACK
- Se ACK arriva entro RTO si pone W = 2 e si trasmettono 2 segmenti
- Ad ogni nuovo ACK ricevuto **W = W + 1**
  - ricevuto l’ACK del terzo segmento risulta W = 4
- Slow start viene interrotto quando si raggiunge la **Slow Start Threshold** (**SSThr**)
  - all’apertura della connessione un valore tipico per la soglia è SSThr = 64 Kbyte

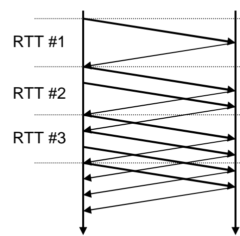
18

Evoluzione temporale

- Ipotesi: l'evoluzione di W avviene per tempi multipli di RTT
- W ha una **crescita esponenziale**
  - al termine di ogni RTT la finestra è raddoppiata rispetto al caso precedente
- La fase di Slow start dura approssimativamente

$$T_{ss} = RTT \times \log_2 SSThr$$

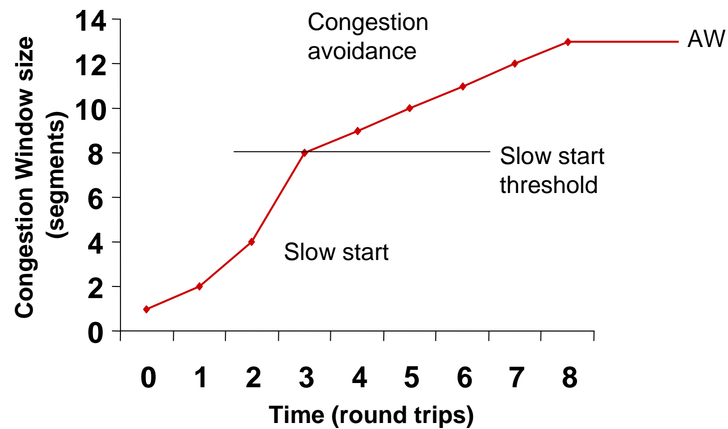




Congestion avoidance

- Raggiunta la soglia dello Slow start, inizia la fase di Congestion avoidance
- Si passa da una crescita esponenziale ad una crescita lineare
- Ad ogni nuovo ACK ricevuto **W = W + 1/W**
  - se tutti i messaggi correttamente ricevuti generano un ACK, ogni RTT risulta W = W + 1
  - se il protocollo utilizza i delayed ACK (si riceve un ACK ogni due messaggi), ogni RTT risulta W = W + ½
- W viene incrementata di uno ad ogni RTT fino a raggiungere teoricamente il valore di AW

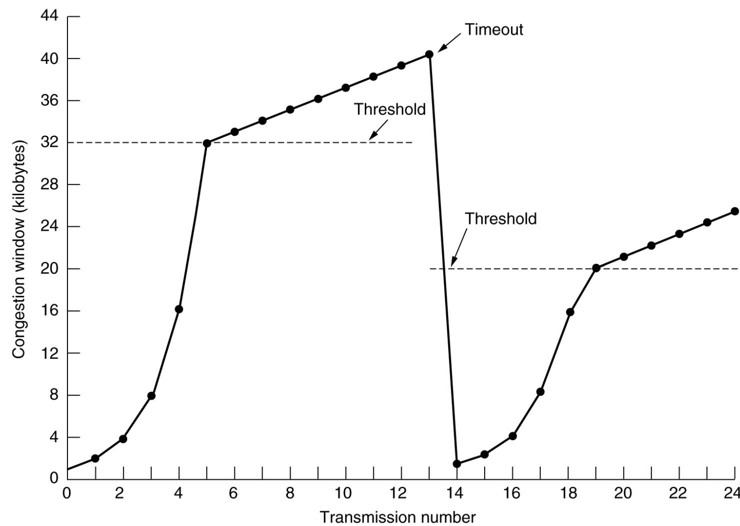
Esempio di evoluzione della finestra



In caso di congestione

- Un segmento non viene riscontrato
  - scade il RTO
- Questo evento viene preso come indicatore di rete congestionata
  - con una buona stima del RTT il time-out scaduto è (quasi) sempre dovuto a perdita del segmento
  - con una tecnologia di trasmissione affidabile la perdita è (quasi) sempre dovuta a saturazione delle code nei router
- TCP in Slow Start
  - si riparte da capo ponendo W = 1
- TCP in Congestion Avoidance
  - si impone SSThr = W/2
  - riparte lo Slow Start con W = 1

Esempio di evoluzione della CW



Da A.S. Tanenbaum, "Reti di Calcolatori"

23

Alcuni commenti

- In questo modo TCP si adatta dinamicamente alle variazioni di capacità della rete, tendendo ad occupare tutta la banda disponibile (protocollo greedy)
- Questa tecnica permette di ottenere un'equa distribuzione della banda disponibile fra tutte le connessioni presenti
- Recentemente sono state proposte nuove implementazioni di TCP, con una gestione più sofisticata dello slow start e della soglia, che risultano più efficienti
- Se la rete è molto inaffidabile (per esempio una rete wireless) non si può attribuire ogni perdita di pacchetto a congestione e occorrono algoritmi speciali

24

Segmenti fuori sequenza

- Quando un segmento viene perso il ricevitore riceve uno o più segmenti fuori sequenza
  - il ricevitore dovrebbe inviare immediatamente un ACK duplicato per ogni segmento fuori ordine
  - l'ACK duplicato è riferito all'ultimo segmento ricevuto correttamente
- La perdita di un segmento genera ACK duplicati
  - questo può essere utilizzato come indicazione di congestione unitamente al time-out
- Sono stati proposti ulteriori algoritmi per sfruttare al meglio questa situazione
  - **Fast retransmit**
  - **Fast recovery**

25

Fast retransmit

- La ricezione di 3 ACK duplicati (cioè di 4 ACK con uguale ackN) indica la perdita del segmento successivo a quello confermato dagli ACK (sono stati ricevuti i byte fino a ackN-1)
- TCP ritrasmette il segmento mancante senza attendere la scadenza del time-out
  - si ritrasmette il segmento tale che seqN=ackN
  - si fa partire la procedura di Fast recovery

26

Fast recovery

- Gli ACK duplicati indicano che il ricevitore sta comunque ricevendo dei segmenti
  - la rete continua a consegnare segmenti a meno di quello mancante che non è giunto a destinazione
  - questo può non essere vero se la rete duplica grandi quantità di segmenti
- Si assume che la perdita del segmento sia un evento puntuale, non un problema di rete generale
  - non c'è ragione di ripartire con lo slow start
  - si spera che la ritrasmissione del segmento mancante sia sufficiente
  - ottenuto il riscontro del segmento mancante si prosegue in congestion avoidance
  - cosa si deve fare tra la ritrasmissione e la ricezione dell'ACK?

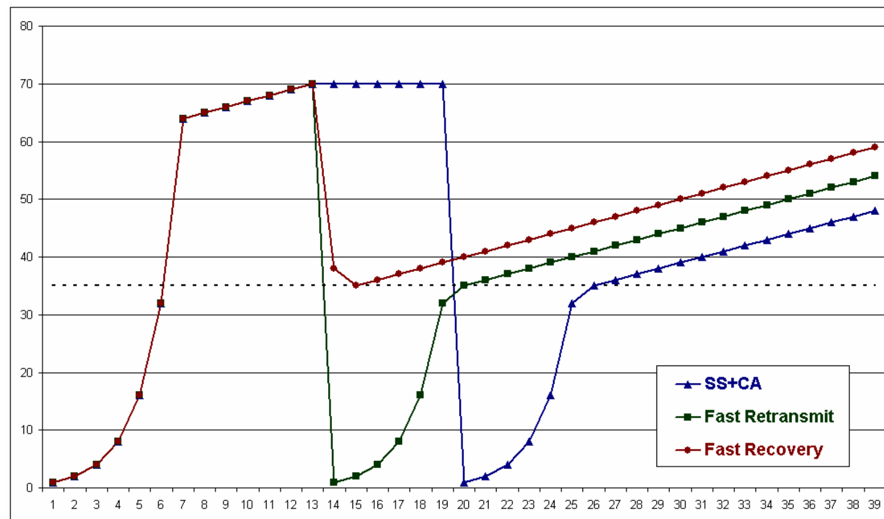
27

Algoritmo di Fast recovery

- Si riduce la soglia
  - SSThr = W/2
- Si pone W = SSThr+3
  - si tiene conto che almeno 3 segmenti successivi al mancante sono stati ricevuti in quanto sono partiti 3 ACK duplicati
- Si continua con la trasmissione aumentando la finestra esponenzialmente
  - ad ogni nuovo ACK duplicato che si riceve si pone W = W + 1
  - si evita così di interrompere la trasmissione a causa del limite imposto dalla finestra
- Quando arriva l'ACK per il segmento perduto, termina la fase di Fast recovery
  - si riparte con Congestion avoidance ponendo W = SSThr

28

Confronto



29

Alcuni commenti

- Fast retransmit e Fast recovery cercano di sfruttare la capacità di TCP di memorizzare i segmenti fuori sequenza
- SSThr viene ridotta per riadattarsi alle condizioni di carico della rete tramite una nuova fase di Congestion avoidance
- Durante il Fast recovery si “gonfia” W di tanto quanti sono i segmenti ricevuti fuori sequenza (che hanno causato gli ACK duplicati)
- Terminata la fase di Fast recovery W viene ridotta per riportarla al valore previsto da Congestion avoidance

30