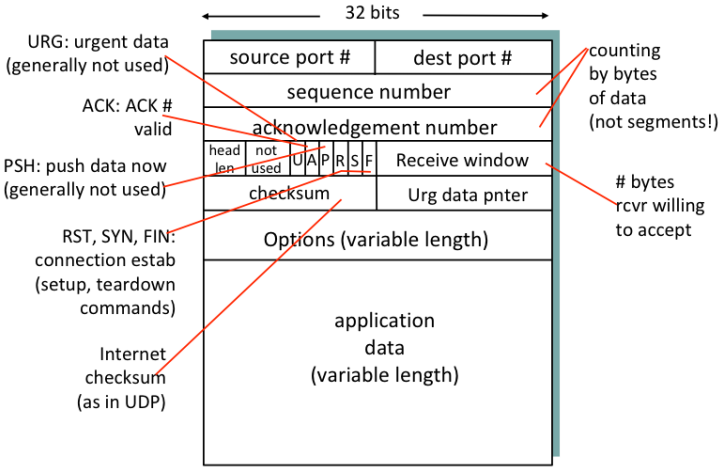
TCP segment structure



Un segmento consiste di due parti: l'header, contenente alcuni campi di servizio, e i dati. Nel campo dati c'è un blocco del messaggio proveniente dal processo applicativo. Il campo dati ha una dimensione fissata dal MSS (massima dimensione di un segmento). Quando il TCP deve inviare un file di grande dimensione, lo frammenta in blocchi di lunghezza MSS (tranne l'ultimo, che ha dimensione minore). Alle applicazioni interattive bastano blocchi di dati di dimensione minore di MSS; ad esempio il campo dati di un messaggio di login come Telnet è molto corto. Poichè l'header TCP è lunga 20 byte (12 byte più grande dell'header UDP).

In comune con l'UDP, l'header del TCP comprende il campo checksum e i campi numeri di porta sorgente e destinazione, che servono per le operazioni di multiplexing/demultiplexing dei dati scambiati con i processi del livello applicazione. Nell'header di un segmento TCP sono contenuti anche:

- un campo di 32 bit riservato al numero di sequenza, un campo di 32 bit riservato a portare il numero della conferma. Entrambi servono ad assicurare l'affidabilità del servizio di trasporto.

- un campo di 16 bit per definire la dimensione della finestra, usato per applicare il controllo del flusso. Il suo valore indica il numero di byte che il ricevitore è in grado di accettare.

- un campo di 4 bit specifica la lunghezza dell'header TCP in unità di doppie word (32 bit). L'header TCP può essere di lunghezza variabile per la presenza del campo option.

- il campo facoltativo, di lunghezza variabile, option viene usato quando il mittente e il destinatario negoziano il parametro (MSS) o il fattore di scala della finestra.

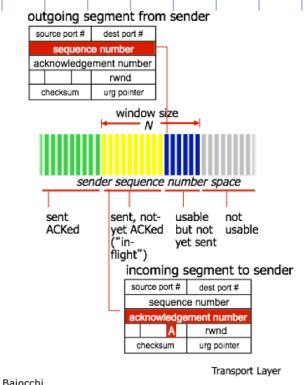
- il campo flag contiene 6 bit.
    - Il bit ACK è usato per indicare che il valore portato dal campo acknowledgment (numero di conferma) è valido, cioè il pacchetto contiene la conferma di un pacchetto ricevuto correttamente.
    - I bit RST, SYN e FIN sono usati per aprire e per rilasciare la connessione.
    - Il valore 1 del bit PSH indica che il ricevitore deve consegnare immediatamente i dati al livello superiore.
    - Il bit URG indica che c'è un dato nel segmento che il processo applicativo mittente ha marcato come "urgente". La posizione dell'ultimo byte di questo dato urgente è specificata dal puntatore di 16-bit. TCP deve informare il processo applicativo del ricevitore quando ci sono dati urgenti e deve passargli il puntatore.

Nella pratica i bit PSH e URG non sono usati, ma sono stati menzionati, perché sono previsti nel formato del pacchetto TCP.

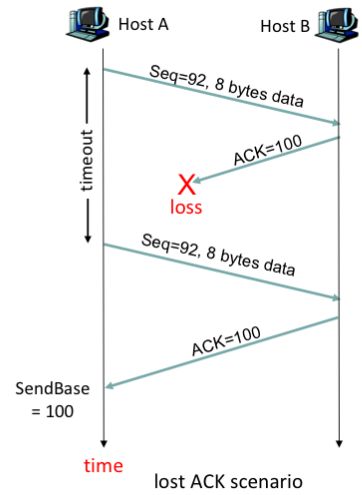
Il TCP vede i dati come un flusso ordinato di byte non strutturati. I **numeri di sequenza**, infatti, si riferiscono alla numerazione dei byte del flusso dei dati trasmessi, non si riferiscono alla numerazione dei segmenti. Il numero di sequenza di un segmento è il numero d'ordine del primo byte nel segmento. Si supponga che un processo nell'host A vuole inviare un messaggio ad un processo sull'host B su una connessione TCP. Il TCP nell'host A implicitamente numera tutti i byte del messaggio. Se il messaggio è un file da 500.000 byte, e se l'MSS è 1000 byte, assumendo che il primo byte abbia numero 0, il TCP costruisce 500 segmenti. Al primo segmento viene assegnato il numero di sequenza 0, al secondo segmento viene assegnato il numero di sequenza 1000, al terzo il numero di sequenza 2000, e così via. Ciascun numero di sequenza viene inserito nel campo **Numero di Sequenza** dell'header del segmento.

Il sender utilizza il campo sequence number per numerare i byte, la connessione TCP utilizzerà tutta la finestra che gli si dà disponibile.

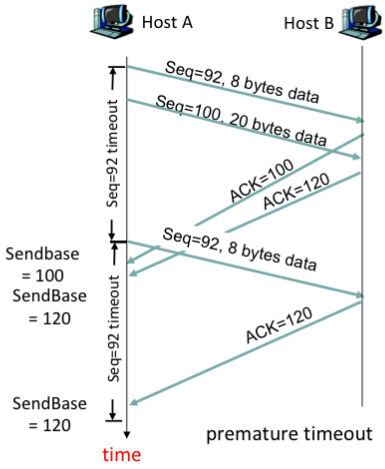
- sequence numbers:
- byte stream "number" of first byte in segment's data
- acknowledgements:
- seq # of next byte expected from other side
- cumulative ACK



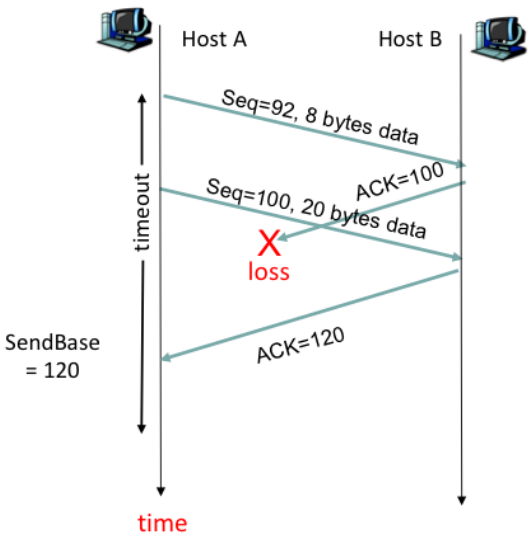
Trasmissione dei dati TCP



lost ACK scenario



premature timeout



Cumulative ACK scenario

(a) lost ACK scenario. La Figura mostra il caso di un host A che invia un segmento all'host B. Si supponga che questo segmento abbia numero di sequenza 92 e contenga 8 byte di dati. Dopo aver inviato il segmento, l'host A passa in attesa di ricevere un segmento da B contenente il valore 100 nel campo "Numero di Conferma". Il segmento dati inviato da A viene ricevuto da B, ma la conferma si danneggia. In questo caso si verifica il timeout e l'host A ritrasmette lo stesso segmento. Naturalmente, quando l'host B riceve il segmento ripetuto il TCP si accorge che quei dati sono stati già ricevuti e scarta il segmento.

(b) premature timeout. Nel secondo caso, l'host A spedisce due segmenti uno di seguito all'altro ed avvia un timer per il primo segmento. Il primo segmento ha numero di sequenza 92 e contiene 8 byte nel campo dati, il secondo segmento ha numero di sequenza 100 e contiene 20 byte nel campo dati. Entrambi i segmenti arrivano integri a B, che quindi invia due conferme separate. Nel campo "Numero di Conferma", il primo ACK contiene 100; il secondo contiene 120. Si supponga che, quando scade il timeout del primo segmento, l'host A non ha ricevuto nessuna delle due conferme. L'host A ritrasmette il primo segmento con numero di sequenza 92 e riavvia il timer. L'host A ritrasmetterà il secondo segmento solo se verificherà il nuovo timeout e non è arrivata la conferma con numero 120. Nell'esempio, si suppone che l'ACK del secondo segmento arrivi prima che scada il timeout e quindi non viene ritrasmesso.

(c) cumulative ACK scenario. Nel terzo caso, l'host A invia due segmenti come nel caso precedente. La conferma del primo segmento si perde nella rete, ma prima che scada il timeout del primo segmento giunge la conferma contenente il numero 120. L'host A quindi sa che entrambi i segmenti sono giunti a destinazione e non ripete nessuno di essi.

Raddoppiare l'intervallo di timeout

- Una prima modifica al comportamento del TCP consiste nel regolare la durata dell'interval timer osservando gli eventi timeout. Come visto, ogni volta che si verifica un timeout, il TCP ritrasmette il segmento non ancora confermato contenente il più piccolo numero di sequenza. Ma ad ogni ritrasmissione, il TCP raddoppia la durata dell'interval timer, anzichè calcolarlo sulla base dei valori medi e della deviazione standard dell'RTT. Ad esempio, se il TimeoutInterval associato al segmento più vecchio, non ancora confermato, è 0.75 sec, quando il TCP ritrasmette questo segmento, reimposta l'interval timer al valore 1.5 sec. Se si verifica nuovamente un timeout, il TCP ritrasmette il segmento e imposta l'interval timer al valore 3 sec. Quindi l'interval timer ha una crescita esponenziale.

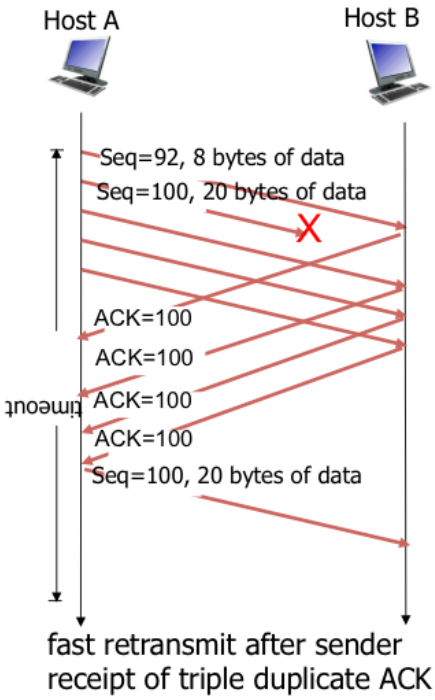
- In ogni altro caso in cui il timer viene avviato (ricezione di un ACK) il TimeoutInterval viene calcolato con i valori più recenti di EstimatedRTT e di DevRTT. Questa modifica contribuisce, anche se in forma marginale, a controllare la congestione. Il motivo più probabile della generazione del timeout è la congestione della rete, cioè si stanno accumulando troppi pacchetti nelle code dei router che si trovano lungo il percorso tra gli host sorgente e destinazione, provocando la sovrascrittura o il ritardo dei pacchetti. Quando la rete è prossima alla congestione, se la sorgente continua a ritrasmettere pacchetti, la congestione si può aggravare. Invece, il TCP riduce il flusso dei pacchetti emessi.

Ritrasmissione

- Uno dei problemi con la ritrasmissione dei pacchetti dovuta alla generazione del timeout è che il timeout potrebbe essere troppo lungo. Quando si perde un segmento, questo lungo intervallo di attesa costringe il mittente a ritardare la ripetizione del pacchetto perso, incidendo sul ritardo totale. Il mittente, però, può riconoscere la perdita di un pacchetto prima che si verifichi il timeout, in particolare quando vede due pacchetti di conferma (ACK) duplicati. Un ACK duplicato è un pacchetto di conferma che era stato già ricevuto. Per comprendere il comportamento del mittente in presenza di conferme duplicate, si pensi perchè il ricevitore ha ripetuto la conferma. La Tabella riepiloga la politica di generazione delle conferme da parte del ricevitore. Quando al TCP ricevitore arriva un segmento con numero di sequenza maggiore di quello atteso, individua un vuoto nel flusso dei dati, cioè un segmento mancante. Questo indica che si è perso un segmento nella rete.

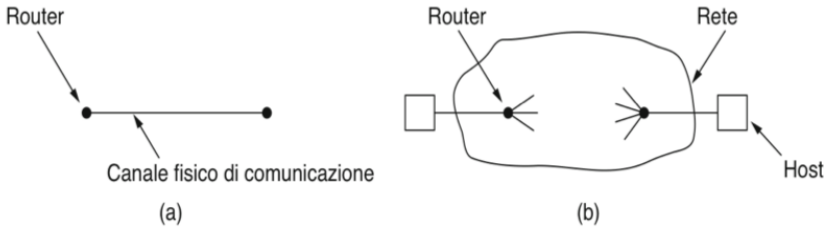
Ci si comporta come se il pacchetto fosse perso e si ritrasmette.

TCP ritrasmissione veloce



RTT è il tempo necessario per trasmettere un messaggio avanti e indietro tra i due collegamenti.

(a) Ambiente del livello di data link
(b) Ambiente del livello di trasporto



Percentile = media + multiplo dev. standard

- $P(X \leq X_{90}) = 0.90$
- $P(X \leq X_p) = p$
- $X = M + kS$
- $M = E[X]$
- $S = STD(X)$

mano mano li raccolgono online valori di RTT, con una stima iniziale, poi si aggiornano mano mano dev. standard e media (avere un percentile grande è "attraente" da un punto di vista di connessione)

Il valore stimato al passo n:

$$X_{med}(n) = (1-\alpha) X_{med}(n-1) + \alpha X_n$$

Round Trip Time nel TCP

Domanda: come impostare il valore del timeout del TCP?

Il TCP usa il timeout generato dal timer per decidere la ritrasmissione di un pacchetto che si ritiene perso. Dopo avere inviato un pacchetto, il trasmettitore resta in attesa del pacchetto di conferma. La scelta della durata del tempo di attesa dipende da due considerazioni. L'intervallo di attesa deve essere di poco maggiore del round-trip time (RTT), cioè il tempo trascorso dal momento in cui viene inviato un segmento al momento in cui viene ricevuta la conferma. Se l'attesa fosse di durata minore si avrebbero delle ritrasmissioni inutili. La soluzione del problema richiede il calcolo dell'RTT. Inoltre ci si chiede se è opportuno avviare un timer per ogni segmento non confermato.

- Deve essere più lungo del RTT.
  - Ma l'RTT varia.
- Se è troppo breve: si ha un timeout prematuro.
  - Con ritrasmissioni non necessarie.
- Se è troppo lungo: la reazione alla perdita di segmenti è lenta.

Domanda: Come stimare il RTT?

- SampleRTT (RTT campionato): il tempo è misurato dalla trasmissione del segmento fino alla ricezione dell'ACK.
  - Ignorare le ritrasmissioni.
- Il SampleRTT varierà, vuole che il RTT stimati sia più "regolare" (smooth).

- La media di diverse misurazioni recenti, non solo il SampleRTT corrente.
- Aggiungere un margine per tenere conto delle fluttuazioni del SampleRTT.

Per comprendere il modo in cui il TCP gestisce il timer si consideri da cosa dipende il round-trip time tra il mittente e il ricevitore. L'RTT campione, denotato `SampleRTT`, di un segmento è il tempo trascorso dal momento in cui il segmento viene trasmesso (cioè consegnato al livello IP) al momento in cui si riceve un segmento di conferma (ACK) per quel segmento. Il TCP prende una misura del `SampleRTT` di un segmento trasmesso e non ancora confermato. Questa misura viene, quindi, effettuata ogni RTT secondi. Inoltre il TCP non calcola il `SampleRTT` di un segmento che è stato ritrasmesso. Il valore del `SampleRTT` varierà da un segmento al successivo, a causa del traffico irregolare presente in rete, quindi la stima attendibile dell'RTT è il valore medio dei valori `SampleRTT` misurati. Questo valore medio dei `SampleRTT`, mantenuto dal TCP, è chiamata `EstimatedRTT`. Per ogni nuovo `SampleRTT`, il TCP aggiorna la media `EstimatedRTT` secondo la seguente formula:

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

- Media mobile esponenziale ponderata.
- L'influenza del campione precedente diminuisce esponenzialmente veloce.
- Valore tipico: $\alpha = 0.125$.

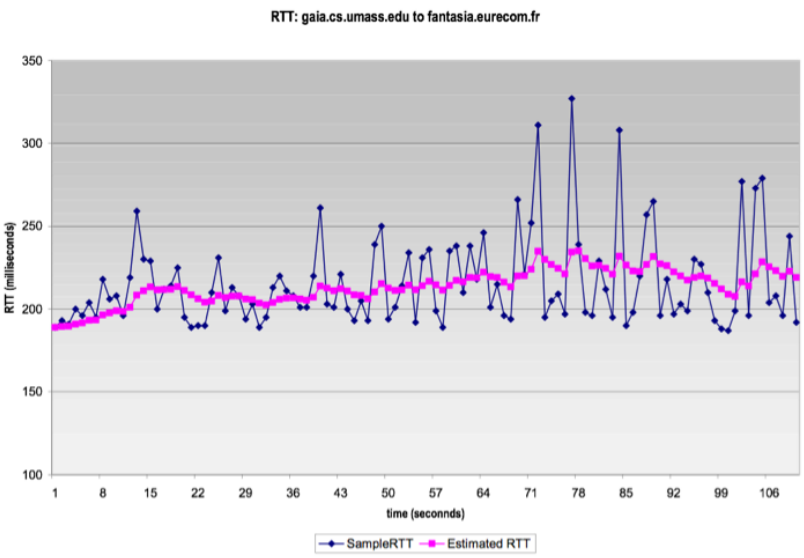
$$Dev\_RTT = (1-\beta) \cdot Dev\_RTT + \beta \cdot (SampleRTT - EstimatedRTT)$$

- Stima della deviazione standard.
- Valore tipico: $\beta = 0.25$.

Con il valore medio EstimatedRTT e con la deviazione standard DevRTT, si può scegliere il valore da assegnare al timer per segnalare il timeout. La durata di questo intervallo è ottenuto da un compromesso tra due situazioni estreme. Dovrebbe essere maggiore o uguale di EstimatedRTT, per evitare inutili ritrasmissioni di pacchetti. Ma il timer non dovrebbe essere impostato ad un valore troppo più alto di EstimatedRTT per non introdurre un ritardo eccessivo quando un pacchetto realmente si perde. Il timer dovrebbe essre inizializzato con EstimatedRTT più un certo margine. Il margine dovrebbe essere grande quando le oscillazioni dei valori SampleRTT sono grandi e dovrebbe essere piccolo quando le variazioni di sampleRTT sono piccole. Il metodo per calcolare la durata del timer è:

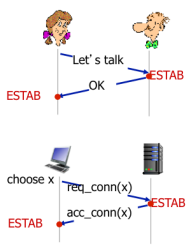
$$TimeoutInterval = EstimatedRTT + 4 \cdot Dev\_RTT$$

Example RTT estimation



Agreeing to establish a connection

2-way handshake:



Gestione della connessione nel TCP

- La connessione TCP è stabilita prima dello scambio di segmenti dati per inizializzare le variabili TCP:
  - Sequenza di #
  - Informazione di controllo del flusso e del buffer (ad esempio RcvWindow)

Ci sono tre modi di accordarsi:

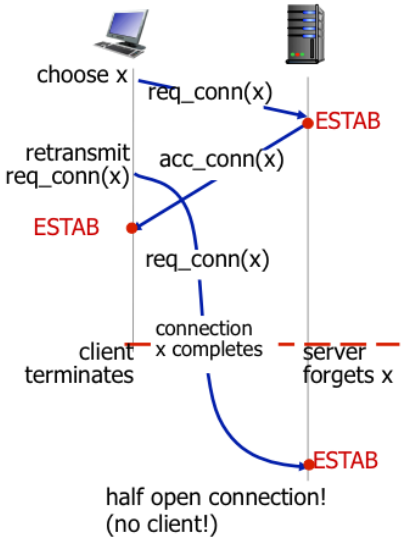
Step 1: Il TCP client invia un segmento speciale al TCP server. È un segmento che non trasporta dati del processo applicativo, ma possiede la flag SYN nell'intestazione del segmento a valore 1. Per questo motivo il segmento è denominato SYN segment. Inoltre il client sceglie un numero di sequenza iniziale (client_isn) e lo inserisce nel campo numero di sequenza del segmento. Il segmento viene consegnato al processo del livello Rete per essere inviato. La scelta di un numero di sequenza iniziale per il segmento di apertura con un valore casuale ha anche lo scopo di impedire attacchi alla sicurezza.

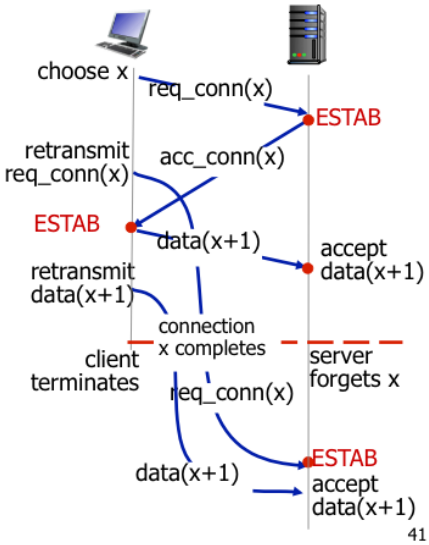
Step 2: Quando il datagramma IP contenente il segmento con il bit SYN=1 arriva al destinatario, il server estrae il segmento SYN dal datagramma, riserva lo spazio per i buffer e per le variabili necessarie a gestire la connessione ed invia un segmento al client che indica che la connessione è pronta. Nemmeno questo segmento contiene dati del processo applicativo. Nell'intestazione contiene tre importanti informazioni: primo, la flag SYN è impostata a 1; Secondo, il campo "Numero di Conferma" contiene il valore client_isn+1. Infine, il server sceglie il proprio numero di sequenza iniziale server_isn e inserisce questo valore nel campo numero di sequenza del segmento. Questo segmento di risposta significa che il ricevitore ha confermato la ricezione del pacchetto con il numero di sequenza client_isn, è pronto a stabilire la connessione e sta indicando al client il proprio numero di sequenza iniziale, server_isn. Il segmento di conferma è anche detto SYNACK.

Step 3: Quando il client riceve il segmento SYNACK dal server, riserva lo spazio per il buffer e per le variabili della connessione. Il client invia un ultimo segmento che conferma al server la ricezione del segmento SYNACK (connection-confirm). A questo scopo, il client scrive il valoreserver_isn+1 nel campo "Numero di Conferma" dell'intestazione del segmento TCP. Il bit SYN è impostato a 0 perchè la connessione è stata aperta. Questo terzo segmento può anche contenere dati che il client invia al server.

Quando i tre passi sono stati completati il client ed il server possono scambiarsi segmenti dati. In ognuno di questi segmenti il bit SYN avrà il valore 0. Notare che per aprire la connessione sono stati usati tre segmenti. La fase di apertura connessione è anche detta three-way handshake.

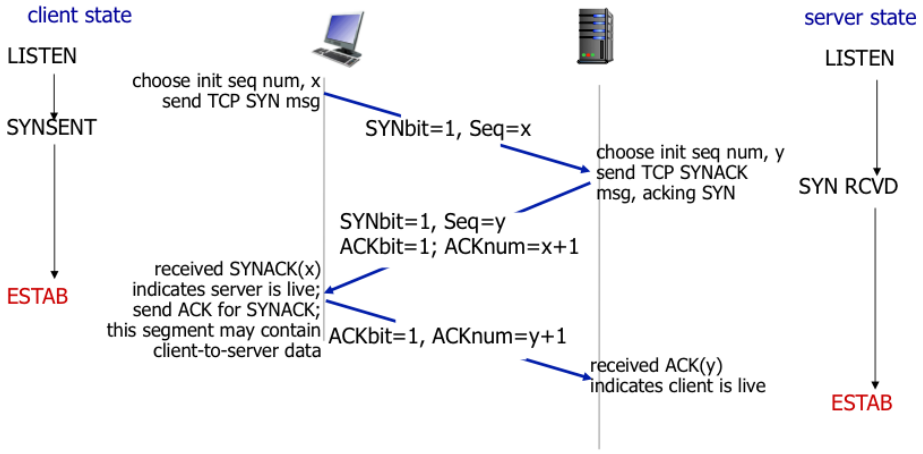
2-way handshake failure scenarios:





41

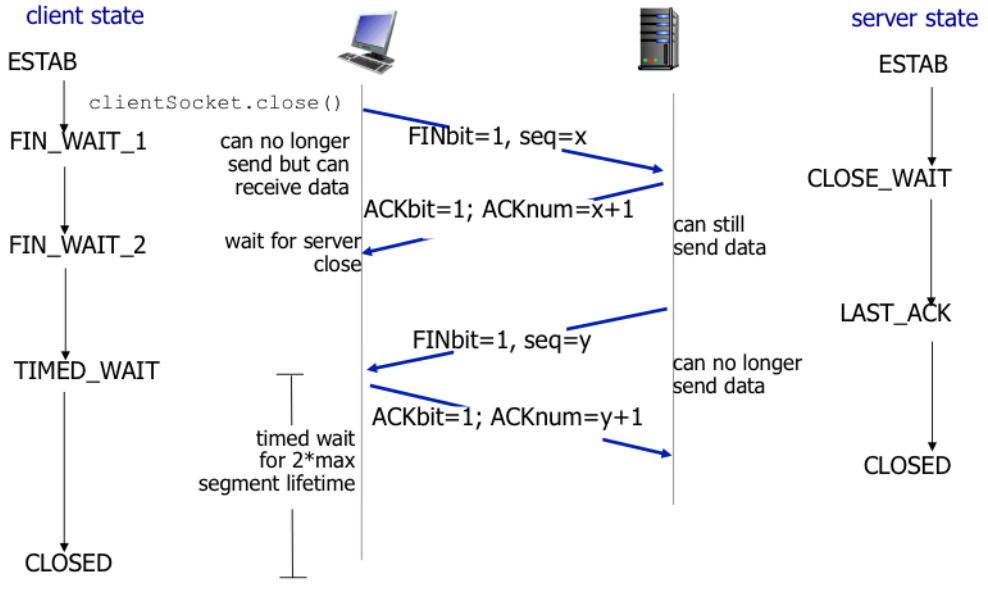
TCP 3-way handshake



Scopo della "3-way hanshake" del TCP

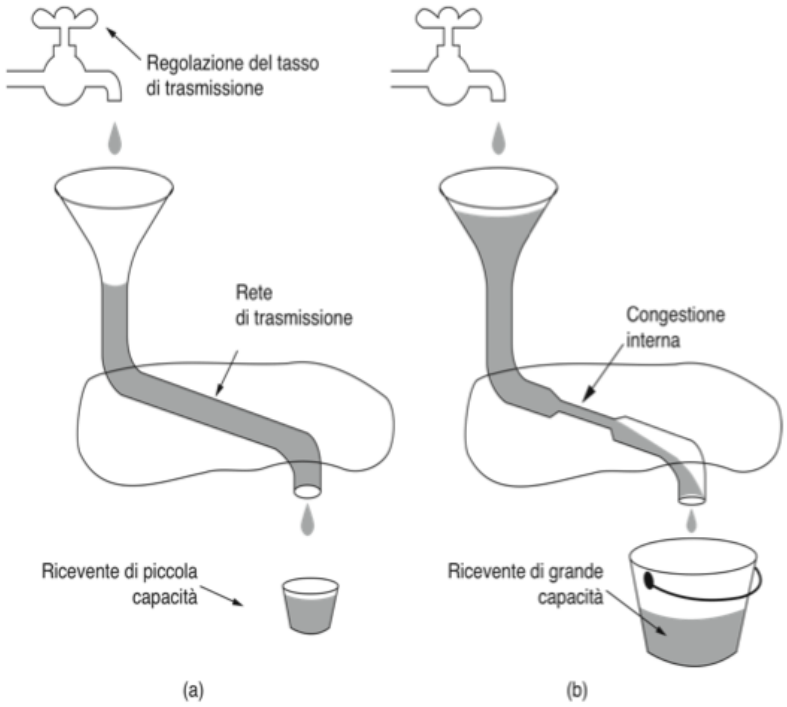
- Consente al server di testare che il cliente sia davvero all'indirizzo che dice di stare usando.
- Sincronizza l'uso di numeri di sequenza.
- È arrivato ad essere usato dalle middleboxe (includendo NAT e firewall).
- Consente ai due endpoint di essere d'accordo sull'uso delle opzioni del protocollo.
- Consente ad entrambi gli endpoint di stimare il RTT corrente prima dell'invio dei dati.
- Consente al client di chiudere tutte le connessioni dulicate che il server apre involontariamente in casi dove all'apertura di un pacchetto da parte del cliente, esso sia duplicato.

TCP: closing a connection



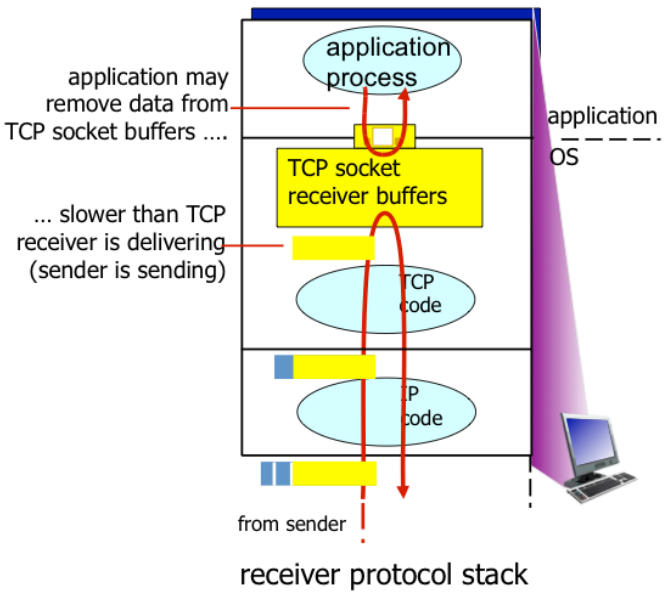
Ognuno dei due processi può chiudere la connessione. La fase di chiusura della connessione serve a rilasciare le risorse (affinchè non vengano lasciate in giro). Si supponga che il client chieda la chiusura. Il client invia un pacchetto di servizio al processo server. In questo segmento il bit FIN, nell'intestazione del pacchetto, è impostato a valore 1. Quando il server riceve questo pacchetto invia il pacchetto di conferma e poi manda il pacchetto di chiusura della sua connessione, contenente nell'header il bit FIN=1. Infine il client invia la conferma. A questo punto le risorse sono state rilasciate su entrambi gli host.

Controllo di flusso vs. controllo di congestione



Nel gioco di trasferimento di dati abbiamo tre elementi: sender, receiver e rete.
Il sender potrebbe inviare dati in sovraccarico, ossia manda dati troppo velocemente rispetto a quanto i dati della rete sono in grado della rete.
Nell'analogia idraulica il sender è colui che inserisce il fluido. La rete è colei che trasporta il fluido.
Controllo di flusso è il nome che si da al meccanismo per evitare il sovraccarico al ricevitore. Il controllo di congestione è invece il meccanismo per evitare il sovraccarico dei rami della rete.

Flow control framework



receiver protocol stack

Il TCP gestisce un buffer di ricezione su ciascun estremo della connessione, nel quale memorizza i messaggi in arrivo, in attesa che il processo applicativo li prelevi. Il TCP avverte il processo destinatario che c'è un messaggio da prelevare, ma il processo applicativo potrebbe ritardare la lettura, ad esempio perché è impegnato in altre operazioni. Se il ricevitore è lento e il trasmettitore invia messaggi, si potrebbe verificare una sovrascrittura nel buffer di ricezione. Per questo il TCP offre, alle applicazioni, il servizio di controllo del flusso. Il controllo del flusso realizza una regolazione della frequenza di invio dei segmenti, cioè il processo più veloce si adegua ai tempi del processo più lento.

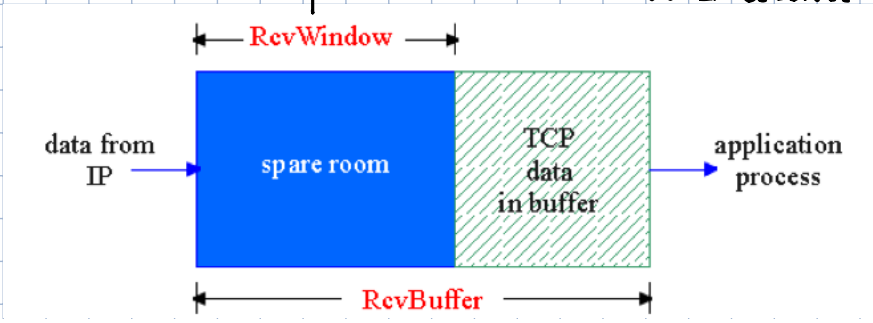
Si deve distinguere il comportamento del TCP quando rallenta il flusso di generazione dei segmenti per evitare la congestione e quando rallenta il flusso per adeguarsi alla velocità del processo applicativo. Nel seguito si descriverà il modo in cui il TCP offre alle applicazioni il servizio di controllo del flusso. Si assume che il ricevitore TCP scarti i segmenti ricevuti fuori ordine.

- Il lato ricezione della connessione TCP ha un buffer di ricezione.
- Servizio di corrispondenza rapida: facendo corrispondere il tasso di trasmissione alla frequenza di drenaggio delle applicazioni riceventi.
- Il processo dell'applicazione può essere lento alla lettura dal buffer.

Dice quanto spazio libero c'è nel buffer.

Trasmission window = W

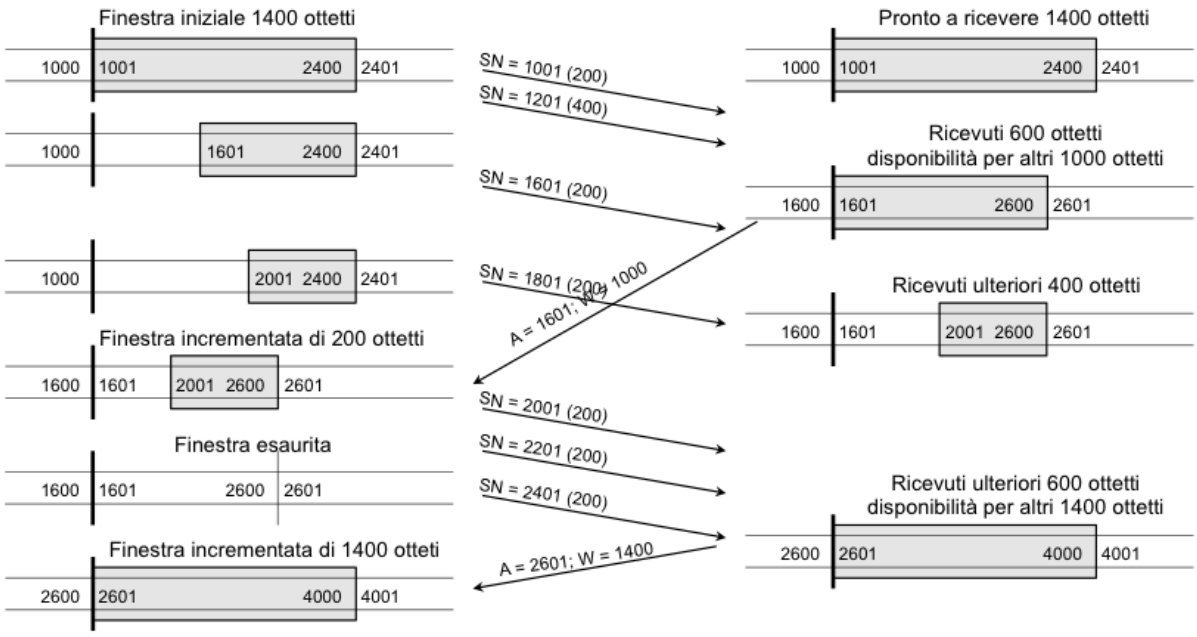
W ≤ receiver window



Per gestire il controllo del flusso, il TCP trasmettitore usa una variabile chiamata **receive window**. Il valore contenuto in questa variabile informa il mittente di quanto spazio libero esiste nel buffer del ricevitore. In una connessione full-duplex ciascun trasmettitore vede la finestra di ricezione del proprio destinatario. Inoltre l'ampiezza della finestra di ricezione cambia dinamicamente per tutto il tempo che esiste la connessione.

- Il receiver annuncia lo spazio di riserva includendo il valore dilla finestre di ricezione (RcvWindow) nei segmenti.
- Il sender limita i dati senza riscontro (senza l'ACK) al RcvWindow.
    - Garantisce che il buffer di ricezione non si sovraccarichi.

Esempio:



Si supponga che l'host A debba inviare un file di grandi dimensioni all'host B servendosi di una connessione. L'host B riserva un buffer di ricezione a questa connessione e indica la sua dimensione con **RcvBuffer**. Allo scopo di sincronizzare i processi mittente e destinatario, si usano le seguenti variabili:

LastByteRead = il numero dell'ultimo byte nel flusso dei dati che il processo applicativo destinatario nell'host B ha letto dal buffer.

LastByteRcvd = Il numero dell'ultimo byte ricevuto memorizzato nel buffer del ricevitore, sull'host B.

Affinchè non si verifichi la sovrascrittura nel buffer di ricezione, deve valere la condizione:

`LastByteRcvd - LastByteRead ≤ RcvBuffer`

L'ampiezza della finestra di ricezione, indicata con **RcvWindow**, contiene la quantità di spazio libero nel buffer:

`RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)`

Il destinatario, l'host B, fa conoscere al mittente, l'host A, quanto spazio libero ha nel buffer di ricezione scrivendo il valore corrente della variabile RcvWindow nel campo "larghezza della finestra" del segmento. Inizialmente l'host B imposta RcvWindow = RcvBuffer.

L'host A, a sua volta, deve gestire altre due variabili, **LastByteSent** e **LastByteAcked**, il cui significato dovrebbe essere ovvio. La differenza tra queste due variabili, **LastByteSent - LastByteAcked**, è il numero di byte che il trasmettitore ha spedito ma dei quali non ha ancora ricevuto la conferma. Mantenendo la quantità di dati non confermati minore del valore **RcvWindow**, l'host A si assicura che non sovrascriverà i dati nel buffer di ricezione sull'host B. Per tutta la durata della connessione, l'host A deve rispettare la condizione:

`LastByteSent - LastByteAcked ≤ RcvWindow.`

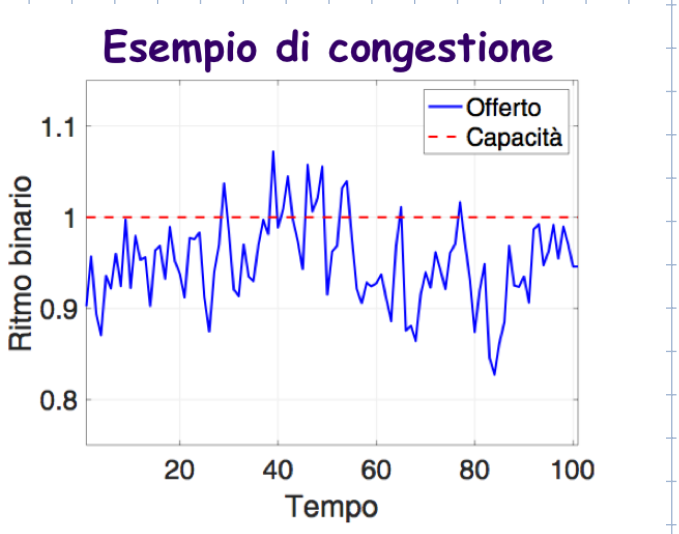
Si supponga che il buffer sia pieno e quindi **RcvWindow = 0**. Il trasmettitore viene avvertito di questa situazione, ma per ipotesi, l'host B non ha più niente da comunicare all'host A, quindi dopo che il processo destinatario svuota il buffer di B, non ci sono segmenti da inviare ad A nei quali inserire il nuovo valore di RcvWindow. Il TCP invia segmenti solo se ci sono dati di un processo applicativo da inviare o se ci sono conferme da spedire. Se si verifica una simile situazione l'host A non saprà mai che si è liberato spazio nel buffer del ricevitore, e resta bloccato, senza poter trasmettere. Le specifiche del TCP impongono che, quando RcvWindow = 0, il trasmettitore invii segmenti contenenti solo un byte allo scopo di forzare la conferma.

UDP non fornisce il servizio "controllo del flusso". UDP inserisce i dati ricevuti in una coda di dimensione finita, dalla quale il processo destinatario, tramite il socket, li leggerà. Se il processo non legge i dati con la necessaria velocità, il trasmettitore sovrascrive i dati.

Se mando dati = W ⟹ buffer = 0. In questo modo però il ricevitore non può mandare nulla se non l'ultimo riscontro

Controllo di congestione

Meccanismo per evitare il sovraccarico nei rami della rete e quindi dei buffer che sono a monte dei rami.
La risorsa condivisa in questo caso è la capacità della rete, bisogna evitare che tutti i server sovraccarichino i rami.



Congestione: somma dei ritmi binari sul collegamento > capacità.

A seguito della congestione si verificano :

- Perdite di pacchetti (sovraccarico del buffer al router)
- Lunghi ritardi (code nei buffer del router)
- Legato al concetto di "equità" (non potendo dare a tutti quelli che chiedono, darò meno, ma secondo quale criterio?"