lo strato di trasporto

in rete locale e in rete geografica

g. di battista

nota di copyright

servizio offerto

- il servizio offerto dallo strato di trasporto deve essere affidabile ed è normalmente *connesso*
- i processi che usano le primitive di trasporto assumono che esse siano *affidabili*
- le primitive di servizio dello strato di trasporto sono offerte ad una popolazione di utenti-programmatori molto *ampia*
  - primitive facili da usare

primitive di trasporto

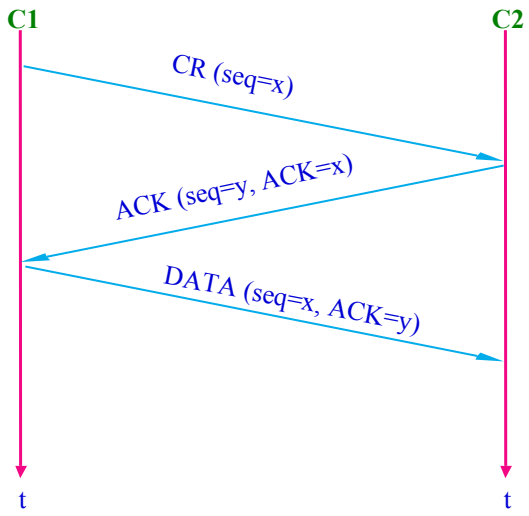
- esempi di primitive di trasporto:

– listen	in attesa di qualche connessione
– connect	tentativo di instaurare una connessione
– send	invio di dati
– receive	in attesa di dati
– disconnect	rilascio della connessione

instaurazione di connessioni

- metodo three-way handshake (calcolatori C1 e C2)
  - C1 sceglie il numero di sequenza iniziale x per le proprie unita' dati
  - C1 invia una connection request con x a C2
  - C2 riceve la connection request con x
  - C2 sceglie il proprio numero di sequenza iniziale y
  - C2 invia una connection accepted riscontrando x e proponendo il proprio numero iniziale y a C1
  - C1 riceve la connection accepted da C2
  - C1 riscontra y a C2

instaurazione di connessioni



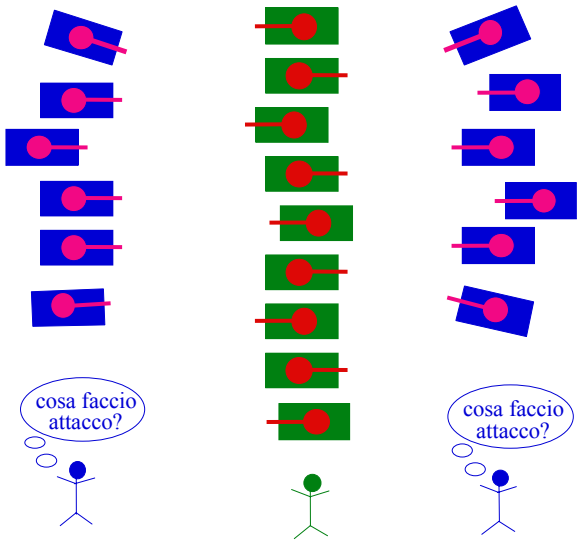
rilascio di connessioni

- attenzione: è più difficile di quanto si possa immaginare
- se il rilascio è troppo rudimentale si possono perdere dati
- esempio:
  - supponiamo di rilasciare la connessione come si termina una telefonata
  - ad un certo punto l'interlocutore A rilascia unilateralmente la connessione
  - se B aveva nel frattempo mandato dei dati c'è il rischio che arrivino ad A dopo che A ha "attaccato il telefono"

rilascio di connessioni

- rilascio simmetrico:
  - ogni direzione e' rilasciata in modo indipendente dall'altra
  - quando un interlocutore ha rilasciato continua comunque a ricevere dati dall'altro
- quando la connessione può dirsi effettivamente rilasciata?
- *problema dei due eserciti*

problema dei due eserciti



problema dei due eserciti

- i blu in totale sono piu' dei verdi ma i verdi sono più di ognuna delle due parti blu
- se i blu attaccano contemporaneamente vincono, se attaccano separatamente perdono
- i blu vogliono sincronizzarsi ma per farlo devono mandare emissari attraverso il campo dei verdi i quali potrebbero catturarli
- esiste un protocollo che faccia vincere i blu con certezza?

problema dei due eserciti

- supponiamo che:
  - il comandante blu 1 mandi il messaggio "attacchiamo oggi alle 11:30, che ne pensi?"
  - il messaggio arrivi e che il comandante blu 2 mandi il messaggio "ok"
  - il messaggio del comandante blu 2 arrivi al comandante blu 1
- l'attacco avverrà?
- probabilmente no visto che il comandante blu 2 non sa se il suo "ok" è arrivato a destinazione (se l' "ok" non è arrivato il comandante blu 1 non attaccherà e quindi l'attacco del comandante blu 2 sarà sconfitto)

problema dei due eserciti

- complichiamo il protocollo:
  - supponiamo che il comandante blu 1 debba riscontrare l' "ok"
  - supponiamo che il comandante blu 1 riscontri
- a questo punto è lui a non sapere se attaccare, visto che non sa se il suo riscontro è arrivato
- in realtà non esiste un protocollo che garantisca il successo
- il problema di rilasciare o no una connessione è assimilabile a quello di decidere se attaccare; in pratica ci si accontenta di tecniche tipo three-way handshake con timeout

tcp

- rfc 793, 1122 e 1323
- servizio bidirezionale contemporaneo (full duplex) punto-punto; multicasting e broadcasting non supportati
- tcp è il primo tra tutti i protocolli visti fino ad ora nella internet protocol suite a garantire l'affidabilità del servizio di trasmissione
- affidabilità conseguita usando tecniche che prevedono la ritrasmissione dei dati in caso di necessità, queste tecniche si basano su acknowledgement e controllo di flusso
- connessione fra due processi, indirizzo ip ed un numero di **port** da entrambe le parti

tcp

- una volta che la connessione è stabilita il processo utente manda i dati allo strato tcp che li manderà correttamente allo strato tcp del ricevente, che li passa a sua volta al processo destinazione
- l'implementazione della connessione fa in modo che i processi utente vedano un canale di comunicazione privato, permettendo una programmazione a livello utente in cui *la rete è trasparente*

port

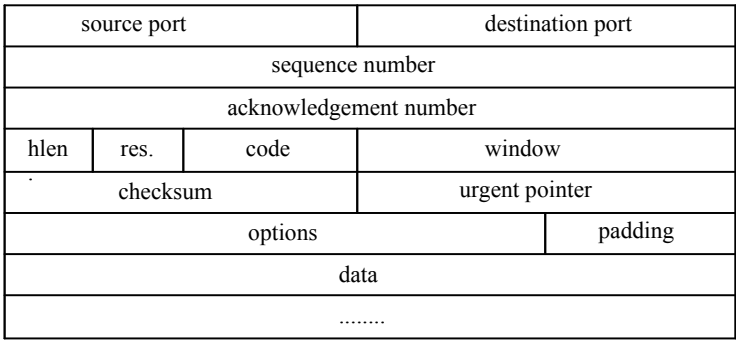
- tcp specifica come distinguere più destinazioni (processi) su una stessa macchina (i port)
- i port sono i tcp tsap
- il numero di port ha 2 byte
- i port < 256 sono riservati per servizi standard (es ftp = 21, telnet=23); rfc 1700

pacchetti e byte

- ogni byte spedito ha un numero di sequenza a 32 bit
  - 32 bit consentono di avere reset della sequenza abbastanza infrequenti
- i numeri di sequenza sono usati anche per gli ack
- la t-pdu tcp si chiama segment
- segment = header + dati (opzionali)
- limiti del segment:
  - limite di 65535 byte per i dati ip
  - mtu (maximum transfer unit); da cosa dipende?

segment

- in tcp il pacchetto è chiamato segment, i segment vengono scambiati per stabilire connessioni, per trasferire dati, per mandare acknowledgement, per chiudere connessioni
- formato del segment:



segment

- il sequence number stabilisce la posizione del pacchetto di dati nel flusso di informazioni del mittente, questi infatti trasferisce un flusso di dati generando un certo numero di pacchetti da mandare attraverso la rete
- l'acknowledgement number viene utilizzato per gestire il meccanismo di ritrasmissione (prossimo byte atteso); il sequence number si riferisce al flusso che va nella stessa direzione del segmento mentre l'acknoledgement number si riferisce al flusso che va in direzione opposta al segmento
- go-back-N orientato al byte

segment

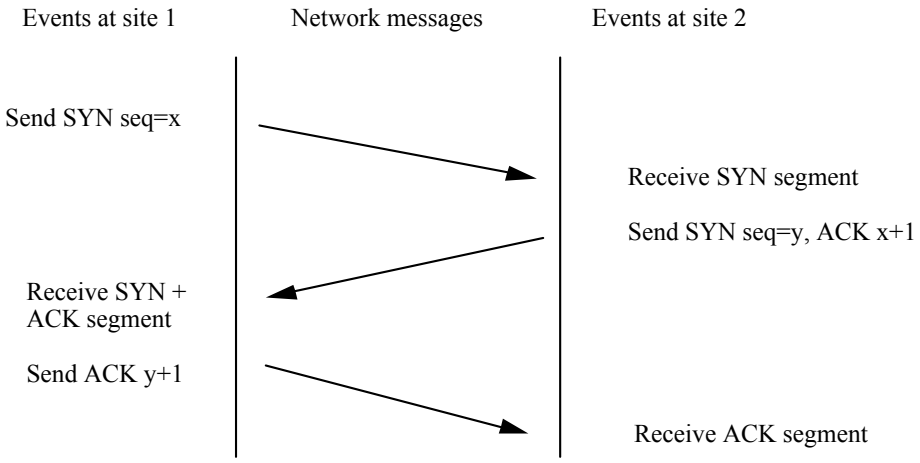
- hlen: numero di parole di 32 bit dell' intestazione
- code (6 bit) determina il tipo di messaggio contenuto nel segmento

  bit (da sinistra a destra) — significato
  - URG — urgent pointer field is valid
  - ACK — acknowledgement field is valid
  - PSH — this segment requests a push
  - RST — reset the connection
  - SYN — synchronize sequence numbers
  - FIN — sender has reached end of its byte stream; rilascio

segment

- urgent pointer specifica l'offset in cui cercare i dati urgenti
- window specifica la ampiezza corrente della finestra: quanti byte si possono ricevere dopo l'ultimo riscontrato; può assumere il valore 0; controllo di flusso
- checksum interessa l'intero segment + gli indirizzi ip; violazione della gerarchia dei protocolli
- opzioni: la più importante specifica la massima ampiezza del campo dati; negoziazione durante l'instaurazione della connessione
  - gli host sono obbligati ad accettare almeno 536 byte di dati

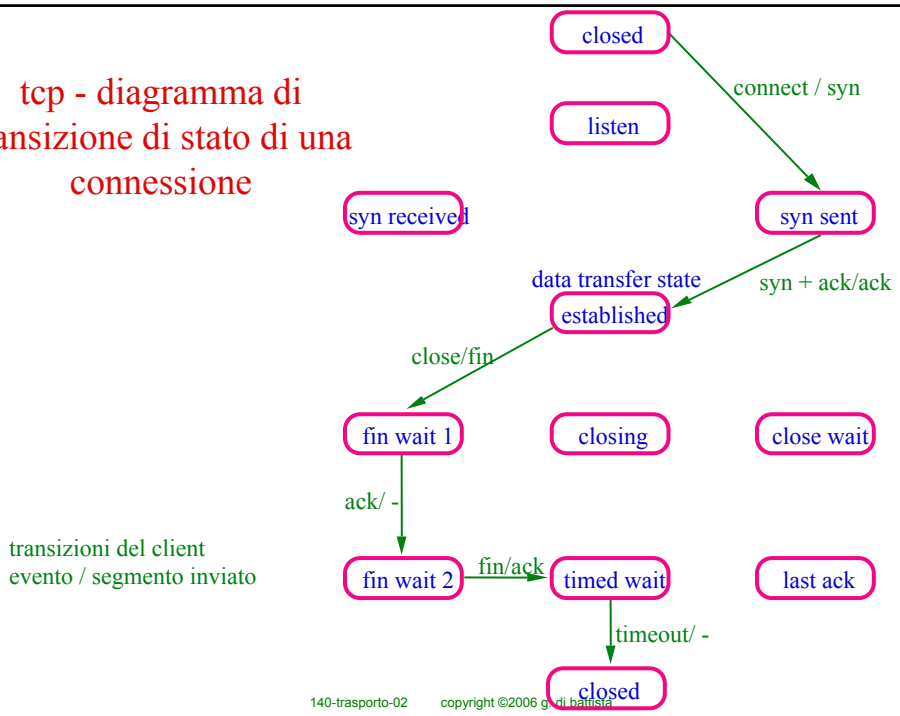
instaurazione di una connessione



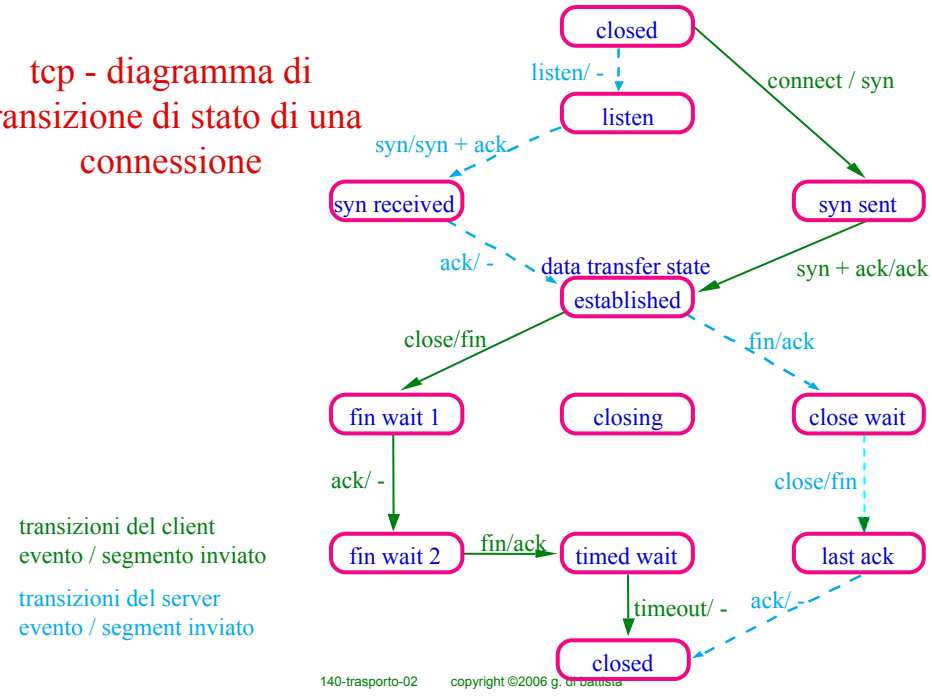
rilascio di una connessione

- per rilasciare la connessione una delle due parti manda un segment con FIN=1; non ha piu' dati da trasmettere
- quando la parte che ha mandato FIN=1 riceve in riscontro un ACK considera chiuso il suo flusso
- i dati possono continuare a fluire in direzione opposta
- la connessione e' rilasciata quando entrambi i flussi sono chiusi
- il rilascio normale richiede 4 segment
- il primo ACK ed il secondo FIN possono essere a bordo dello stesso segment
- uso di timeout se ACK non arriva; timeout = 2 * tempo di vita di un pacchetto

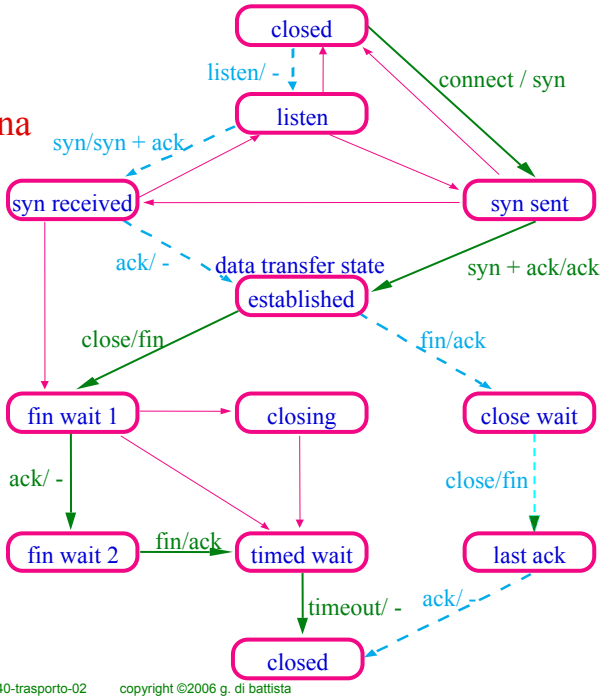
tcp - diagramma di transizione di stato di una connessione



tcp - diagramma di transizione di stato di una connessione



tcp - diagramma di transizione di stato di una connessione



udp

- udp fornisce un servizio di trasmissione non affidabile, il mittente cioe' non ha informazione sull'esito della ricezione
- viene utilizzato da servizi di rete del tipo:
  - Sincronizzazione del tempo su rete
  - trivial file transfer protocol.
- l' header e' diviso in 4 campi di 16 bit che specificano il port da cui viene mandato il messaggio, il port destinazione, la lunghezza del datagram ed il checksum

