

UDP → connection-less → unreliable

prima di poter trasmettere dati deve
↑ attivare connessione

IP e PORTA → connessione TCP identificata dagli indirizzi IP dei due host
host e dalle porte utilizzate sui due host
socket ↔ socket

TCP → connection-oriented → affidabile → connessioni full-duplex → il traffico può procedere in entrambe le direzioni contemporaneamente punto a punto

FLUSSO DI BYTE = SEGMENTO → confermato da un ACK (20 byte)

frazionato in blocchi
intestazione fissa di 20 byte seguita da 0 o più byte di dati

consegna ordinata dei dati una volta sola "at most once"
ritrasmissione di quelli persi

controllo di errore sui pacchetti grazie al campo Checksum

Controllo di flusso → non vengono spediti più dati di quanti ne può ricevere la sorgente in un dato momento

Controllo di congestione → si cerca di non spedire più dati di quanti ne può smaltire la rete

PERDITE DI SEGMENTI

si ritiene sia andato perso un segmento se

scade l' RTO (time-out) arriva 3 dupACK

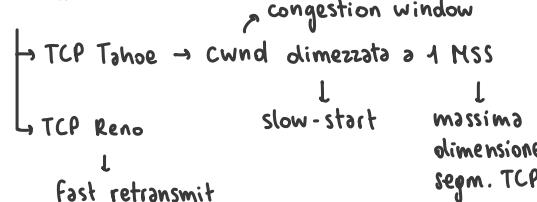
se non sono arrivati ACK non duplicati per un periodo più lungo di RTO

RTO è calcolato dinamicamente sulla base di RTT

DECENTRALIZZATO a retroazione

ogni istanza del TCP decide per conto suo

generati alla ricezione dei segmenti successivi a quello mancante



APERTURA DI UNA CONNESSIONE

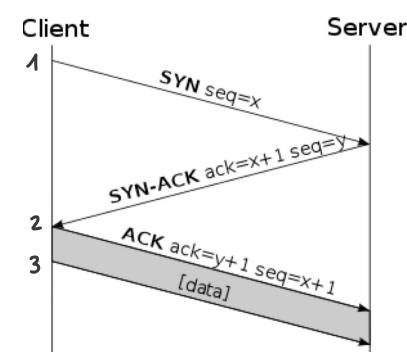
→ three-way handshake → stretta di mano

in 3 passaggi → necessità di scambiare 3 messaggi tra host mittente e host ricevente affinché la connessione sia instaurata correttamente

nel traffico TCP i segmenti SYN stabiliscono nuove connessioni, mentre quelli con il flag non attivo appartengono a connessioni già instaurate

i segmenti utilizzati durante l'handshake sono solitamente "solo header" → campo Data vuoto

ISN → SEQUENCE NUMBER → indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall'Initial Sequence Number (ISN)
deciso all'inizio della connessione



✓

CHIUSURA DI UNA CONNESSIONE → doppio two-way handshake → viene utilizzato quando la disconnessione non è contemporanea tra i due host in comunicazione

↓
dopo che è stata stabilita una connessione bidirezionale ma piuttosto come l'interazione di due connessioni monodirezionali

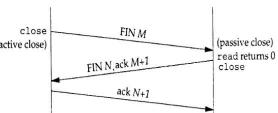
↓
ognuna delle parti deve terminare la propria connessione

↓
possono esistere anche connessioni chiuse a metà, in cui solo uno dei due host ha chiuso la connessione poiché non ha più nulla da trasmettere.

↓
ma può (e deve) ancora ricevere dati dall'altro host

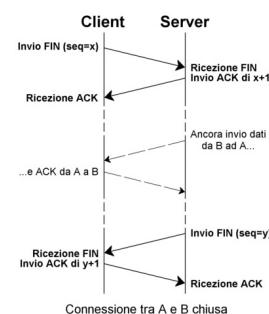
↓
Oppure con l'handshake a tre vie, omologo a quello utilizzato per l'inizio di connessione ma anziché SYN c'è FIN

↓
nel caso in cui la chiusura di connessione avvenga contemporaneamente



↓
uno dei due host invia la richiesta di FIN, e attende l'ACK di risposta; l'altro terminale farà poi altrettanto

↓
si generano in totale 4 segmenti



↓
è possibile anche una modalità più aggressiva di chiusura della connessione

↓
flag RESET

↓
interrompe connessione in entrambe le direzioni

AFFIDABILITÀ DELLA COMUNICAZIONE

↓

consegna ordinata ed eliminazione di duplicati → SEQUENCE NUMBER

↓

riconoscere dei pacchetti e ritrasmissione → ACKNOWLEDGMENT NUMBER

↓

CONTROLLO DI FLUSSO

↓

far in modo che il flusso di dati in trasmissione non superi le capacità di ricezione del ricevente

↓

viene attuato attraverso la specifica da parte del destinatario di un opportuno campo noto come rwnd (receiver window)

↓

protocollo a finestre scorrevoli → spazio disponibile nel buffer (in byte) indicato nel campo window size

↓

se $windowsize = 0$ il mittente non può spedire dati

↳ retransmission time-out

↳ variabile dinamica

↓
specificà il numero massimo di segmenti ricevibili dal destinatario

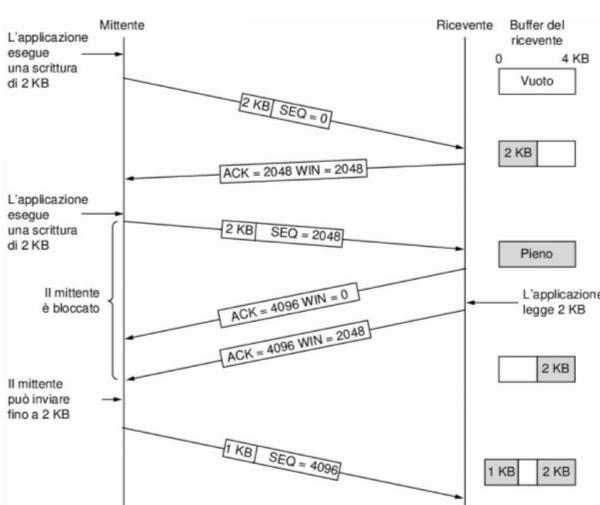
↳ a meno che non siano dati urgenti;

↓ oppure

segmento sonda (window probe)

↓
mandato x chiedere all'altra stazione → apposito timer

↓
sia mittente che destinatario usano timer, così il destinatario può confermare anche solo l'ultimo byte ricevuto senza far scadere il timer



CONTROLLO DI CONGESTIONE

↓

attuato congiuntamente al controllo di flusso

↓

si basa sul fatto che le due entità messe in comunicazione dal TCP si scambiano pacchetti contenenti dati e pacchetti di riscontro (ACK) che viaggiano nel verso opposto rispetto ai dati

contr. flusso limita quantità di dati

impone a ciascun mittente un limite alla velocità di invio dei pacchetti in funzione della congestione di rete percepita

→ in assenza di perdite di pacchetti, il tempo che intercorre tra l'invio di un segmento e la ricezione dell'ACK definisce RTT

↓
in presenza di congestione nella rete, code nei router + lunghe, + ritardo per pacchetti e ACK, aumenta l'RTT

↓ \hookrightarrow ACK clock

talè limite rappresentato dalla variabile cwnd (congestion window)

← il controllo di congestione rileva l'approssimarsi di questa situazione nelle entità terminali delle connessioni di TCP

scadenza di un time-out per mancata ricezione di un ACK
ricezione di dup ACK

eventi che sono sintomi di perdita di pacchetti

come rileva la congestione?

SLOW START

↓

la finestra di congestione viene impostata di default a 1 → velocità di invio iniziale $\frac{MSS}{RTT}$ → Maximum Segment Size → massima dimensione del payload (corpo di dati)

↓
TCP invia il primo segmento dati e aspetta ACK
cwnd aumenta di una quantità pari ad MSS per ciascun segmento che, dopo esser stato trasmesso, riceve ACK

↓
quindi ad ogni RTT la cwnd raddoppia di dimensioni (e raddoppia velocità trasmissiva)
tempistica dell'ack clock: se il percorso nella rete è lento, gli ACK arriveranno lentamente (dopo ritardo di RTT), se è veloce arrivano velocemente (sempre dopo RTT)

poiché lo slow start causa una crescita esponenziale, prima o poi spedirà in rete troppi pacchetti e troppo velocemente

↓
code nella rete → pipeline piena → perdita di pacchetti → TCP non riceverà ACK
per tenere sotto controllo lo slow start → slow start threshold
inizialmente valore fissato = rwnd → slow start cresce fino a

↓ quando supera la ssthresh

INCREMENTO ADDITIVO

↓
cwnd viene aumentata di un segmento per ogni RTT → $MSS \cdot (\frac{MSS}{cwnd})$
ogni volta che riceve ACK

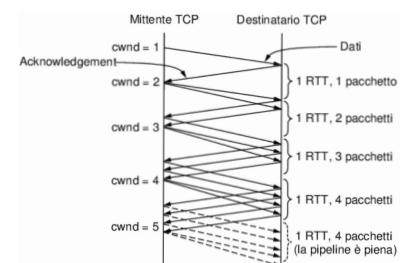
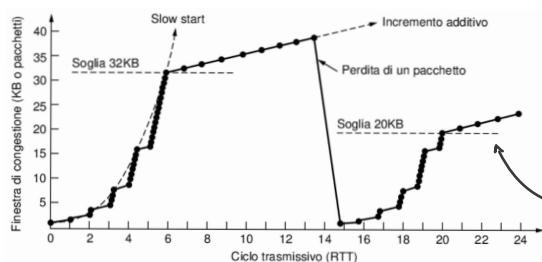
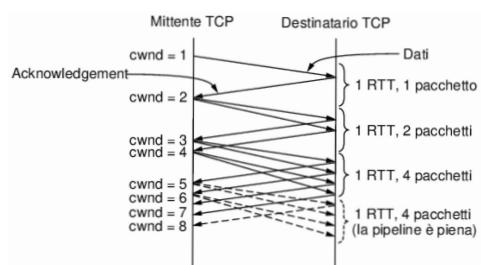
↓
crescita più lenta ma più lineare → rispetto allo slow start

↓
se si perde un pacchetto → no ACK
cwnd piena → scatterà time-out

↓
se arrivano 3 dup ACK → il pacchetto perso sarà il prossimo in sequenza

FAST RETRANSMISSION

↓
 $ssthresh = \frac{cwnd}{2}$ → per far ripartire lo slow start
 $cwnd = MSS$

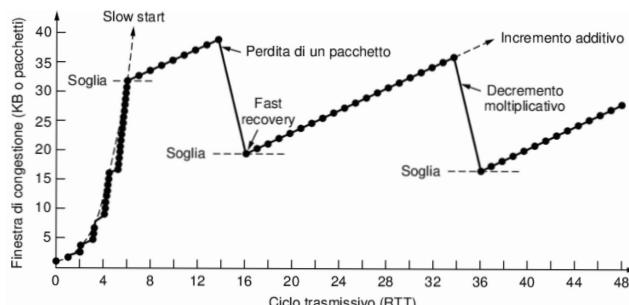


TCP Reno \rightarrow TCP Tahoe + FAST RECOVERY \rightarrow modalità temporanea \rightarrow dopo l'arrivo di 3 dupACK \rightarrow ssthresh_{nuova} = $\frac{cwnd}{2}$

\downarrow
evita lo slow start

\downarrow
il cwnd non viene chiuso
eccessivamente

\downarrow
cerca di mantenere ack clock in funzione
quindi ritrasmissione veloce



\downarrow
 $cwnd = ssthresh + 3 \rightarrow$ fast recovery

\downarrow
finché numero di pacchetti non scende al di sotto
della nuova soglia contando i dupACK (inclusi i 3
che hanno fatto scattare ritrasmissione veloce)

\downarrow impiega RTT/2

per ogni successivo dupACK la cwnd aumenta
di 1 segmento permettendo la trasmissione
di nuovi dati

\downarrow trascorso RTT

\downarrow quando la sorgente riceve l'ACK di conferma
del segmento ritrasmesso \rightarrow cwnd = ssthresh

\downarrow si procede la trasmissione con il fast retransmit

\downarrow

se il segmento ritrasmesso viene perso, si attende
scadere RTO, cwnd = 1 \rightarrow Slow start

riassuntino \heartsuit

\downarrow

all'inizio della trasmissione \rightarrow cwnd = 1 (MSS)

\downarrow

la cwnd evolve secondo $ssthresh = rwnd$ oppure $ssthresh = \frac{cwnd}{2}$

(dipende dalle implementazioni)

lo slow start fino al

raggiungimento di ssthresh \rightarrow per ogni RTT, cwnd cresce in modo
esponenziale, raddoppiando

\downarrow

dopo cwnd cresce secondo

l'incremento additivo \rightarrow lineare, per ogni RTT aumenta di 1

\downarrow

la finestra cresce fino al raggiungimento di rwnd \rightarrow in caso di errore o perdita di segmenti \rightarrow si attende timeout RTO

trasmissione si interrompe

$ssthresh = \frac{cwnd}{2} = 1$

\downarrow slow start

$ssthresh_{nuova} = \frac{cwnd}{2}$ e
senza attendere RTO ritrasm.
(fast retransmit)

se alla sorgente
arrivano 3 dupACK

errori di trasmiss.

congestione

generalmente influisce
su un solo segmento

perdita di + segmenti

\downarrow
è una reazione "eccessiva" da parte della sorgente chiudere cwnd e
attendere il timeout

\downarrow

algoritmi implementati sempre in coppia progettati per gestire singole perdite

- \rightarrow fast retransmit \rightarrow il segmento perso viene subito ritrasmesso
- \rightarrow fast recovery \rightarrow la cwnd non viene chiusa eccessivamente

(TCP)

\downarrow
orientato alla connessione

\downarrow
affidabile \rightarrow ACK, RTO, ritrasmissione,
consegna ordinata (sequence
number)

flusso di byte

\downarrow
trasferimento di file

(UDP)

\downarrow
senza connessione \rightarrow invia solo i datagrammi richiesti dal livello applicativo

\downarrow
non affidabile \rightarrow non dà garanzie sull'arrivo dei segmenti e sul loro ordine di arrivo

\downarrow
oggetto della comunicazione: singoli datagrammi

\downarrow
Forte vincolo sulla velocità e risorse di rete \rightarrow live streaming o multiplayer

LAN → broadcast → accesso condiviso

