Livello trasporto in Internet

Livello trasporto in Internet

- Due protocolli di trasporto alternativi: TCP e UDP
- Modelli di servizio diversi
  - TCP orientato alla connessione, affidabile, controllo di flusso e congestione, stateful
  - UDP non connesso, inaffidabile, stateless
- Caratteristiche comuni:
  - multiplazione e demultiplazione mediante le porte
  - rilevazione errori su header e dati (opzionale in UDP)

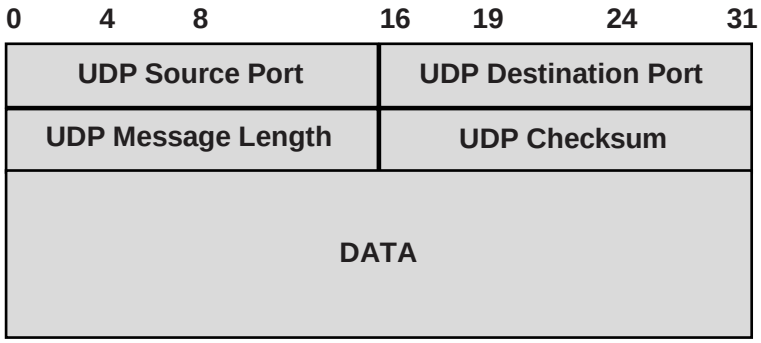
Mux/demux: le porte

- Il destinatario finale dei dati non è un host ma un processo in esecuzione sull'host
- L'interfaccia tra processi applicativi e strato trasporto è rappresentata da una porta
  - numero intero su 16 bit
  - associazione tra porte e processi
  - processi server "pubblici" sono associati a porta "ben nota", inferiore a 1024 (es: 80 per WWW, 25 per email)
  - processi client usano porta assegnata dinamicamente dal sistema operativo, superiore a 1024

UDP: User Datagram Protocol

- Protocollo di trasporto di tipo non connesso
- Non fornisce garanzie di consegna
- Due funzionalità:
  - multiplazione delle informazioni tra le varie applicazioni tramite il concetto di porta
  - checksum (opzionale) per verificare l'integrità dei dati
- Un'applicazione che usa UDP deve risolvere problemi di affidabilità, perdita di pacchetti, duplicazione, controllo di sequenza, controllo di flusso, controllo di congestione
- Standardizzato in RFC 768

UDP: formato pacchetto



UDP: applicabilità

- Utile quando:
  - Si opera su rete locale (affidabilità)
  - Applicazione mette tutti i dati in un singolo pacchetto (non apro connessione)
  - Non è importante che tutti i pacchetti arrivino a destinazione
  - Necessità di protocollo veloce
    - Evita overhead apertura connessione
    - Meccanismi di ritrasmissione per affidabilità non utilizzabili per vincoli temporali
  - Applicazione gestisce meccanismi di ritrasmissione

Protocollo TCP

- TCP (Transmission Control Protocol) è un protocollo di livello trasporto orientato alla connessione
- Affidabile:
  - garantisce la consegna corretta ed in sequenza al ricevitore dei pacchetti trasmessi dal trasmettitore
- Utilizzato da applicativi che richiedono la trasmissione affidabile dell'informazione
  - telnet (terminale remoto)
  - ftp (file transfer protocol)
  - smtp (simple mail transfer protocol)
  - http (hypertext transfer protocol)

TCP: riferimenti bibliografici

- Richard Stevens: TCP Illustrated, vol.1
- RFC 793 (1981)
  - Transmission Control Protocol
- RFC 1122/1123: (1989)
  - Requirements for Internet Hosts
- RFC 1323: (1992)
  - TCP Extensions for High Performance (PRP STD)

TCP: riferimenti bibliografici

- RFC 2018: (1996)
  - TCP Selective Acknowledgment Options (PRP STD)
- RFC 2581:
  - TCP Congestion Control (PRP STD)
- RFC 2582:
  - The NewReno Modification to TCP's Fast Recovery Algorithm
- RFC 2883:
  - An Extension to the Selective Acknowledgement (SACK) Option for TCP
- RFC 2988:
  - Computing TCP's Retransmission Timer (PRP STD)

TCP

- Fornisce porte per (de)multiplazione
- Una entità TCP di un host, quando deve comunicare con un'entità TCP di un altro host, crea una connessione fornendo un servizio simile ad un circuito virtuale
  - bidirezionale (full duplex)
  - con controllo di errore e di sequenza
- Richiede maggiore capacità di elaborazione rispetto a UDP e di mantere informazioni di stato negli host per ogni connessione

TCP

- TCP segmenta e riassembla i dati secondo le sue necessità:
  - tratta stream di dati (byte) *non strutturati* dai livelli superiori
  - non garantisce nessuna relazione tra il numero di read e quello di write (buffer tra TCP e livello applicazione)
- Protocollo a finestra per ottenere affidabilità
- Esegue un controllo di flusso e di congestione regolando la velocità del trasmettitore variando finestra di trasmissione

Identificazione connessioni

- Una connessione TCP tra due processi è definita dai suoi endpoints (punti terminali), univocamente identificati da un socket:
  - Indirizzi IP host sorgente e host destinazione
  - Numeri di porta TCP host sorgente e host destinazione
- Nota: TCP o UDP usano porte indipendenti
- Esempio: connessione TCP tra porta 15320 host 130.192.24.5 e porta 80 host 193.45.3.10


Trasmettitore TCP

Suddivide i dati dell'applicazione in segmenti

- Protocollo a finestra, con ritrasmissione stile GBN (funzionamento complessivo ibrido GBN-SR)
- Stima RTT
- Attiva timer quando invia i segmenti:
  - segmenti non confermati allo scadere del timer (timeout - RTO) provocano ritrasmissioni
- Calcola e trasmette checksum obbligatorio su header e dati
- Regola velocità con dimensione finestra
  - controllo di flusso e congestione


Ricevitore TCP

- Riordina segmenti fuori sequenza e scarta segmenti errati
  - consegna stream ordinato e corretto a processo applicativo
- Invia ACK cumulativi
- Annuncia spazio libero nel buffer di ricezione per controllare velocità trasmettitore (controllo di flusso)

Ricevitore TCP

- Segmento corretto ed in sequenza
  - Memorizza (ed eventualmente consegna al livello superiore) ed invia ACK cumulativo
- Segmento duplicato
  - scarta ed invia ACK relativo all'ultimo segmento ricevuto in sequenza
- Segmento con checksum errato
  - Scarta senza inviare ACK
- Segmento fuori sequenza
  - Memorizza (non obbligatorio, ma standard de facto) ed invia ACK relativo ultimo segmento ricevuto (ACK duplicato)

Delayed ACK

- Motivazioni
  - Il ricevitore riduce la quantità di ack da inviare (riduzione di traffico di controllo)
  - Posso attendere che l'applicazione crei dati in risposta ai dati ricevuti e sfruttare il piggybacking per inviare ack
  - Il ricevitore può svuotare il buffer di ricezione e dichiarare finestre disponibili maggiori
- Svantaggi
  - Altero il RTT (Round Trip Time) della connessione
  - Modifico crescita finestra (vedi dopo)

Delayed ack: algoritmo

- ACK inviati
  - o ogni 2 segmenti ricevuti
    - crescita della finestra a velocità “dimezzata”
  - o dopo 200ms dalla ricezione di un singolo segmento
- Invio immediato dell’ACK si ha solo per segmenti fuori sequenza: conferma l’ultimo segmento ricevuto in sequenza.
  - origina ACK duplicati

TCP: generazione ACK

[RFC 1122, RFC 2581]

Eventi	Azioni ricevitore TCP
arrivo segmento in ordine, senza vuoti inviato ACK correttamente per tutti segmenti precedenti	delayed ACK. Attendi fino a 200ms max per segmento successivo, altrimenti invia ACK
arrivo segmento in ordine, senza vuoti delayed ACK in attesa	invio immediato di ACK
arrivo segmento fuori sequenza con numero maggiore di quello atteso vuoto rilevato	invia ACK duplicato, indicando come numero di sequenza il prossimo byte che si attende di ricevere
arrivo di segmento che riempie vuoti prazialmente o completamente	ACK immediato se il segmento copre parte iniziale della finestra

Controlli flusso e congestione

- Generico protocollo a finestra: la velocità di trasmissione in assenza di errori:

$$\frac{\text{Finestra di trasmissione}}{\text{Round trip time}}$$

  - Connessioni “corte” ottengono banda maggiore
- Per regolare velocità di trasmissione posso agire su
  - round trip time (ritardando invio di ack)
    - genero ritrasmissioni
  - dimensione finestra

Controlli flusso e congestione

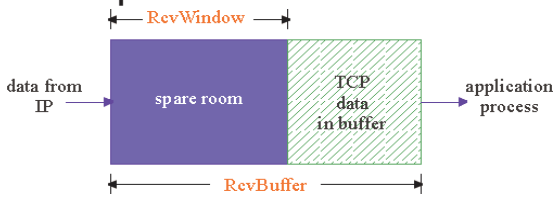
- Se cresce finestra oltre valore per cui bit rate in trasmissione supera capacità del collo di bottiglia
  - si memorizzano dati nei buffer lungo il percorso, e cresce round trip time
- TCP: Velocità di trasmissione di un trasmettitore è regolata da:
  - controllo di flusso: evita che un host veloce saturi un ricevitore lento
  - controllo di congestione: evita che un host ‘aggressivo’ saturi la rete trasmettendo sempre alla velocità massima consentita dal ricevitore

Controlli flusso e congestione

- TCP impiega un controllo end-to-end basato su controllo della dimensione della finestra del trasmettitore:
  - controllo di flusso: il ricevitore impone la dimensione massima della finestra del trasmettitore, indicando negli ACK la finestra di ricezione disponibile
  - controllo di congestione: il trasmettitore si autoimpone una dimensione massima della finestra in funzione delle perdite riscontrate per mancato arrivo di ACK

Controllo di flusso in TCP

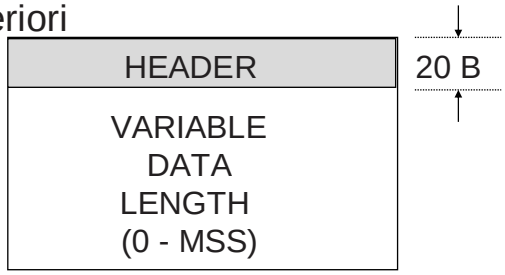
- RX informa esplicitamente TX della memoria disponibile (variabile nel tempo)
  - campo rwnd nell'intestazione segmento TCP
- TX: finestra (dati trasmessi senza avere ricevuto ACK) non eccede mai ultimo valore di finestra disponibile dichiarato da RX



Buffer di ricezione

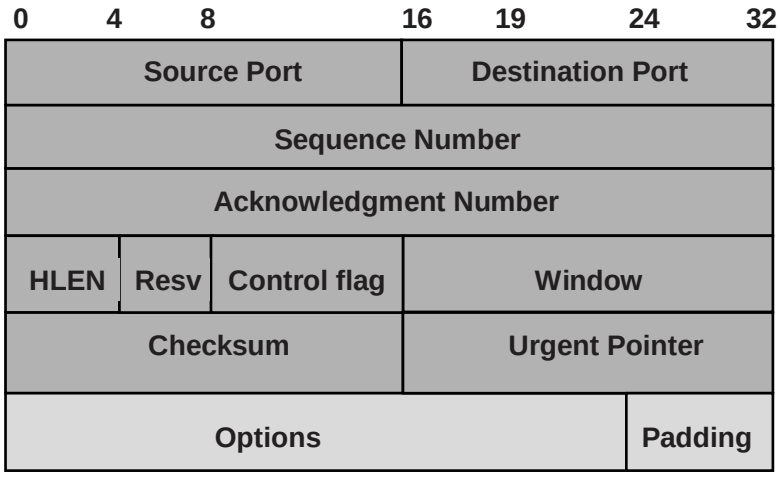
La PDU TCP

- La PDU di TCP è detta *segmento*
- La dimensione dei segmenti può variare dal solo header (ACK, 20 byte) fino ad un valore massimo MSS concordato con il ricevitore e dipendente dalla MTU IP
- La dimensione del singolo segmento dipende dallo stream dei livelli superiori



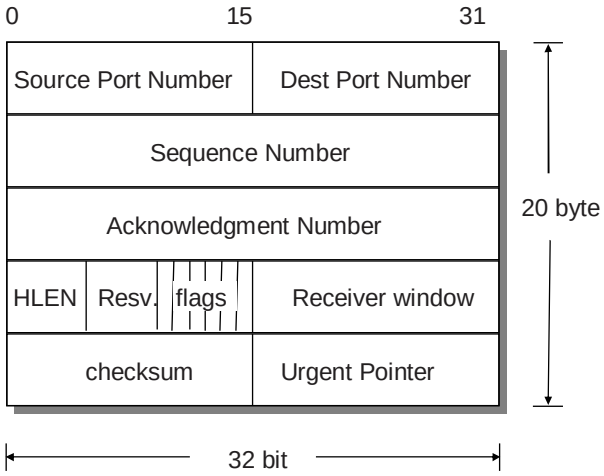
AA 2004-2005 Reti e Sistemi Telematici 23

TCP: intestazione

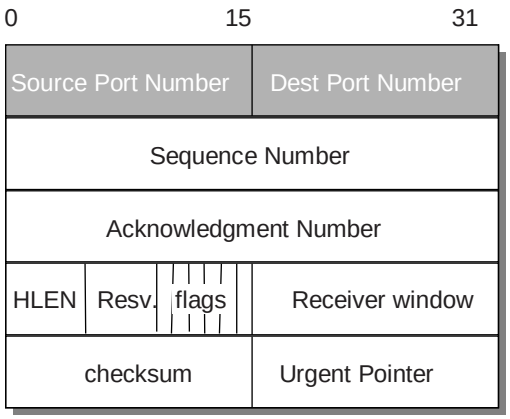


AA 2004-2005 Reti e Sistemi Telematici 24

TCP intestazione (senza opzioni)

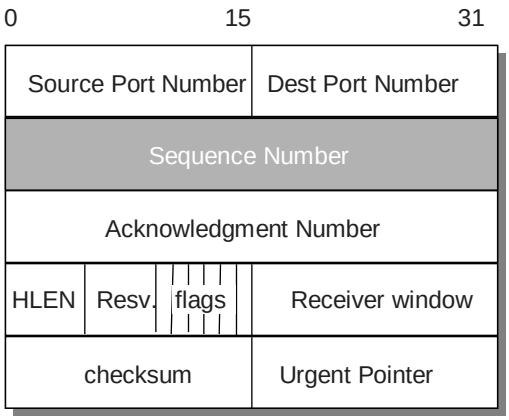


TCP header



Identificano l'applicazione che sta inviando e ricevendo dati. Combinati con i rispettivi indirizzi IP, identificano in modo univoco una connessione

TCP header



- Identifica, nello stream di dati, la posizione del primo byte del payload del segmento
- Ogni direzione della connessione procede con numeri di sequenza diversi e indipendenti

AA 2004-2005 Reti e Sistemi Telematici 27

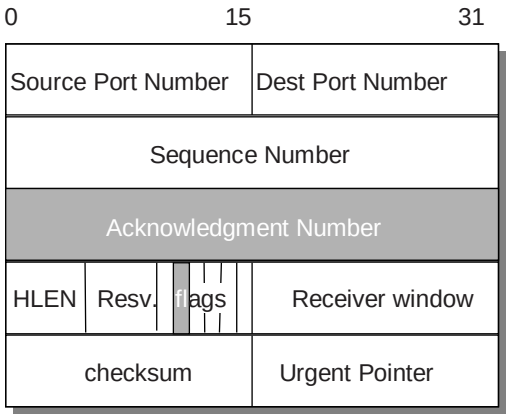
Numerazione segmenti

- Numero di sequenza su 32 bit
- In funzione della banda, ho diversi tempi di Wrap Around (torno a numero di sequenza iniziale)

Capacità		Tempo prima di wrap around	
T1	(1.5Mbps)	6.4	ore
Ethernet	(10Mbps)	57	minuti
T3	(45Mbps)	13	minuti
FDDI	(100Mbps)	6	minuti
STS-3	(155Mbps)	4	minuti
STS-12	(622Mbps)	55	secondi
STS-48	(2.5Gbps)	14	secondi

AA 2004-2005 Reti e Sistemi Telematici 28

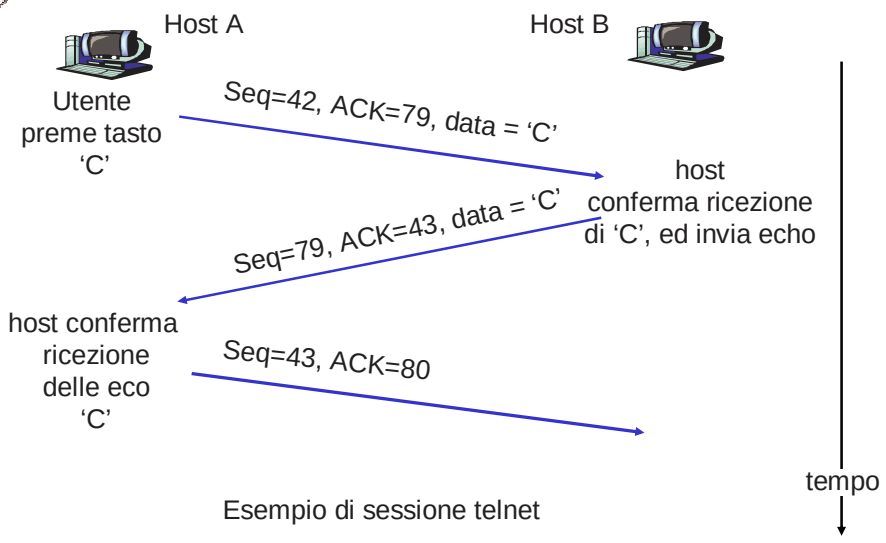
Intestazione TCP



Numero di sequenza più 1 dell'ultimo byte di dati ricevuto correttamente

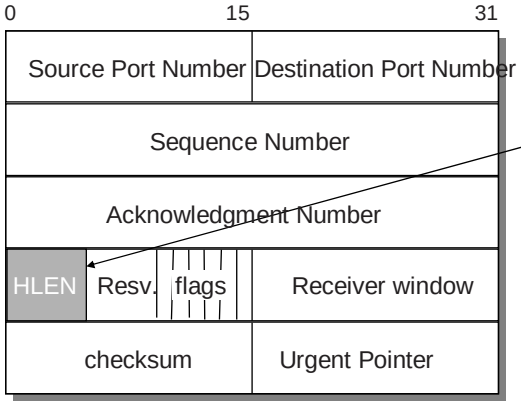
Valido solo con ACK flag settato

Numeri di sequenza e di ack



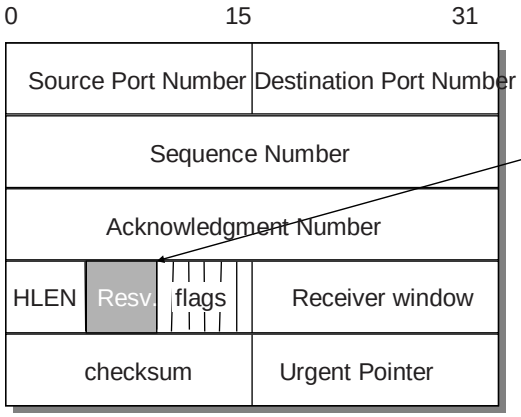
Esempio di sessione telnet

Intestazione TCP



Lunghezza dell'header in parole di 32 bit

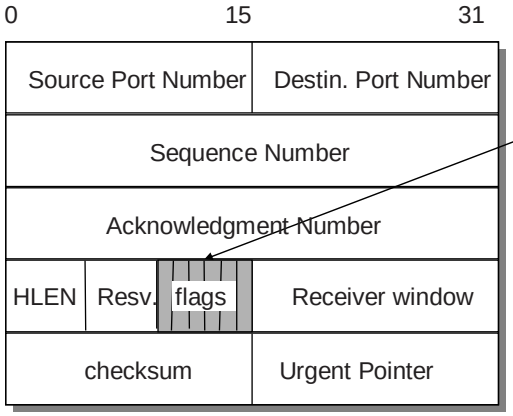
Intestazione TCP



Riservato per usi futuri (ECN)

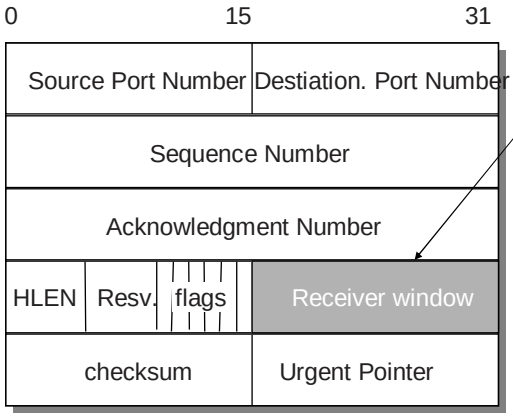
Intestazione TCP

- Gestione connessione



- Sei bit di flag, uno o più possono essere settati insieme:
  - URG: urgent pointer valido
  - ACK: numero di ack valido
  - PSH: forza passaggio dati applicazione
  - RST: reset connessione
  - SYN: synchronize seq. No. Apertura connessione
  - FIN: chiusura connessione

IntestazioneTCP



Numero di byte, a partire da quello nel campo di ACK, che il ricevitore è disposto ad accettare per controllo di flusso

Valore massimo rwnd 65535 byte, a meno che sia usata la *window scaling option* per finestre grandi

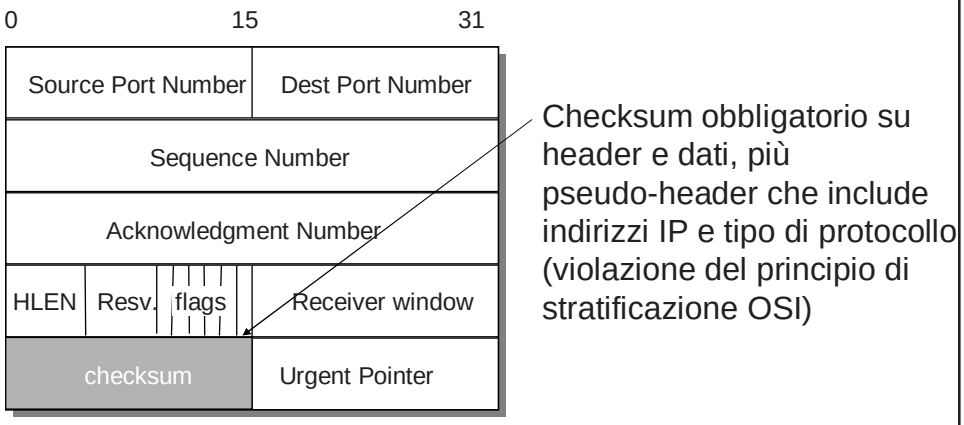
Finestra necessaria per ottenere velocità massima

- Massima quantità di dati in transito per RTT:
  - 16-bit rwnd = 64kB max
- Prodotto banda x ritardo per RTT=100ms

Banda		banda x ritardo
T1	(1.5Mbps)	18KB
Ethernet	(10Mbps)	122KB
T3	(45Mbps)	549KB
FDDI	(100Mbps)	1.2MB
STS-3	(155Mbps)	1.8MB
STS-12	(622Mbps)	7.4MB
STS-48	(2.5Gbps)	29.6MB

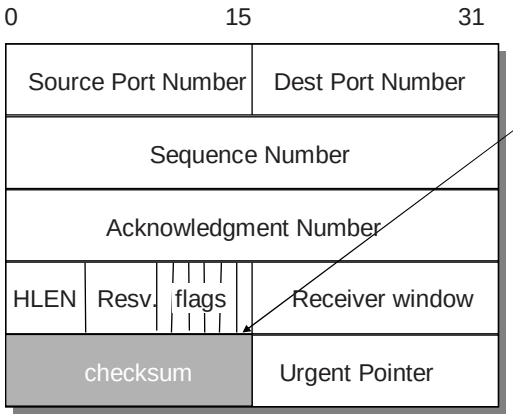
- Limite superabile con window scale option

Intestazione TCP



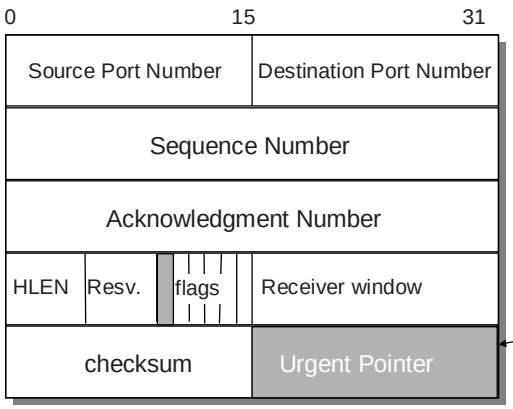
Checksum obbligatorio su header e dati, più pseudo-header che include indirizzi IP e tipo di protocollo (violazione del principio di stratificazione OSI)

Intestazione TCP



- Algoritmo di checksum
  - allineamento di header, dati e pseudo-header su 16 bit
  - somma in complemento a 1 di ogni riga
  - si ottiene numero a 32 bit, che si divide in due parti di 16 bit
  - somma in complemento a 1 delle due parti, incluso il riporto
  - inserisco nell'header i 16 bit risultanti

Intestazione TCP



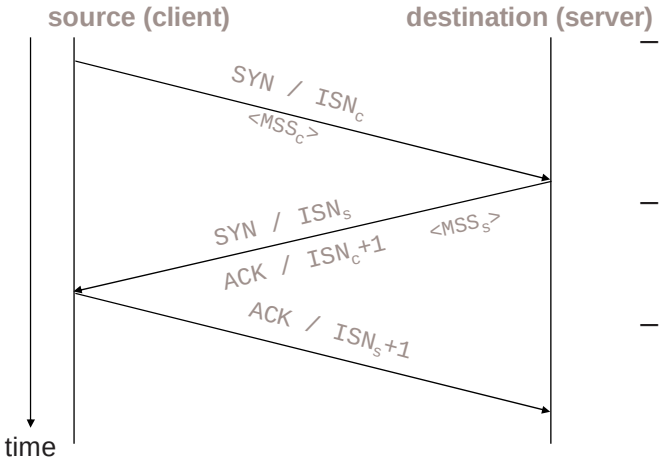
Puntatore a "dati urgenti" nel campo dati (es. ctrl-C in una sessione telnet).
Offset rispetto al num. di seq.

Valido solo se flag URG è settato

TCP options (MSS)

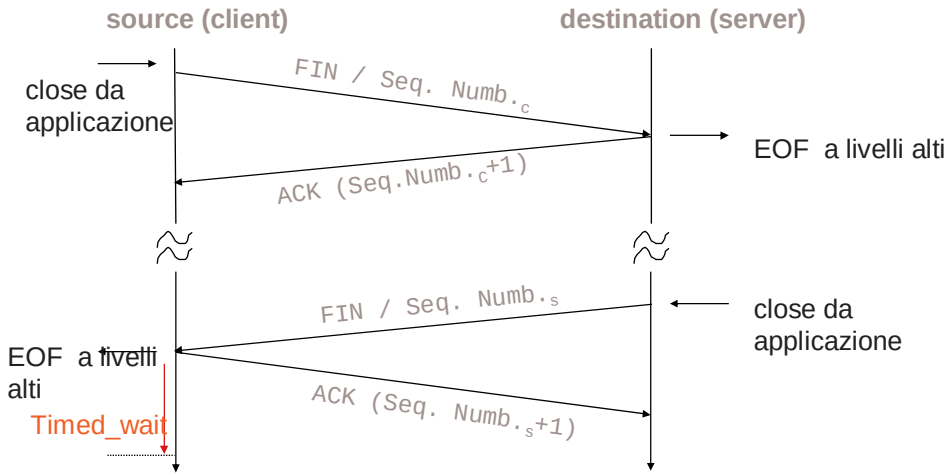
- Estensione dell'header (precede i dati)
- Opzione più usata è MSS (Maximum Segment Size), inviato nel segmento iniziale di una connessione
- Non è negoziato, ogni lato annuncia MSS che si aspetta di ricevere
- Se non presente, si usa il default a 536 byte
- Al massimo: MSS = MTU_{IP} - 20 $byte_{IP}$ -20 $byte_{TCP}$
  - Se non si usano SACK o altre opzioni

Apertura di connessione (three-way handshake)



- three-way handshake
- il client esegue una open attiva, mentre il server esegue una open passiva
- ISN (initial sequence number) è generato casualmente
- un SYN consuma un numero di sequenza

Chiusura di connessione (TCP half-close)



TCP: controllo di congestione

- Inizialmente (<1988) era previsto che TCP controllasse la velocità di emissione utilizzando solo la finestra imposta dal RX
- Questa soluzione funziona se i due host sono sulla stessa LAN, ma se ho router intermedi e linee “lente” posso avere congestione
- Il risultato è una pesante riduzione del throughput delle connessioni TCP, costrette a frequenti ritrasmissioni

Elementi del controllo di congestione

- Oltre alla finestra imposta dal RX (rwnd), il TX si "autoimpone" una "congestion window" (cwnd), regolata da 4 algoritmi
- Nota: la descrizione degli algoritmi assume che ogni pacchetto sia di dimensione pari a 1 MSS
- Il TX può inviare fino a n segmenti TCP con

  n = min (rwnd, cwnd)
- Esiste una soglia (ssthresh) sulla dimensione della cwnd che ne determina la legge di variazione

Algoritmi di controllo di congestione

- 4 algoritmi di controllo, introdotti in fasi successive:
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
  - Fast Recovery

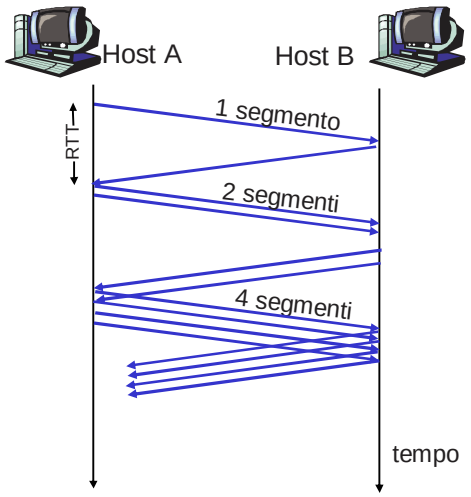
Slow Start e Congestion Avoidance

- La scelta dell'algoritmo dipende dalle variabili ssthresh e cwnd:
  - Slow start se ssthresh > cwnd
  - Congestion Avoidance se ssthresh < cwnd
  - se ssthresh = cwnd, si esegue, indifferentemente, l'uno o l'altro algoritmo

Algoritmo di Slow Start

- Principi
  - La velocità di trasmissione di nuovi segmenti si adatta al rate di ricezione degli ACK
  - Il TX "prova" la capacità della rete fino a perdere
- All'inizio della connessione, si ha cwnd = 1 segmento di dimensione massima (cwnd=MSS)
- Ad ogni ACK ricevuto, cwnd = cwnd + 1
- La crescita risultante è esponenziale

Algoritmo di Slow Start



Algoritmo di Congestion Avoidance

- Principi
  - interviene quando cwnd>ssthresh
  - rallenta la crescita della cwnd
- Ad ogni ack ricevuto in sequenza
  - cwnd = cwnd + 1/ cwnd oppure
  - cwnd = cwnd + MSS*MSS/ cwnd (in byte)
- La crescita risultante è lineare
  - cwnd cresce di 1 MSS per ogni RTT

Se si perde un segmento…

- …vuol dire che il throughput del TX ha superato la banda disponibile → congestione ?
- Principio controllo di congestione:
  - Reset della finestra (cwnd=1)
  - Recupero "veloce" della velocità perduta agendo sulla finestra
- Il TX ritrasmette il segmento mancante se non riceve l'ACK relativo entro un timeout (RTO), poi riparte in slow start

Riepilogo

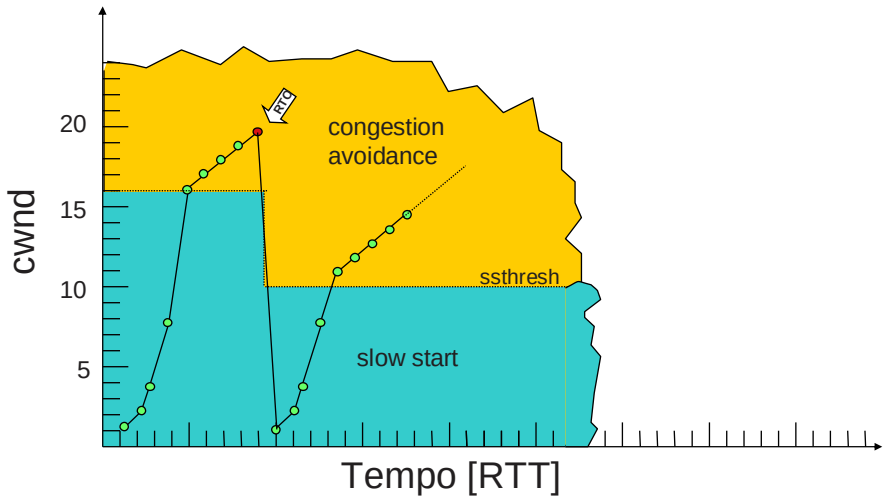
SLOW START

1) `cwnd` = 1 MSS
   `ssthresh` = infinito
2) `cwnd` = `cwnd` + 1 ad ogni `ack` finché
   `cwnd` > `ssthresh` (goto 3)
   se ho un RTO:
   `ssthresh` = min(`cwnd`,`rwnd`)/2
   `cwnd` = 1
   goto 2)

CONGESTION AVOIDANCE

3) `cwnd` = `cwnd` + 1/ `cwnd` ad ogni `ack`
   se ho un RTO:
   `ssthresh` = min(`cwnd`,`rwnd`)/2
   `cwnd` = 1
   goto 2)

Riepilogo



Fast Retransmit e Fast Recovery

- Ulteriore modifica all'algoritmo di Congestion Avoidance proposta nel 1990 (RFC 2001, Stevens)
- Permette la ritrasmissione immediata di segmenti singoli andati perduti (Fast Retransmit)
- … e per evitare di ritornare nella fase di Slow Start quando ad essere perso è un solo segmento (Fast Recovery)

Fast Retransmit

- Osservo gli ack duplicati:
  - se sono uno o due, può essere solo uno scambio di segmenti
  - se sono tre o più è una forte indicazione di segmento perso (ma alcuni segmenti arrivano!)
- Se vengono ricevuti tre ack duplicati, ritrasmetto il segmento “mancante” senza aspettare la scadenza del timeout (Fast Retransmit).

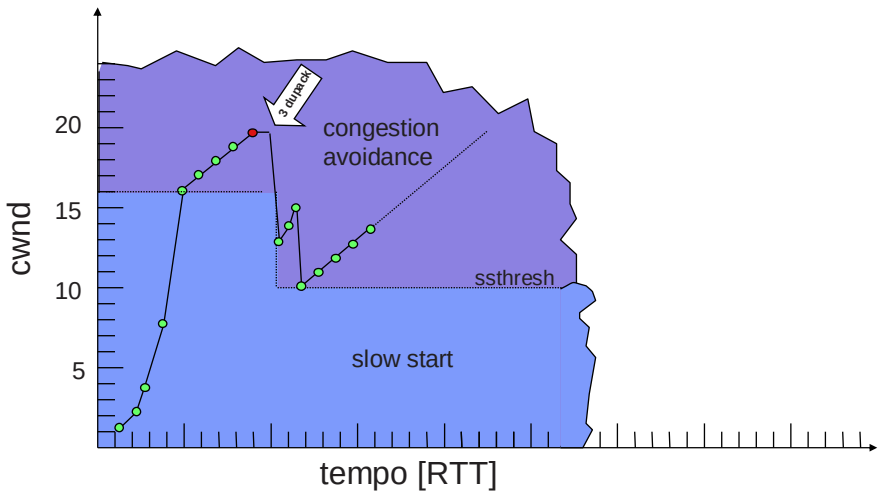
Fast Recovery

- Eseguo congestion avoidance e non slow start dopo fast retransmit
- Quando ricevo il 3° ack duplicato consecutivo:
  - ssthresh = min(cwnd,rwnd)/2
  - ritrasmetto il segmento mancante
  - cwnd=ssthresh+3
- Ad ogni ack duplicato successivo:
  - cwnd=cwnd+1
  - abilita la trasmissione anche durante il F.R.

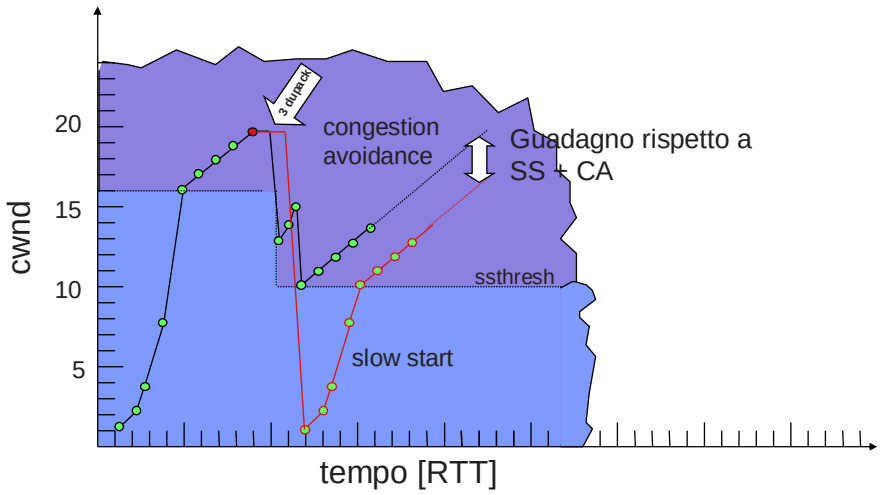
Fast Recovery

- Quando arriva un ack che conferma (anche in modo implicito) il segmento ritrasmesso:
  - cwnd=ssthresh
  - cwnd=cwnd+1/cwnd per ogni ack in sequenza (crescita “alla” Congestion Avoidance)

Riepilogo



Riepilogo

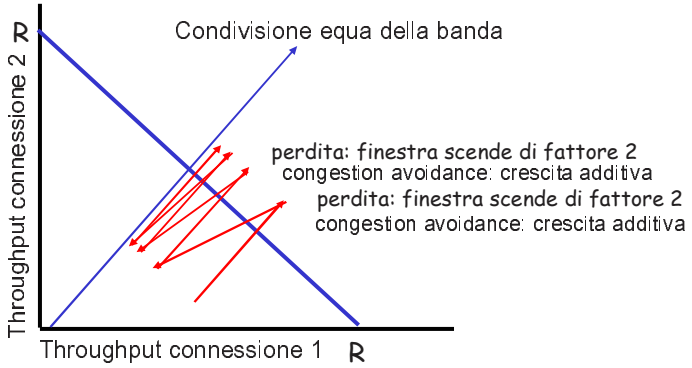


Equità di TCP

- L'algoritmo di controllo della congestione TCP è di tipo AIMD (additive increase, multiplicative decrease)
  - la finestra cresce di 1 MSS per RTT
  - la finestra decresce di un fattore 2 a fronte di perdita di segmenti
- Fairness: se N connessioni TCP condividono un canale collo di bottiglia, ciascuna dovrebbe ottenere 1/N della capacità del canale (a pari RTT)

Perché AIMD è equo

- Due connessioni TCP che competono
  - AI, throughput pendenza unitaria
  - MD decrescita proporzionale



Determinazione del timeout

- Il valore del timeout è essenziale per un buon funzionamento di TCP
- Non può essere inferiore a 200ms (delayed ack e granularità del clock del trasmettitore)
- È funzione del round-trip time della connessione, che varia al variare del traffico e della congestione di rete
- Occorre quindi una stima del round-trip time per determinare il timeout da impostare

Determinazione del timeout

- Per ogni segmento calcolo la differenza di tempo “M” tra il suo invio e la ricezione di un ack riferito al pacchetto
  - RTT “istantaneo”
- La stima del round-trip time (RTT) viene mediata da un coefficiente α:

  $RTT=\alpha*RTT+(1-\alpha)*M \quad (\alpha=0.1)$
- Il timeout (RTO) viene calcolato come:

  $RTO= \beta *RTT \quad (\beta >1)$

Note su stima RTT

- L’algoritmo di stima è valido, ma sempre limitato dalla granularità del timer (10ms su moderni Unix, 200-500ms su sistemi più vecchi)
  - Il RTT può essere comparabile (RTT=100-200ms per connessioni internazionali)
- L’accuratezza della stima del RTT è fondamentale per il controllo di congestione (evita ritrasmissioni inutili)

Problema sul timeout

- Valore iniziale?
- Mancando stima del RTT, scelgo valore grande per essere conservativo
  - timeout iniziale pari ad 3s
- Connessioni TCP soffrono molto la perdita del primo segmento