RETI DI CALCOLATORI

Autunno 2018

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 10:
LIVELLO TRASPORTO:
UDP
TCP: CONTROLLO DEGLI ERRORI
08/11/2018

parte di queste slides sono ricavati da slides pubblicate in corsi universitari tenuti da colleghi italiani e stranieri

MATERIALE DIDATTICO

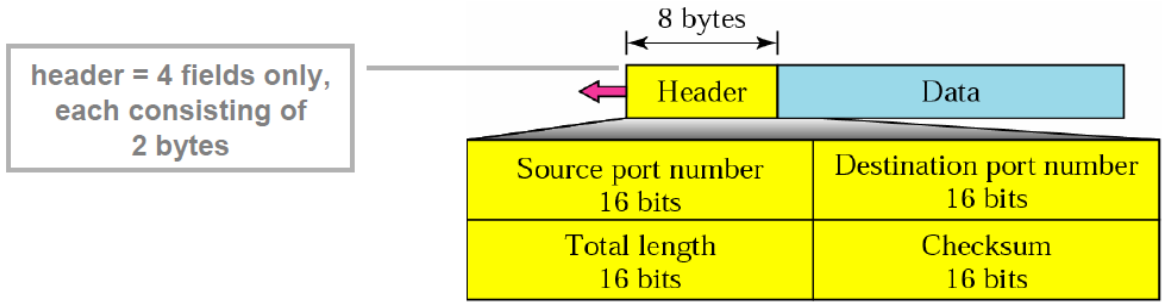
- Forouzan
  - Paragrafo 3.3
  - Pargrafo 3.4.1, 3.4.2, 3.4.3, 3.4.6, 3.4.8

USER DATAGAM PROTOCOL

- Connectionless, non affidabile
  - nessun meccanismo di controllo del flusso
  - semplice controllo degli errori
    - servizio "best effort"
    - aggiunge al servizio IP solo il meccanismo di multiplexing/demultiplexing
- ...ma allora...perchè usare UDP?
  - nessun delay per la definizione e per la gestione di una connessione
    - TCP richiede 3-way handshake
  - non necessario gestire stato della connessione
    - TCP mantiene lo stato della connessione negli end-systems (receive e send buffers, numeri di sequenza ed acknowledgement)
  - Header di piccola dimensione (solo 8 bytes...)
    - TCP richiede un header di 20 bytes
  - nessun controllo del flusso/della congestione
    - UDP invia i pacchetti il più rapidamente possibile
  - message oriented invece che di byte oriented

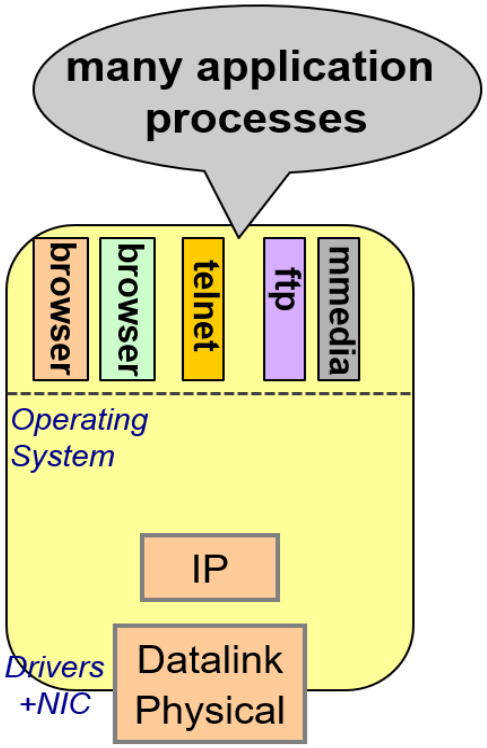
USER DATAGAM PROTOCOL (UDP)

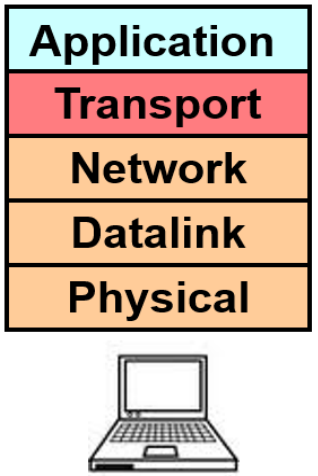
- UDP Datagram: 8 byte header + application data [definita in RFC 768]



- Total Length
  - definisce la lunghezza totale del datagram UDP (header + data)
  - 16 bits: dimensione teorica: massimo 65,535 bytes
  - tuttavia molti SO limitano la dimensione del pacchetto UDP a 8192 bytes
- Checksum
  - usata per individuare errori all'interno dell'intero datagram UD (header + data)
  - opzionale: se non è calcolata, è riempita con degli 0
  - se viene individuato un errore, il pacchetto viene semplicemente scartato

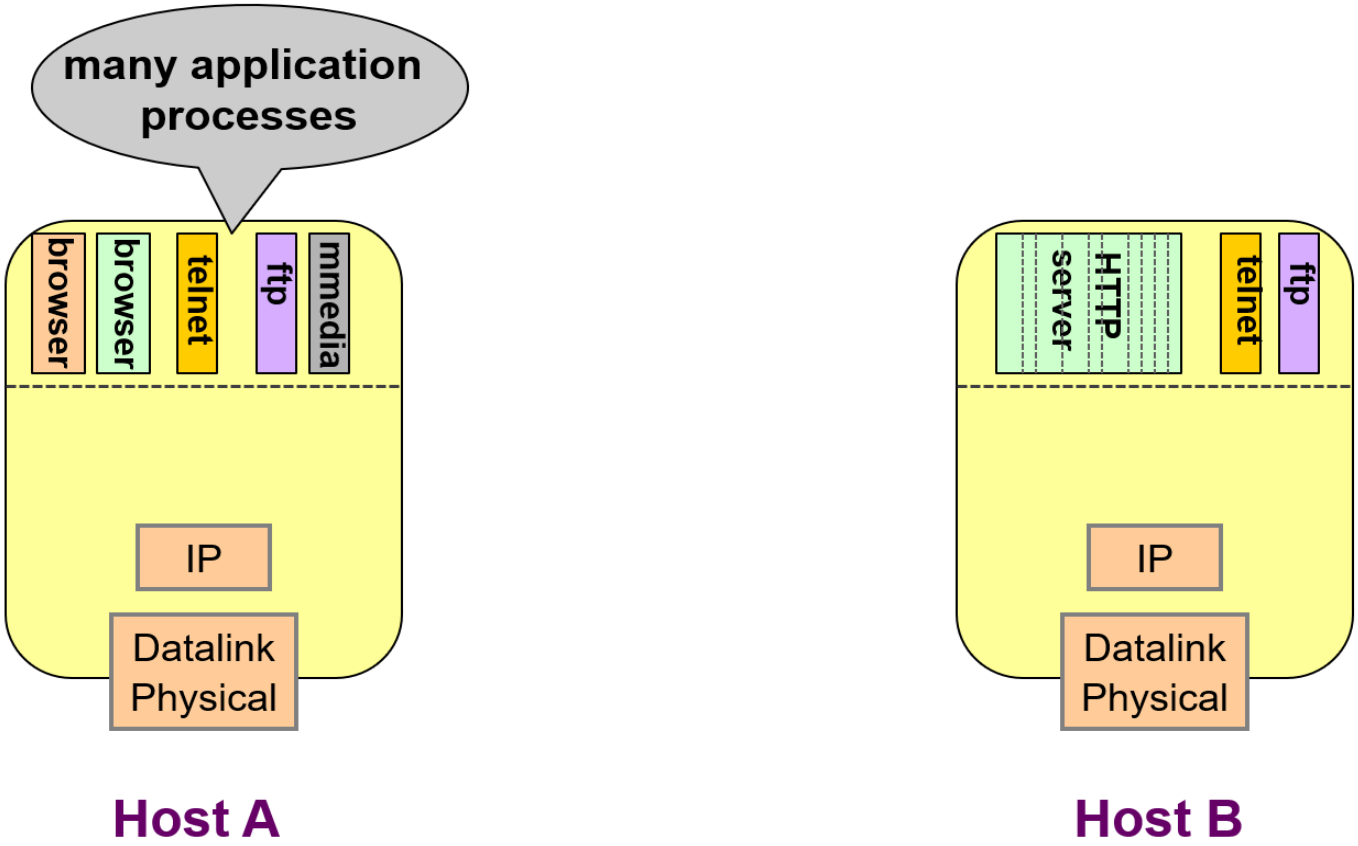
MULTIPLEXING E DEMULTIPLEXING





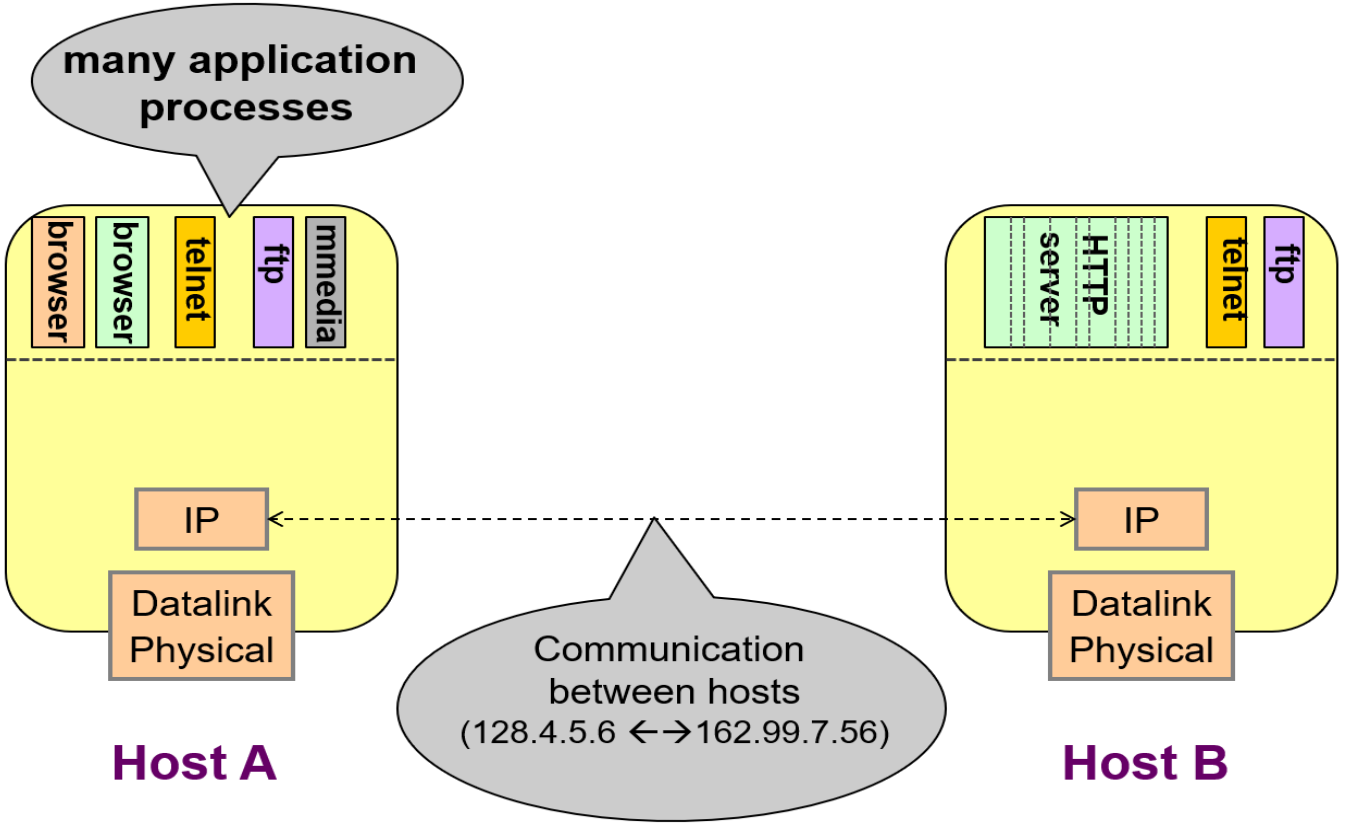
- i pacchetti IP packets sono indirizzati verso un host, ma la comunicazione end-to-end avviene tra processi applicativi negli end host
  - necessario un meccanismo per decidere quali pacchetti indirizzare a quali applicazioni (*multiplexing/demultiplexing*)

MULTIPLEXING E DEMULTIPLEXING



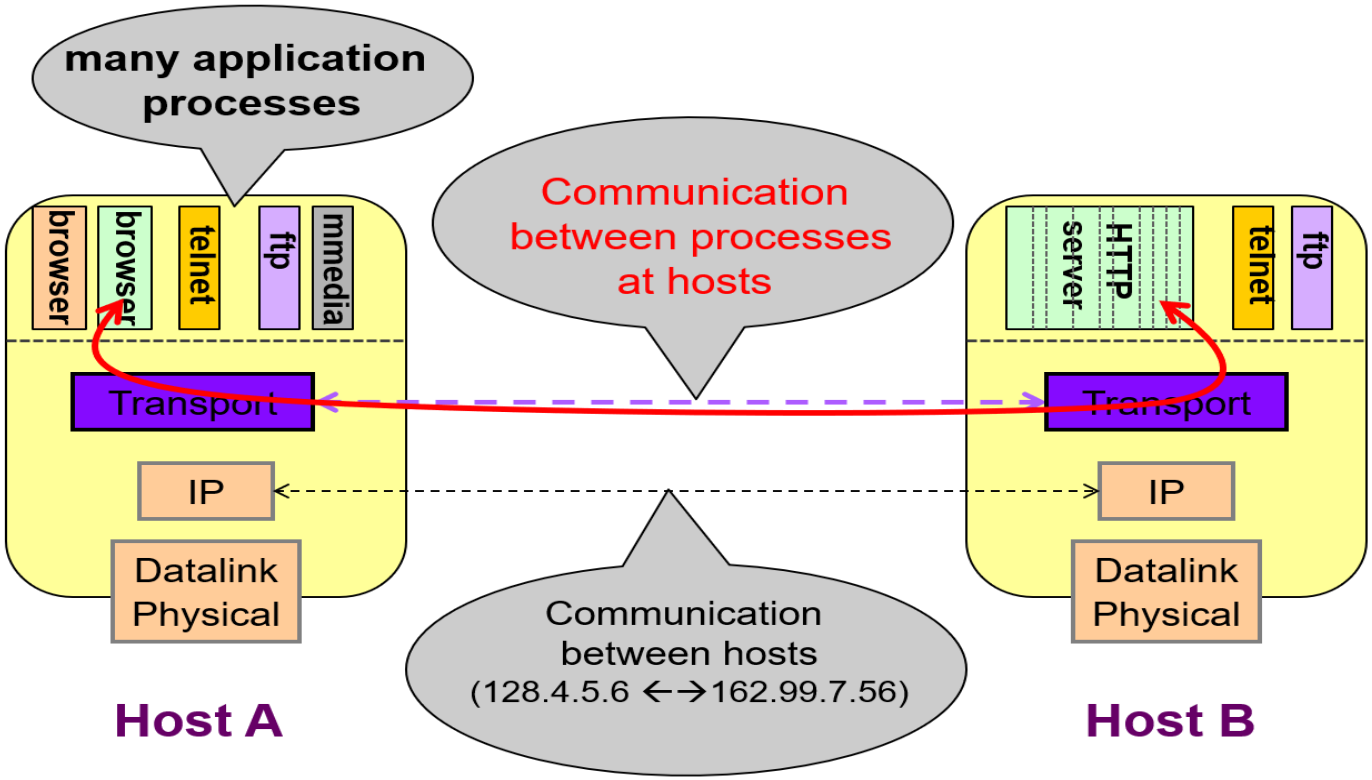
- i pacchetti IP sono indirizzati verso un host, ma la comunicazione end-to-end avviene tra processi applicativi negli end host
  - necessario un meccanismo per decidere quali pacchetti indirizzare a quali applicazioni (*multiplexing/demultiplexing*)

MULTIPLEXING E DEMULTIPLEXING



- i pacchetti IP sono indirizzati verso un host, ma la comunicazione end-to-end avviene tra processi applicativi negli end host
- necessario un meccanismo per decidere quali pacchetti indirizzare a quali applicazioni (*multiplexing/demultiplexing*)

MULTIPLEXING E DEMULTIPLEXING

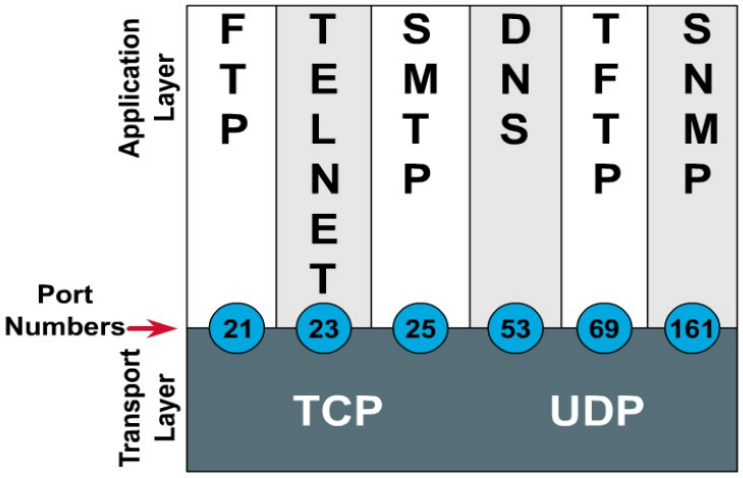


- i pacchetti IP sono indirizzati verso un host, ma la comunicazione end-to-end avviene tra processi applicativi negli end host
  - necessario un meccanismo per decidere quali pacchetti indirizzare a quali applicazioni (multiplexing/demultiplexing)

MULTIPLEXING E DEMULTIPLEXING: LE PORTE

- 16-bit: 65,535 porte usate da TCP o UDP per identificare l'end point di una comunicazione logica

- well-known port numbers (intervallo 0-1023), reserved/contact ports.
  - assegnate e controllate da ICANN (Internet Corporation for Assigned Names & Numbers)
  - usate per servizi/protocolli Internet standard.
  - client programmati per connettersi ad una porta specifica del server: browser si collega automaticamente alla porta 80



FTP	21	File Transfer	TCP
SSH	22	Secure Shell Login	TCP
Telnet	23	Remote login	TCP
SMTP	25	Simple Mail Transfer	TCP
DNS	53	Domain Name Service	TCP/UDP
TFTP	69	Trivial FTP	UDP
Finger	79	user lookup	TCP
HTTP	80	World Wide Web	TCP
POP3	110	Post Office Protocol V3	TCP
SNMP	161	Network Management	UDP
SNMP Traps	162	Network Management Traps	UDP
HTTPS	443	Secure HTTP	TCP
IMAP4	993	Message Access over TLS/SSL	TCP

http://www.iana.org/assignments/port-numbers

APPLICAZIONI UDP

Port	Application Layer Protocol	Description
7	Echo	Echoes a received datagram back to the sender
11	Users	Active users
13	Daytime	Returns the date and the time
53	DNS	Domain Name Service
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol

QUANDO USARE UDP?

- invio pacchetti contenenti informazioni per la gestione della rete
  - Applicazioni che mirano soprattutto alla efficienza: DNS
  - applicazioni per la gestione della rete spesso vengono eseguite quando la rete è sotto stress
    - SNMP (Simple Network Management Protocol)
    - poco adatto TCP che controlla congestione
- applicazioni di tipo domanda-risposta che richiedono risposte immediate
  - Echo, Daytime, Users
- applicazioni/funzioni per cui non è importante la affidabilità
  - keep-alive tra peer in molte applicazioni P2P
- applicazioni che richiedono multicast IP o broadcast
  - TCP supporta solo applicazioni punto-a-punto, unicast

QUANDO USARE UDP?

- usato per applicazioni multimedia real-time
  - applicazioni sensibili alla responsiveness, mentre possono tollerare piccole perdite di pacchetti
- VoIP
  - codecs e protocolli a livello applicativo progettati per compensare un certo livello di perdita di pacchetti
  - richiedono che i pacchetti arrivino ad intervalli regolari e che ci sia un basso ritardo nella trasmissione
    - basso jitter e delay
  - protocolli affidabili come TCP hanno caratteristiche che possono influenzare negativamente jitter e delay
    - pacchetti persi causano ritrasmissioni che causano ritardi e aumentano jitter
    - ordinamento di pacchetti richiede gestione delle code che aumenta sia il jitter che il delay

TRASFERIMENTO AFFIDABILE: RIASSUNTO

- Stop-and-wait protocol
  - non trasmette un segmento fino a che il precedente è stato riscontrato
  - usa la rete in modo poco efficiente
- Pipelining protocols
  - Go-back-N
    - dimensione della finestra W: non invia più di W segmenti non riscontrati
    - acknowledgement cumulativi
    - la ricezione di un numero di sequenza n implica che tutti i segmenti fino ad n sono stati ricevuti
    - timeout: ritrasmette tutti i segmenti non riscontrati
  - Selective Repeat (SR)
    - riscontra i singoli segmenti
    - finestra del mittente: N segmenti a partire dal primo pacchetto non riscontrato
    - un timer per ogni pacchetto: ritrasmette solo un pacchetto al timeout
    - finestra del ricevente: N segmenti a partire dal primo pacchetto mancante
    - il ricevente bufferizza i segmenti riscontrati, ma fuori ordine
    - il ricevente consegna alla applicazione i segmenti, in ordine

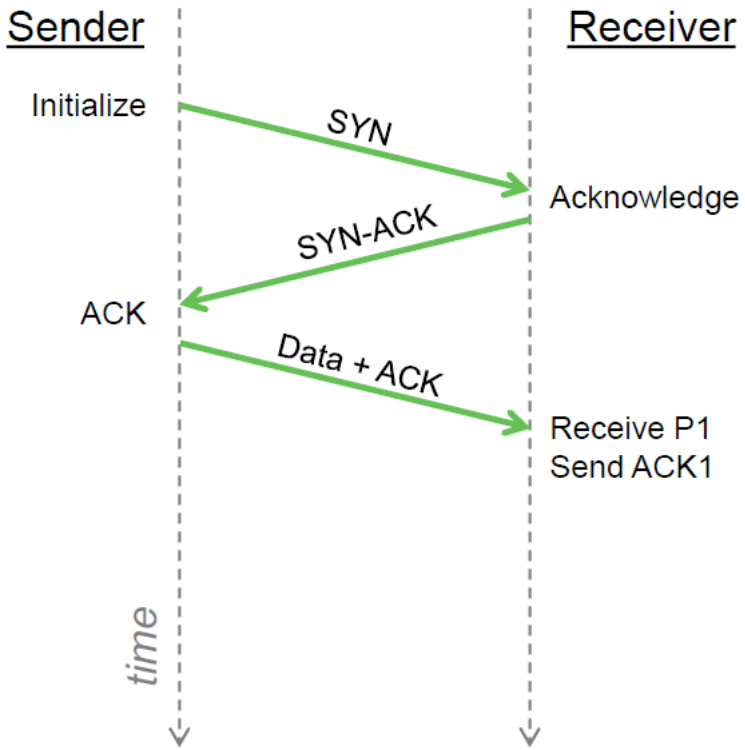
RELIABLE TRANSFER: RIASSUNTO

- Protocollo di livello trasporto ... come UDP
- ma...
  - connection-oriented
  - canali di comunicazione bidirezionale
  - trasferimento affidabile
  - controllo del flusso
- lo "stato" della connessione è mantenuto solo nei network stack degli end-system
  - connessione gestita negli end systems
  - routers non sono consapevoli di TCP
  - nessuna risorsa dedicata ad una connessione TCP nei routers

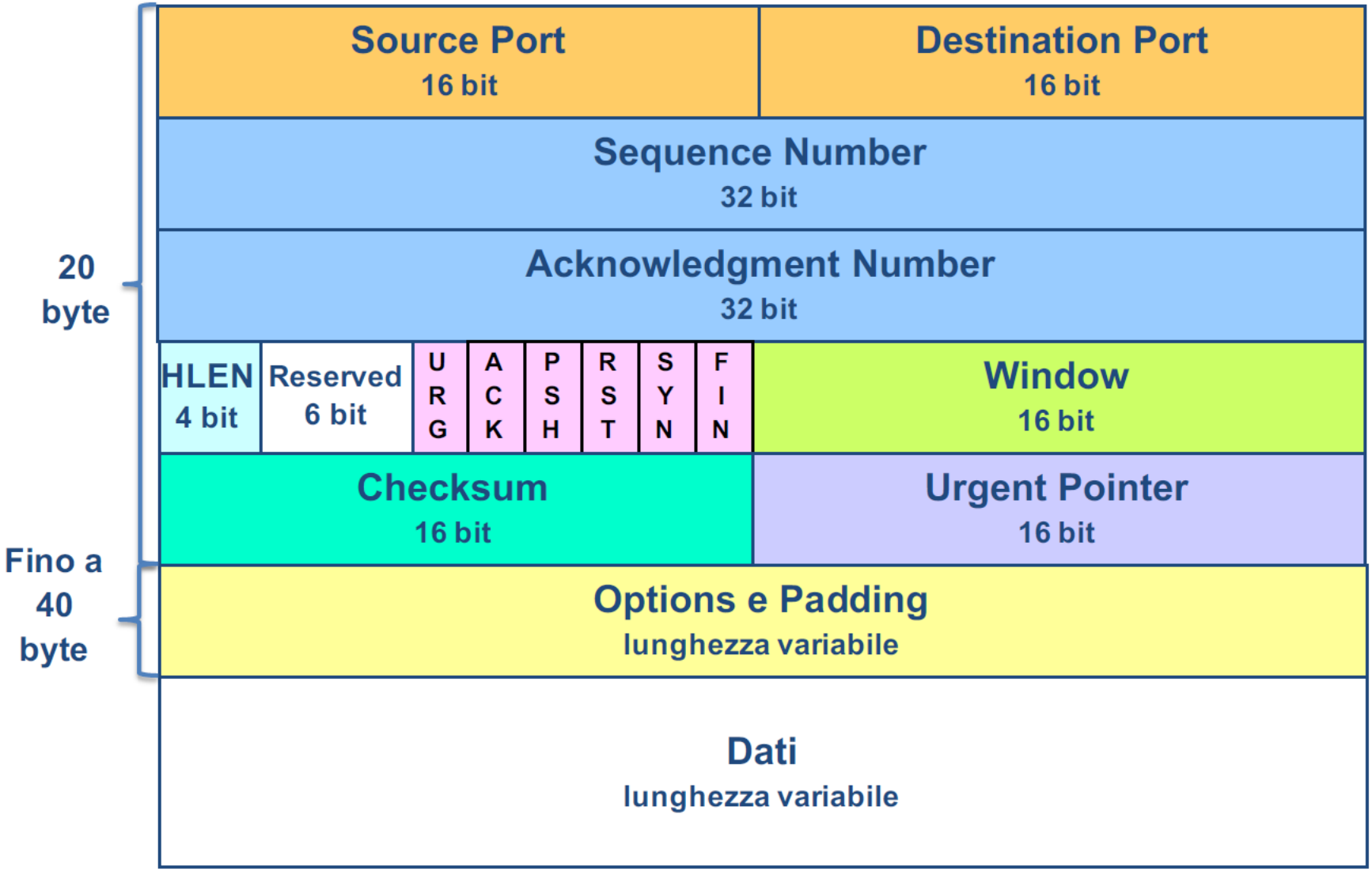
TCP: SET UP DELLA CONNESSIONE

Three way handshake:

- prima dell'invio dei dati
- negoziazione dei parametri che controllano la connessione
- inizializzazione stato per la gestione di quella connessione negli end-host

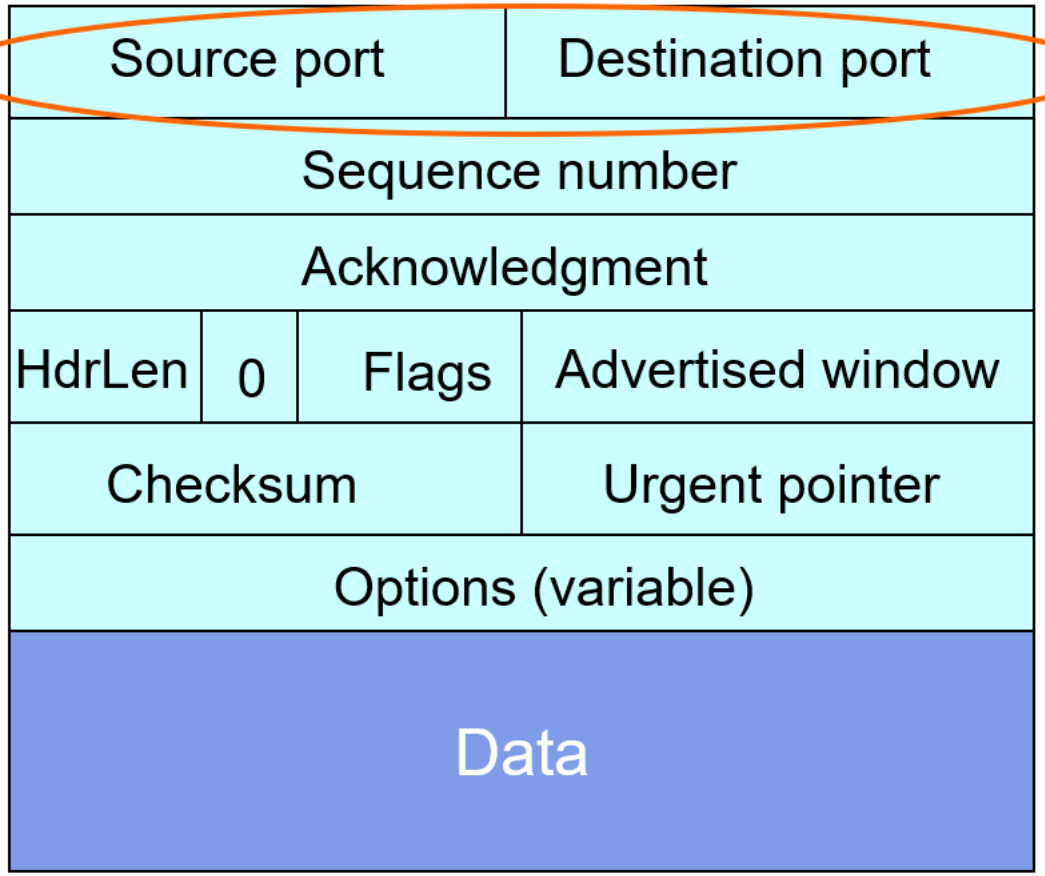


TCP HEADER


Source Port
16 bit
Destination Port
16 bit
Sequence Number
32 bit
Acknowledgment Number
32 bit
20 byte
HLEN
4 bit
Reserved
6 bit
URG
ACK
PSH
RST
SYN
FIN
Window
16 bit
Checksum
16 bit
Urgent Pointer
16 bit
Fino a 40 byte
Options e Padding
lunghezza variabile
Dati
lunghezza variabile

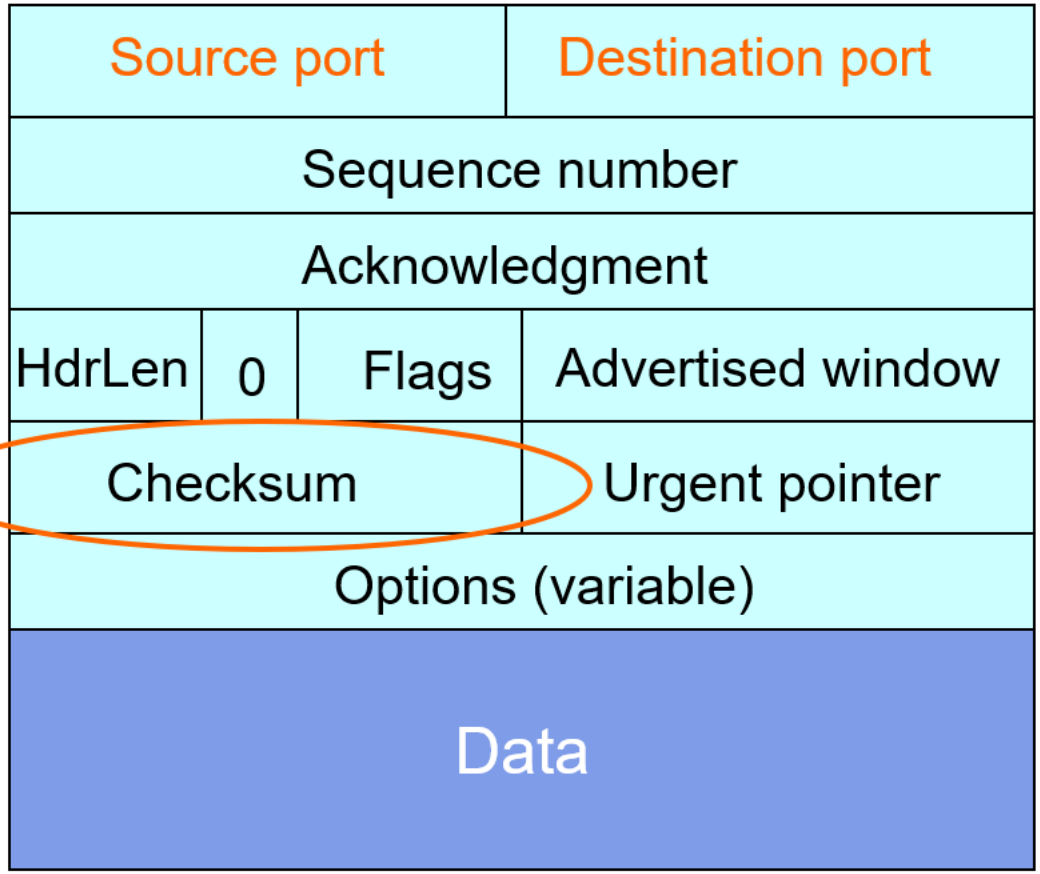
TCP HEADER: MULTIPLEXING E DEMULTIPLEXING

USATO PER MULTIPLEXING E DEMULTIPLEXING



TCP HEADER: CHECKSUM

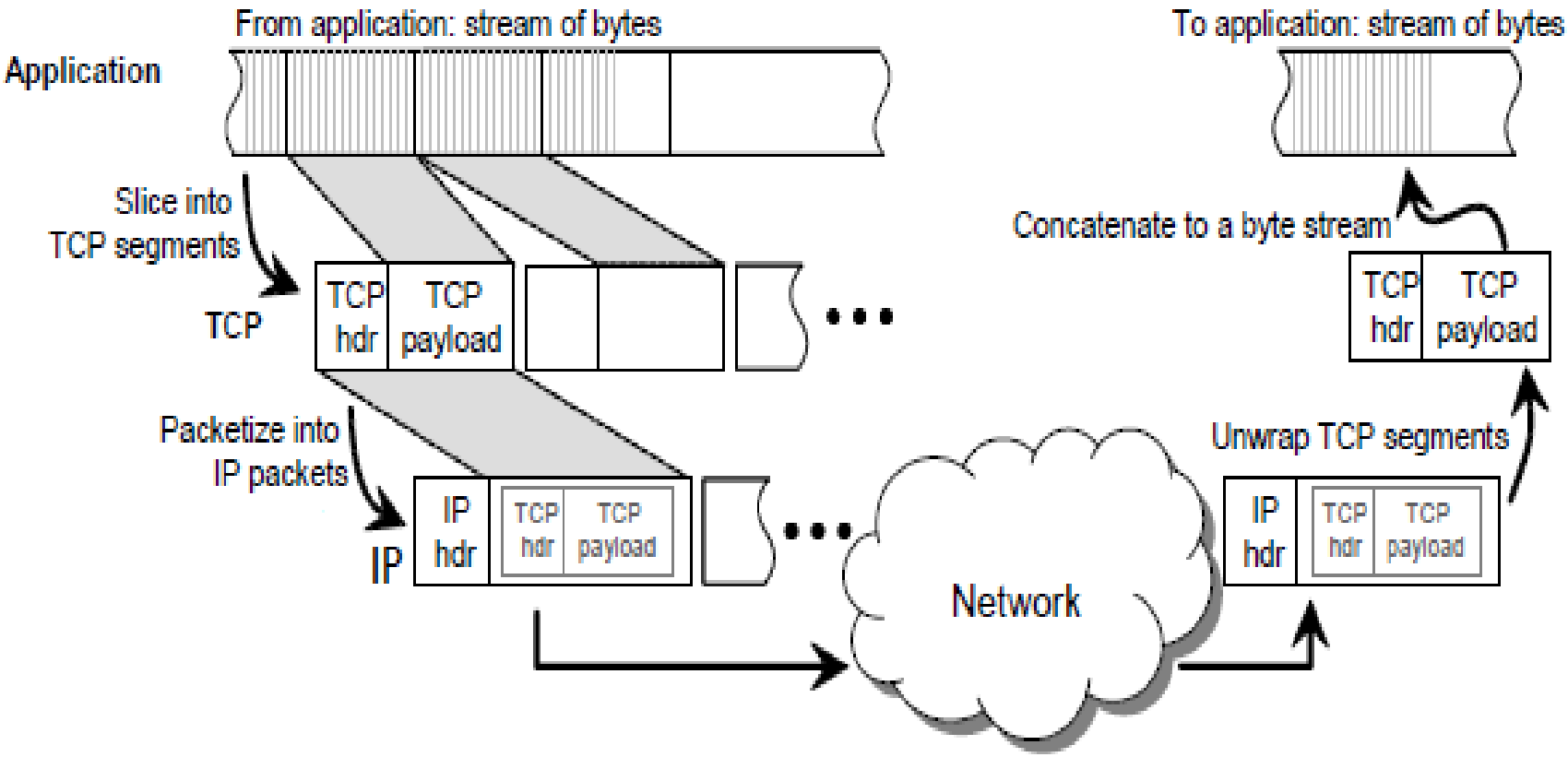
CALCOLATA SU HEADER E DATA



TCP STREAM DELIVERY

- a differenza di UDP, TCP è un protocollo "stream oriented"
- mittente e destinatario scambiano uno stream continuo di bytes.
- il bytes nello stream vengono inviati in pacchetti TCP, generalmente indicanti segmenti TCP
  - segmentazione dello stream di bytes in input
  - segmenti TCP incapsulati in pacchetti IP
- nessuna "delimitazione" tra messaggi consecutivi, che vengono memorizzati nello stesso segmento o in segmenti consecutivi
  - un'applicazione può effettuare più "TCP write" e i dati relativi a tutte le write vengono inclusi in un solo segmento, senza separazione e poi inviati
  - esempio:
    - "write 20 bytes" + "write 30 bytes" + "write 10 bytes"
    - il destinatario non riesce a distinguere i bytes scritti dalle diverse write: può leggere i 60 bytes con due read oppure leggere 30 bytes alla volta.

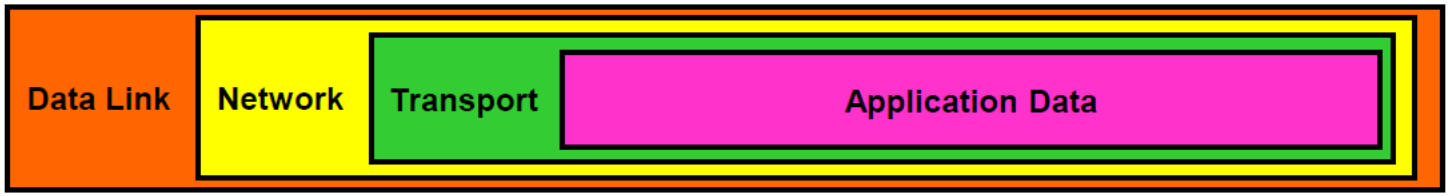
TCP STREAM DELIVERY



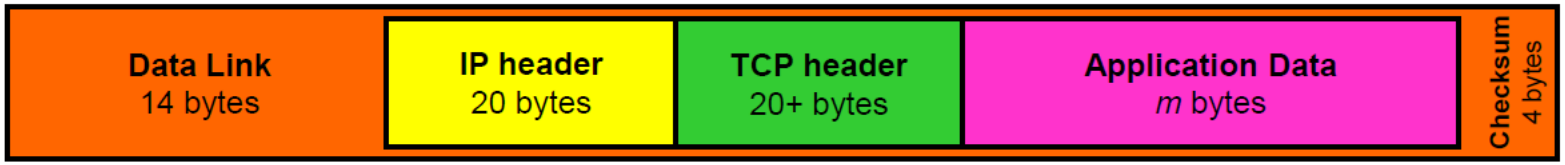
TCP SEGMENTS

- dimensione variabile, ma limitata da Maximum Segment Size (MSS)
- passati al livello IP
- quando sono spediti dal mittente?
  - quando sono arrivati dalla applicazione MSS bytes
  - scadere di un timer che comporta la ritrasmissione del segmento
- come vengono gestiti dal destinatario se arrivano out-of-order?
  - scartati: inefficiente
  - bufferizzati: necessaria gestione di puntatori nel destinatario
  - TCP non lo specifica, ma la maggior parte delle implementazioni bufferizzano

SEGMENTO TCP: MTU



Protocol encapsulation: logical view



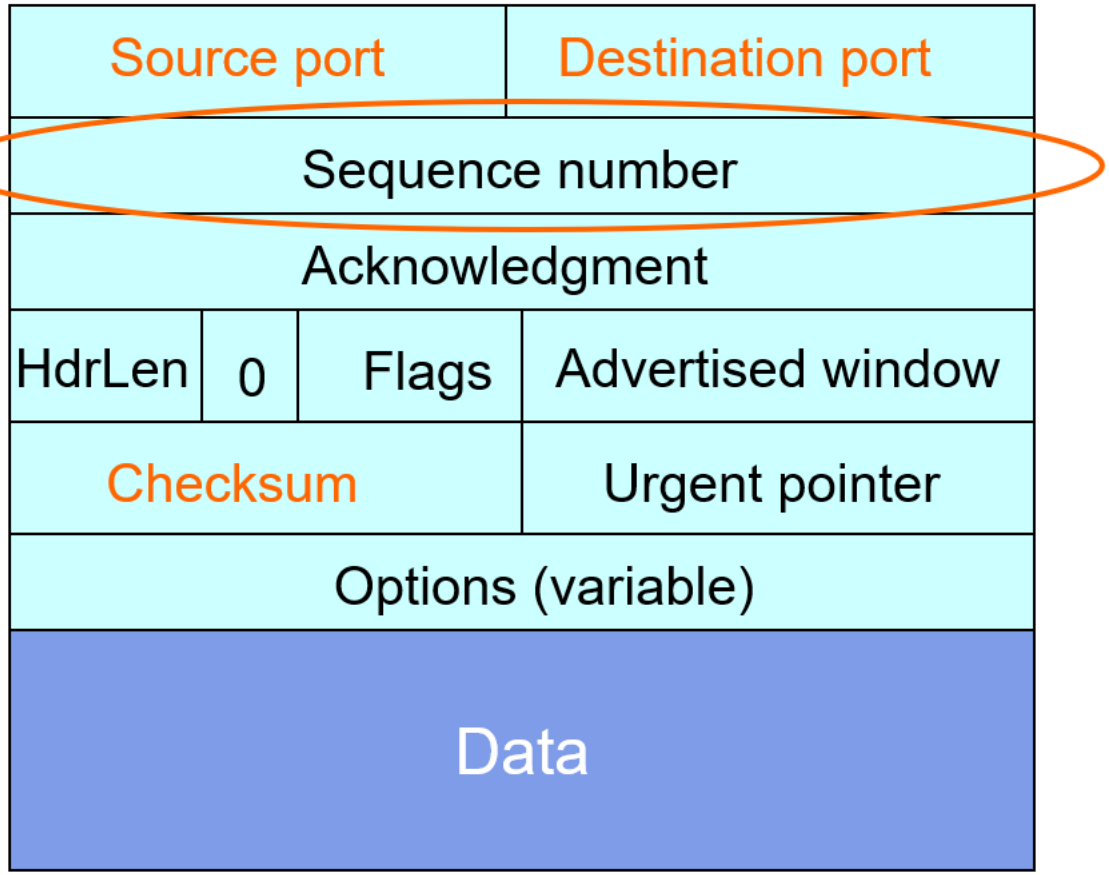
MSS = Maximum Segment Size
= (IP datagram size - 40 bytes)

MTU = Maximum Transmission Unit
1500 bytes for Ethernet v2 (→MSS = 1460 bytes)
9000 bytes for Jumbo frames in gigabit Ethernet (→MSS = 8960 bytes)

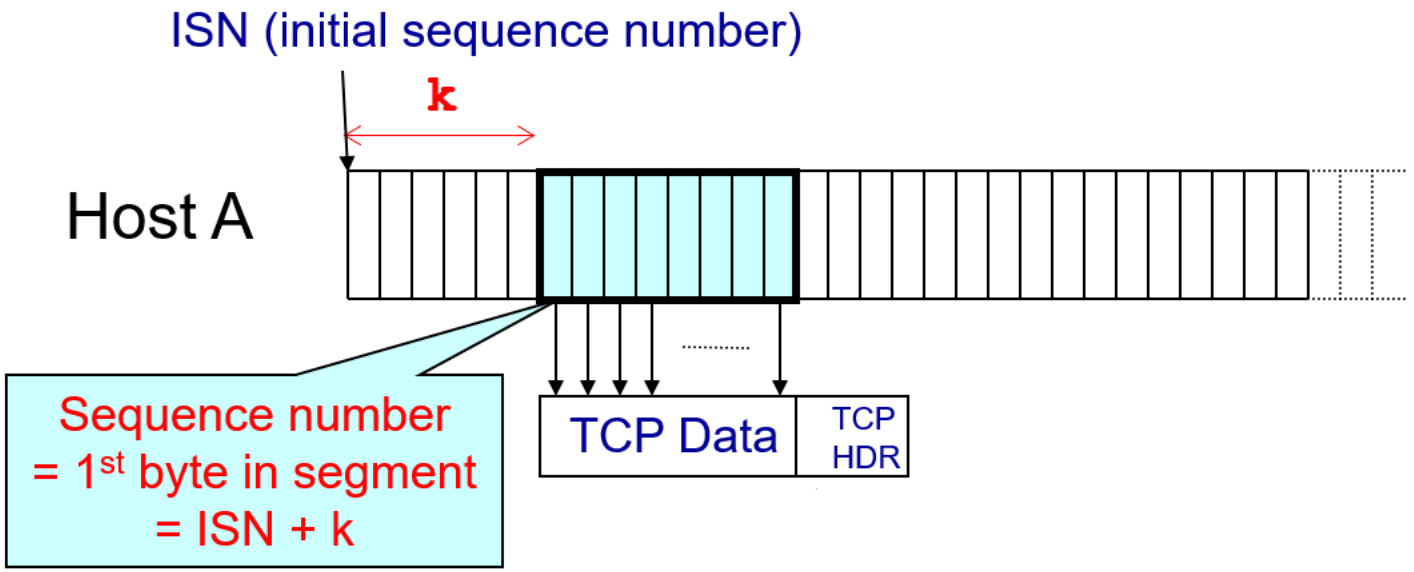
Maximum Segment Size (MSS) is dependent on MTU (=MTU-40)

TCP HEADER: SEQUENCE NUMBER

OFFSET DEL PRIMO BYTE TRASPORTATO CON QUESTO SEGMENTO



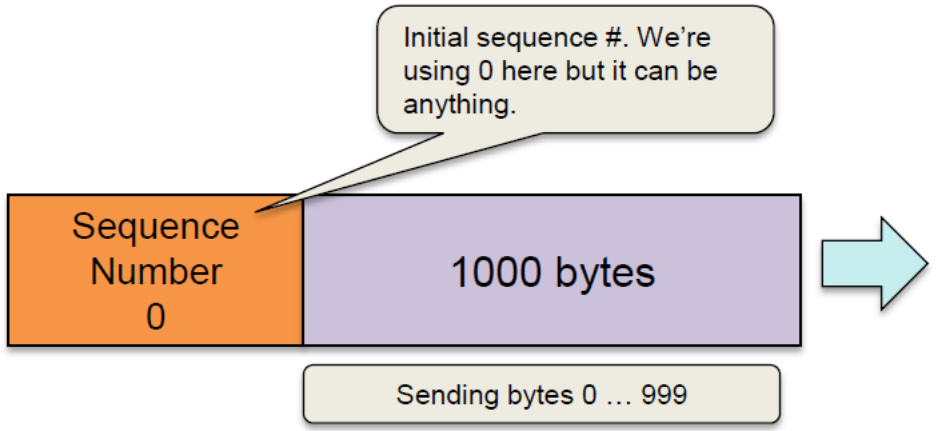
TCP: NUMERI DI SEQUENZA



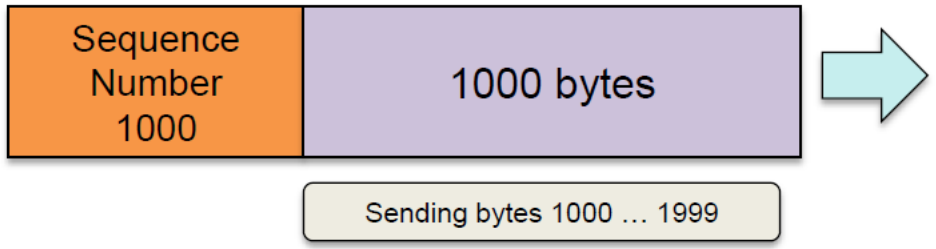
- ISN: numero generato casualmente, non necessariamente 0
  - stabilito in fase di apertura della connessione
- sequence number: 32 bits
  - TCP numera ogni byte trasmesso, per cui ogni byte ha un numero progressivo
  - i segmenti sono formati da gruppi di bytes
  - nell'header del segmento TCP è registrato il numero di sequenza del primo byte nel segmento stesso

TCP: NUMERI DI SEQUENZA

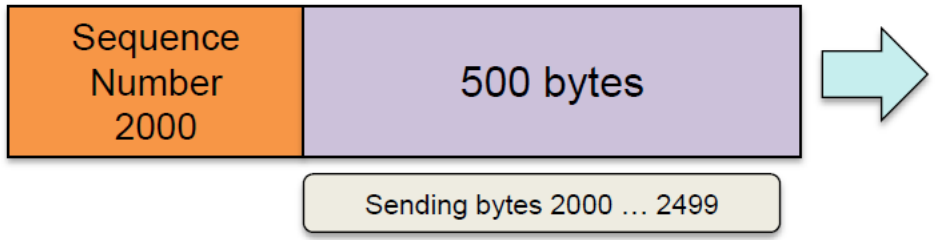
Suppose initial sequence # = 0
and we send a segment with 1000 bytes



Send next segment with 1000 bytes

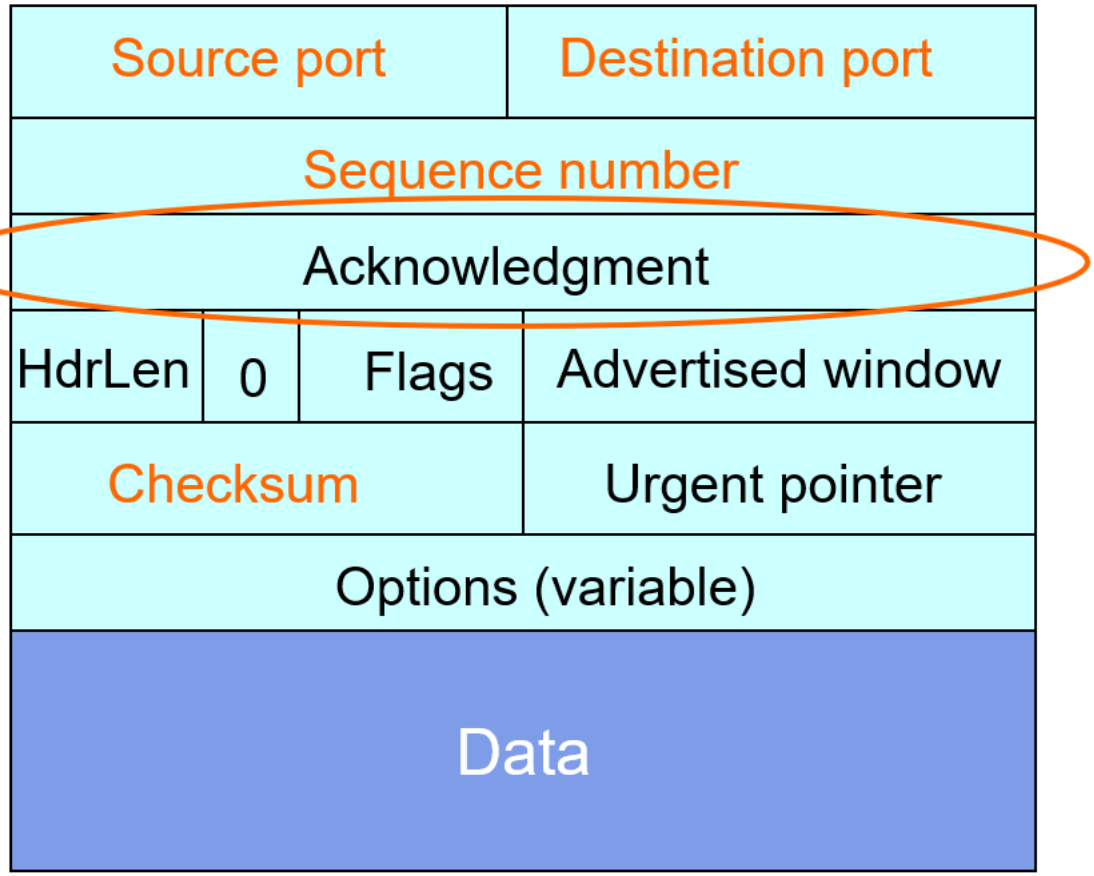


Send next segment with 500 bytes



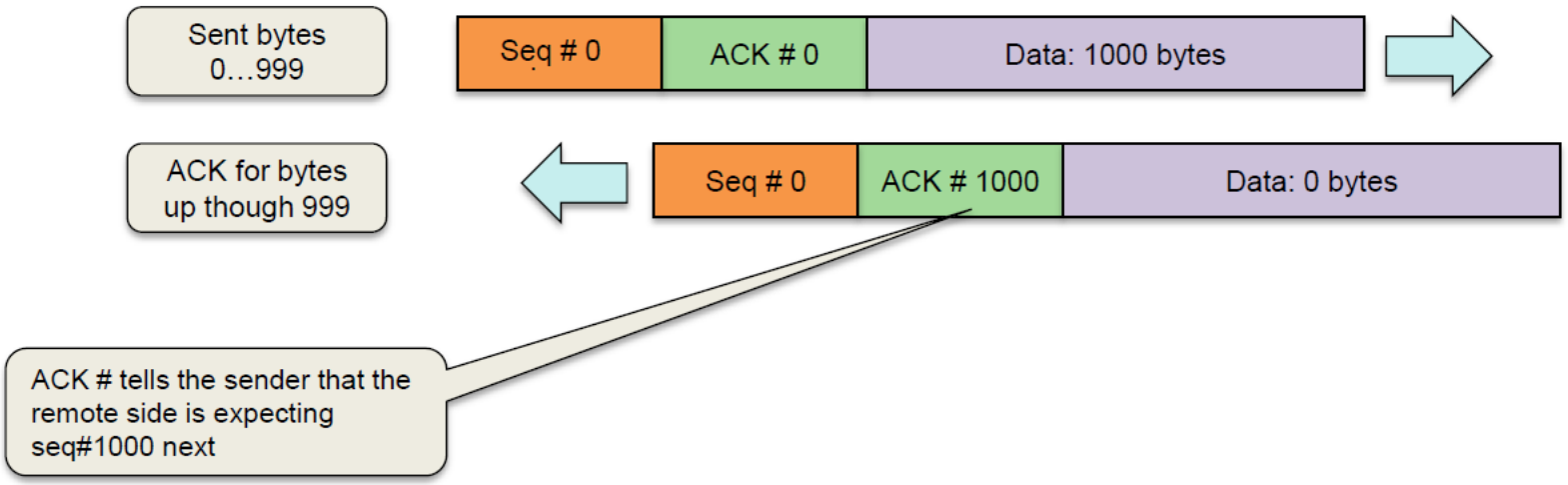
TCP HEADER: ACKNOWLEDGMENT

ACKNOWLEDGMENT:
NUMERO DI SEQUENZA
SUCCESSIVO ALL'
ULTIMO BYTE
RICEVUTO IN ORDINE



TCP ACKNOWLEDGEMENT NUMBER

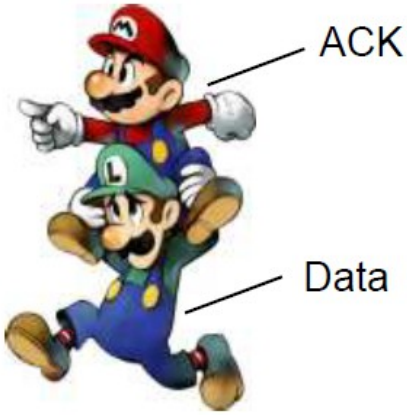
- dopo la ricezione di un pacchetto contenente B bytes, il mittente invia un ACK.
- supponiamo che il primo byte del segmento ricevuto sia X.
  - se tutti i dati precedenti ad X sono già stati ricevuti
    - l'ACK riscontra X+B (che indica il prossimo byte atteso)
  - se l'ultimo byte ricevuto in ordine è Y (Y+1) <X
    - l'ACK riscontrato è Y+1
    - anche se quel byte è già stato riscontrato in precedenza (duplicate ACK)

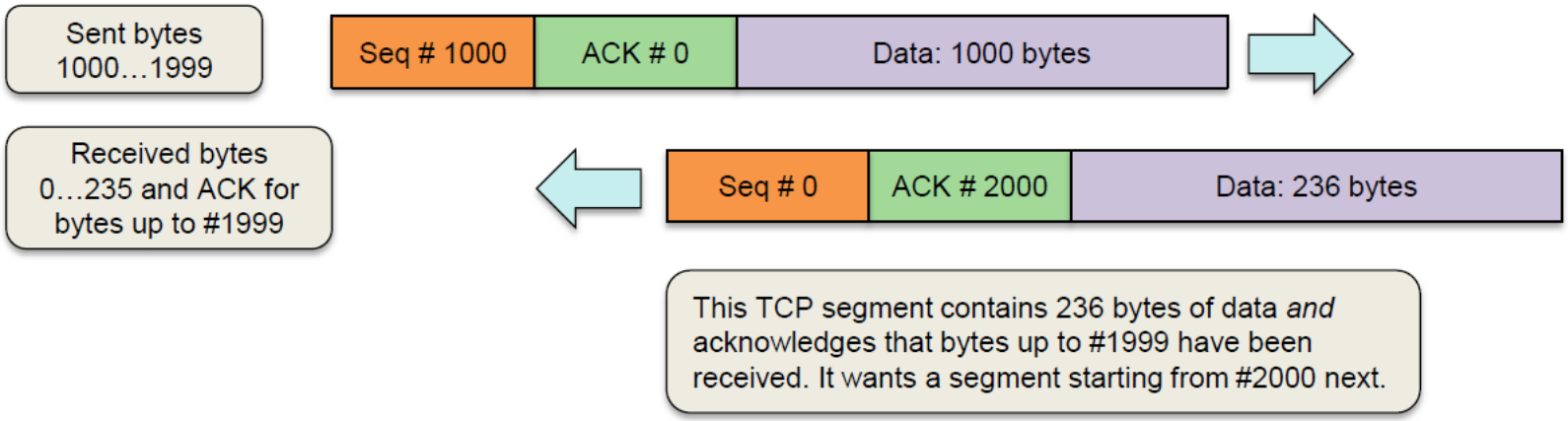


Ricezione in ordine

PIGGYBACKING ACKNOWLEDGEMENTS

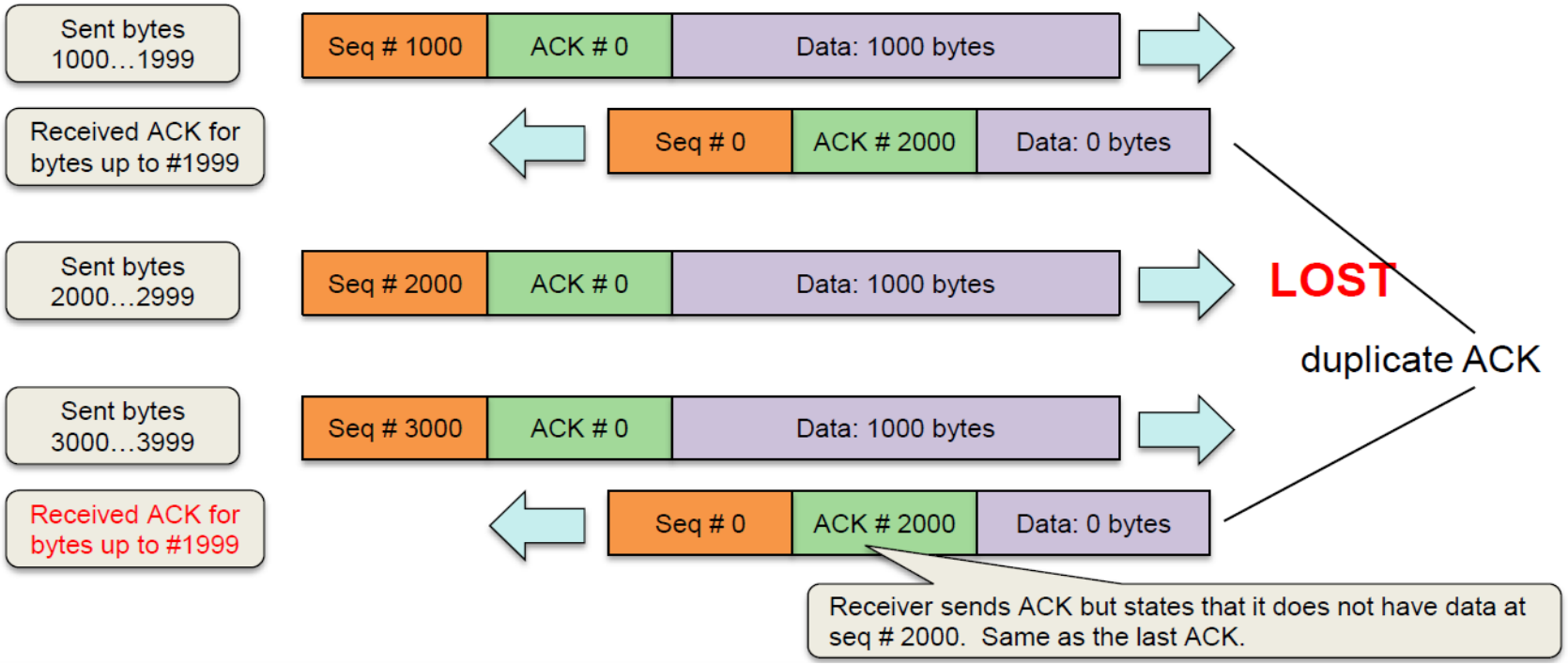
- se il destinatario ha dati da trasmettere su una connessione
  - inserisce l'acknowledgement nell'header TCP per prossimo messaggio (piggyback)
  - non invia un messaggio di acknowledgement separato
- se il destinatario non ha dati da trasmettere
  - l'acknowledgement è inviato senza alcun dato





CUMULATIVE E DUPLICATE ACKNOWLEDGEMENTS

- TCP usa cumulative acknowledgements
  - il numero di ACK è il numero di byte che il ricevente vuole ricevere successivamente
  - tutti i bytes precedenti sono stati ricevuti correttamente
- TCP usa duplicate acknowledgements
  - mittente invia 3 segmenti TCP, ma il secondo viene
    - duplicate acknowledgement: due ACK che si riferiscono allo stesso segmento, quello ricevuto correttamente



OUT OF ORDER SEGMENTS

- un segmento che arriva out of order non è riscontrato
  - viene inviato un ACK che richiede il segmento più vecchio mancante
- Cosa fare con il segmento ricevuto? Due opzioni
  - scartarlo
  - bufferizzarlo ed aspettare i dati mancanti
  - più complesso, ma più efficiente per la rete

DUPLICATE ACK

- Il mittente invia pacchetti di 100B e numeri di sequenza:

  100, 200, 300, 400, 500, 600, 700, 800, 900, …

  - supponiamo che il quinto pacchetto (numero sequenza 500) venga perso, ma non gli altri
  - stream di ACK ricevuti dal mittente

    200, 300, 400, 500, 500, 500, 500,…

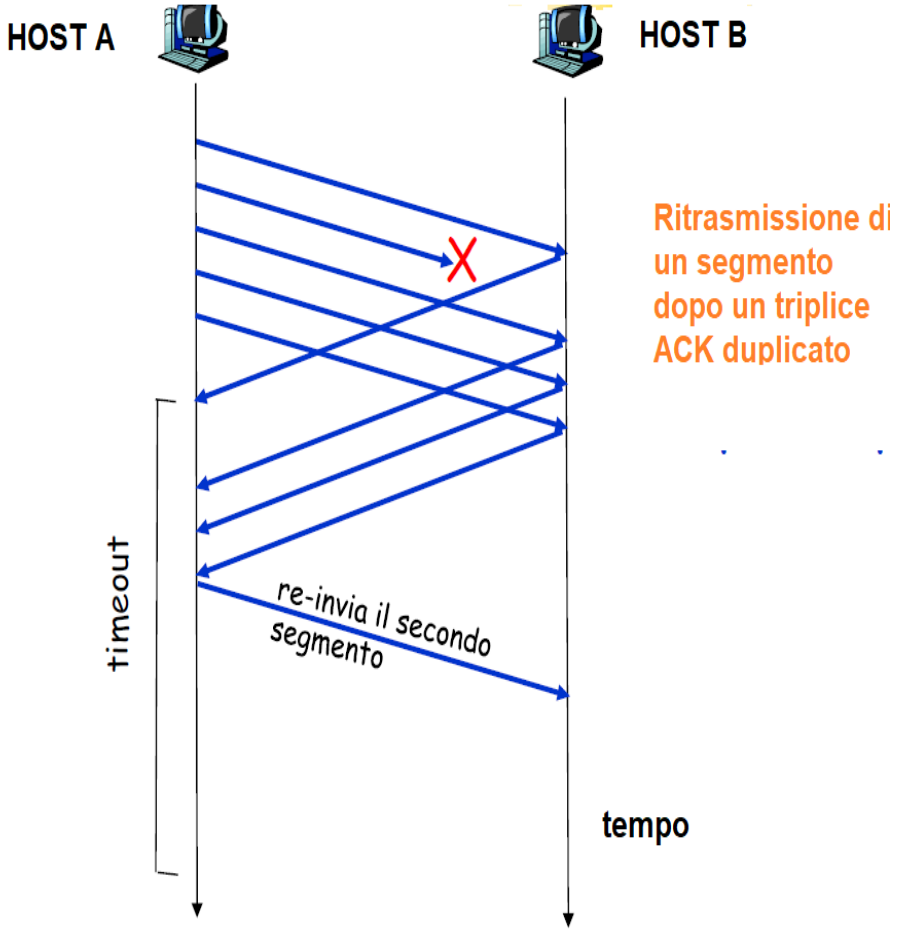
- Gli ACK duplicati indica una "perdita isolata"
  - la non progressione degli ACK significa che il pacchetto 500 non è stato ricevuto
  - uno stream di ACK ripetuti signinficano che I pacchetti successivi sono stati ricevuti

- rispedire il pacchetto dopo aver ricevuto k ACK duplicati
  - TCP usa k=3

TCP FAST RETRASMIT

- TCP garantisce l'affidabilità ristrasmettendo i segmenti dopo:
  - un timeout
  - aver ricevuto 3 ack duplicati cumulativi consecutivi

- idea alla base di fast retrasmit
  - usare il meccanismo degli ACK pe segnalare la perdita di segmenti.
  - aspettare lo scadere dei time-ou provoca un ritardo nell ritrasmissione incrementa la latenz end-to-end
  - reinvio di segmenti prima dell scadere del timer

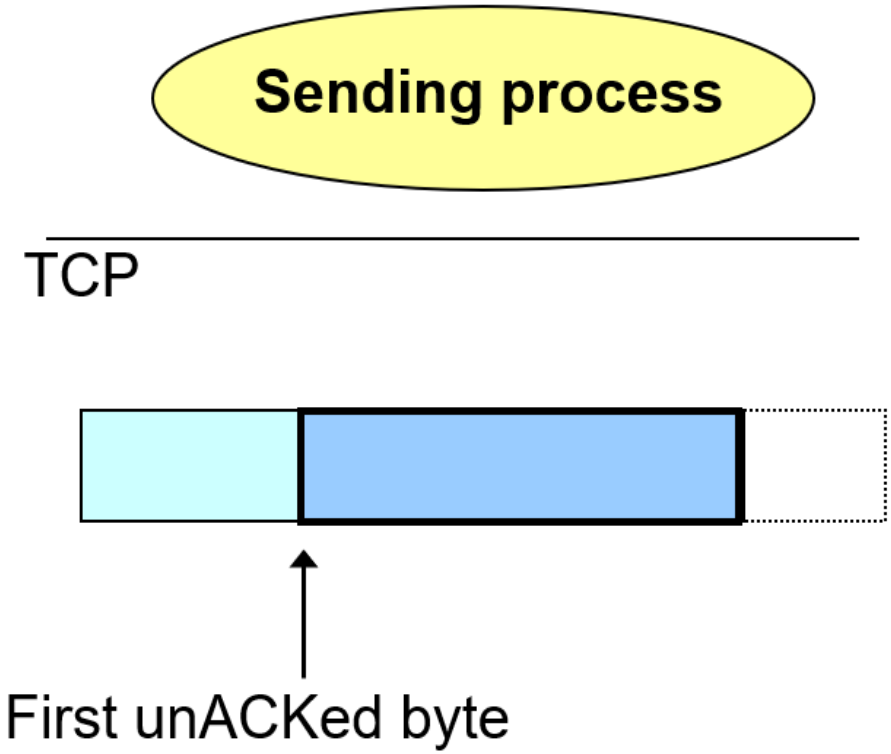


GENERAZIONE DEGLI ACK IN TCP [RFC 112, RFC 2581]

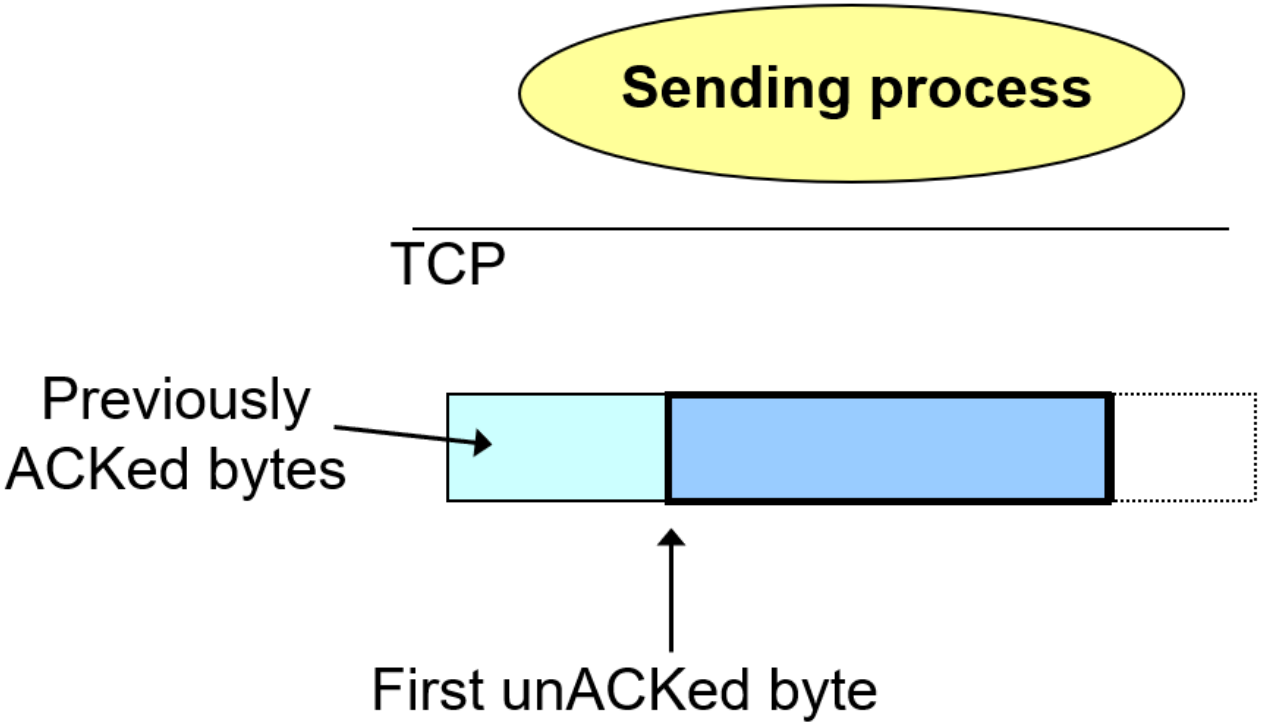
Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed.	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK. (prevents creating extra traffic)
Arrival of in-order segment with expected seq #. One other segment has ACK pending.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected.	Immediately send duplicate ACK, indicating seq. # of next expected byte. (enables fast retransmit)
Arrival of segment that partially or completely fills gap.	Immediately send ACK, provided that segment starts at lower end of gap.

Many of TCP error correction procedures have evolved as a result of more than 15 years of experience!

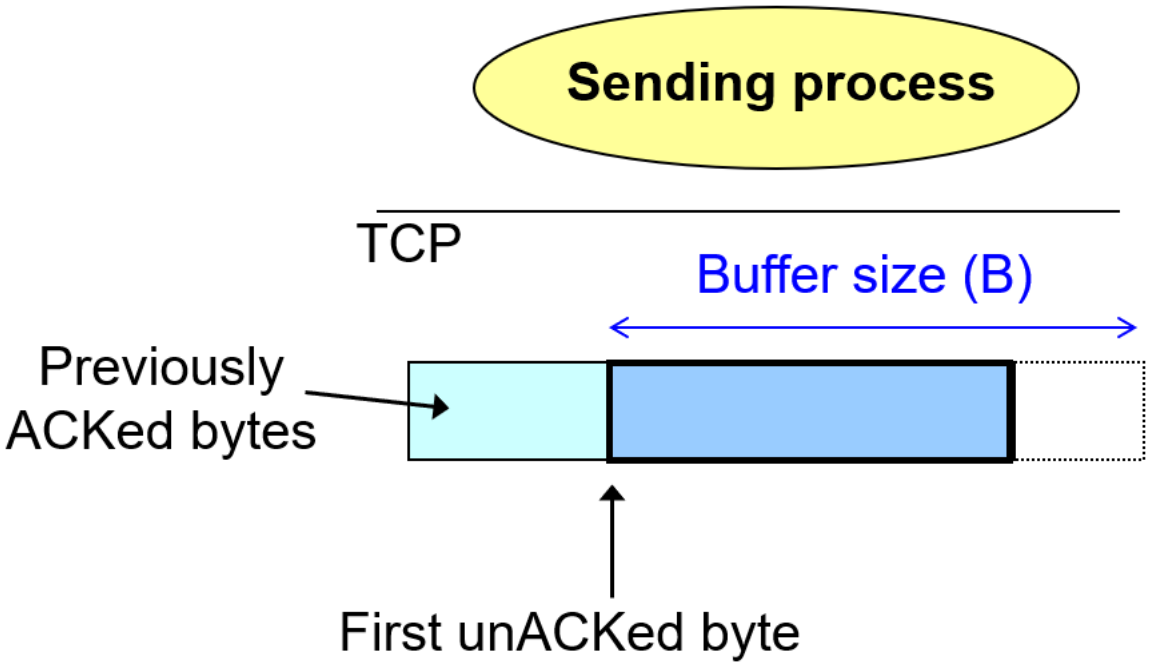
TCP SLIDING WINDOWS NEL MITTENTE



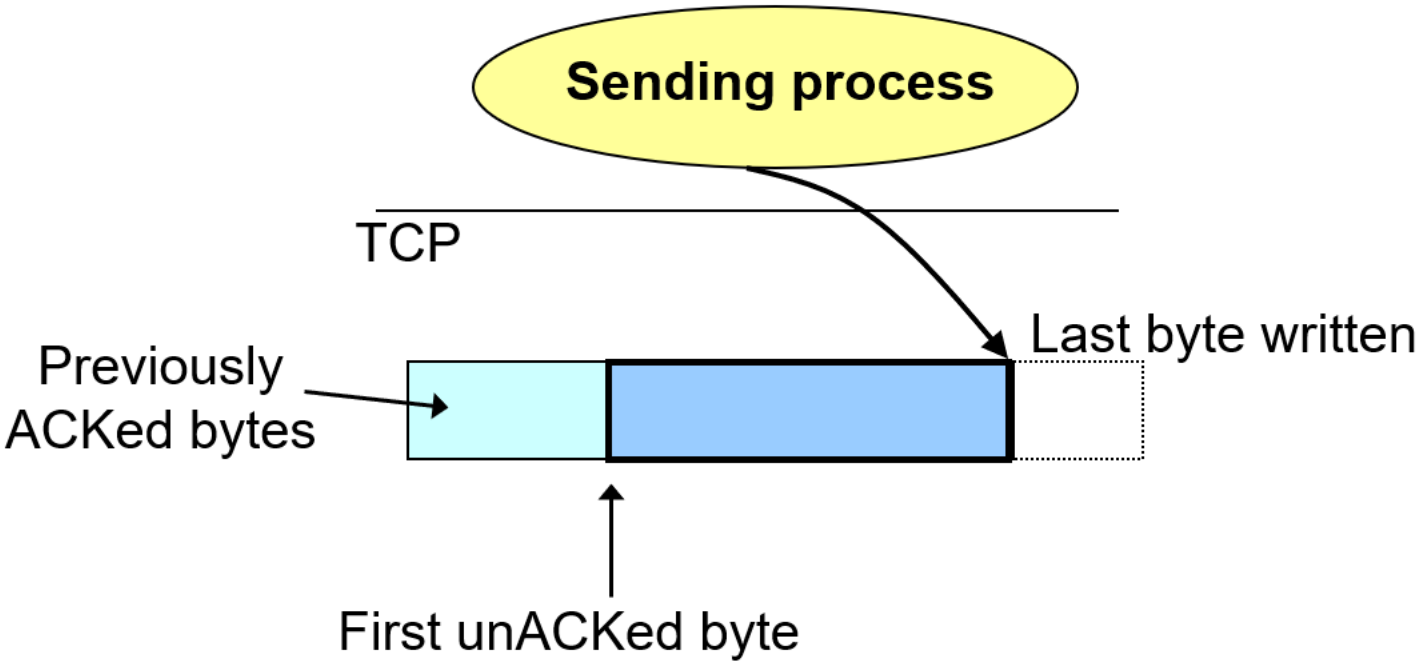
TCP SLIDING WINDOWS NEL MITTENTE



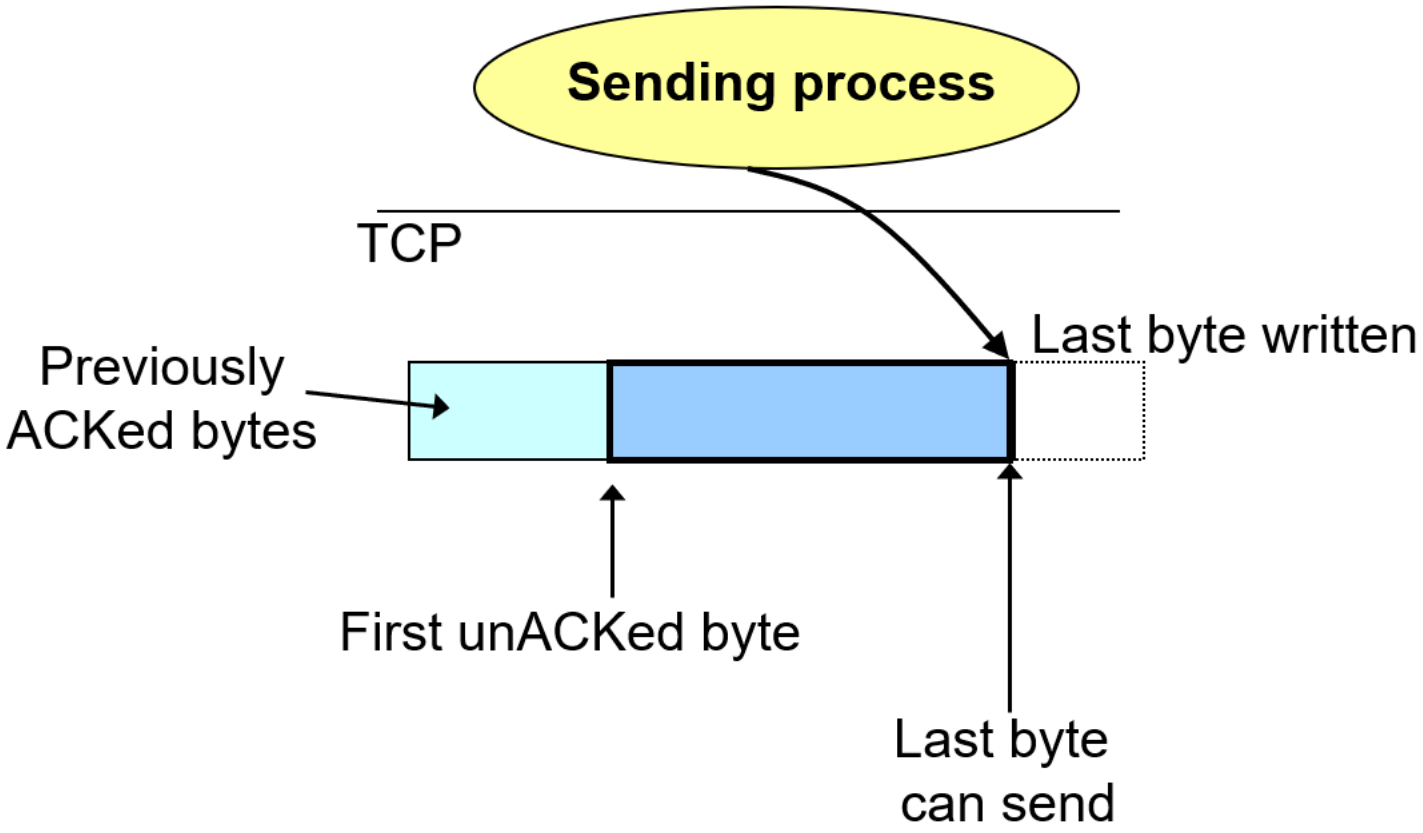
TCP SLIDING WINDOWS NEL MITTENTE



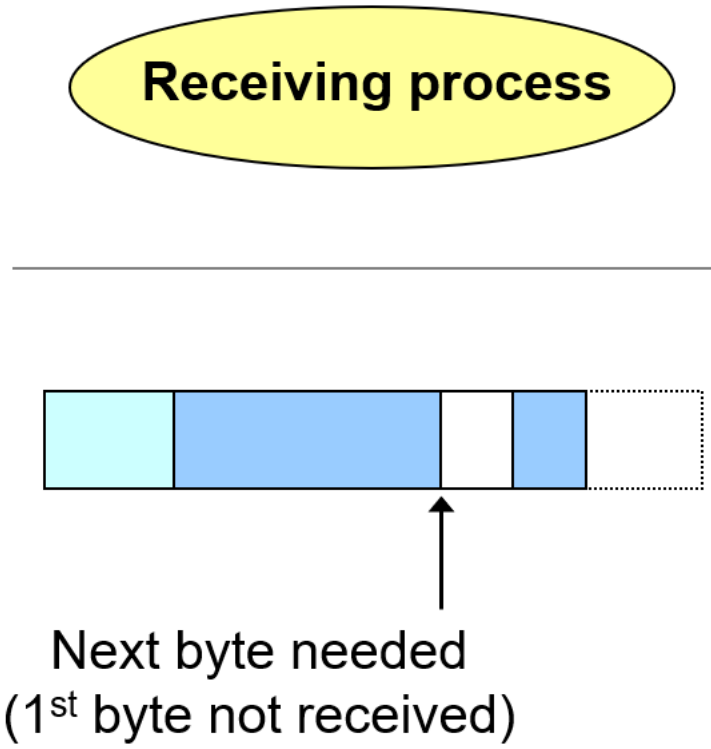
TCP SLIDING WINDOWS NEL MITTENTE



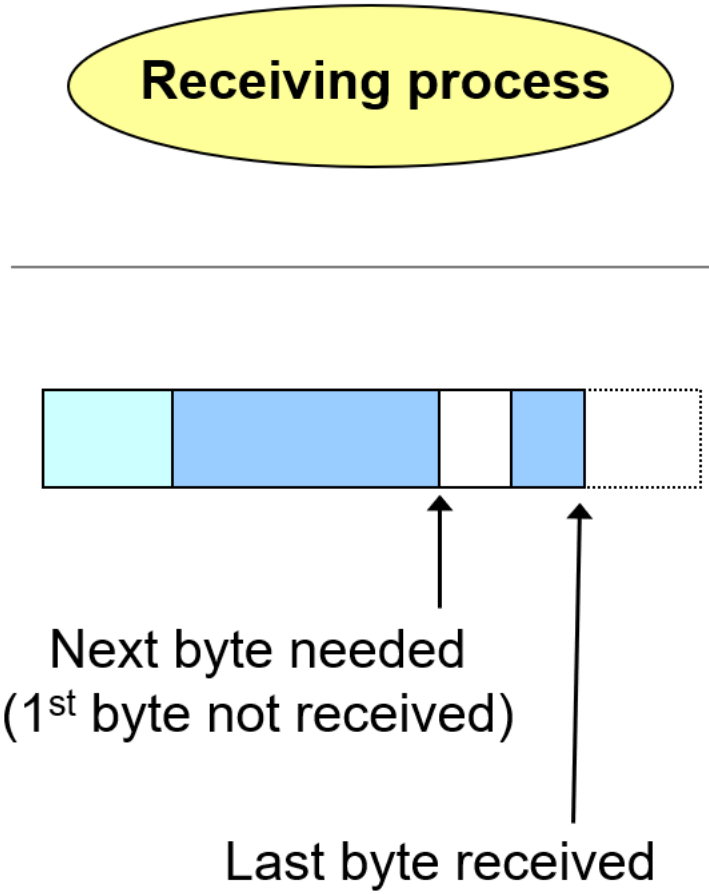
TCP SLIDING WINDOWS NEL MITTENTE



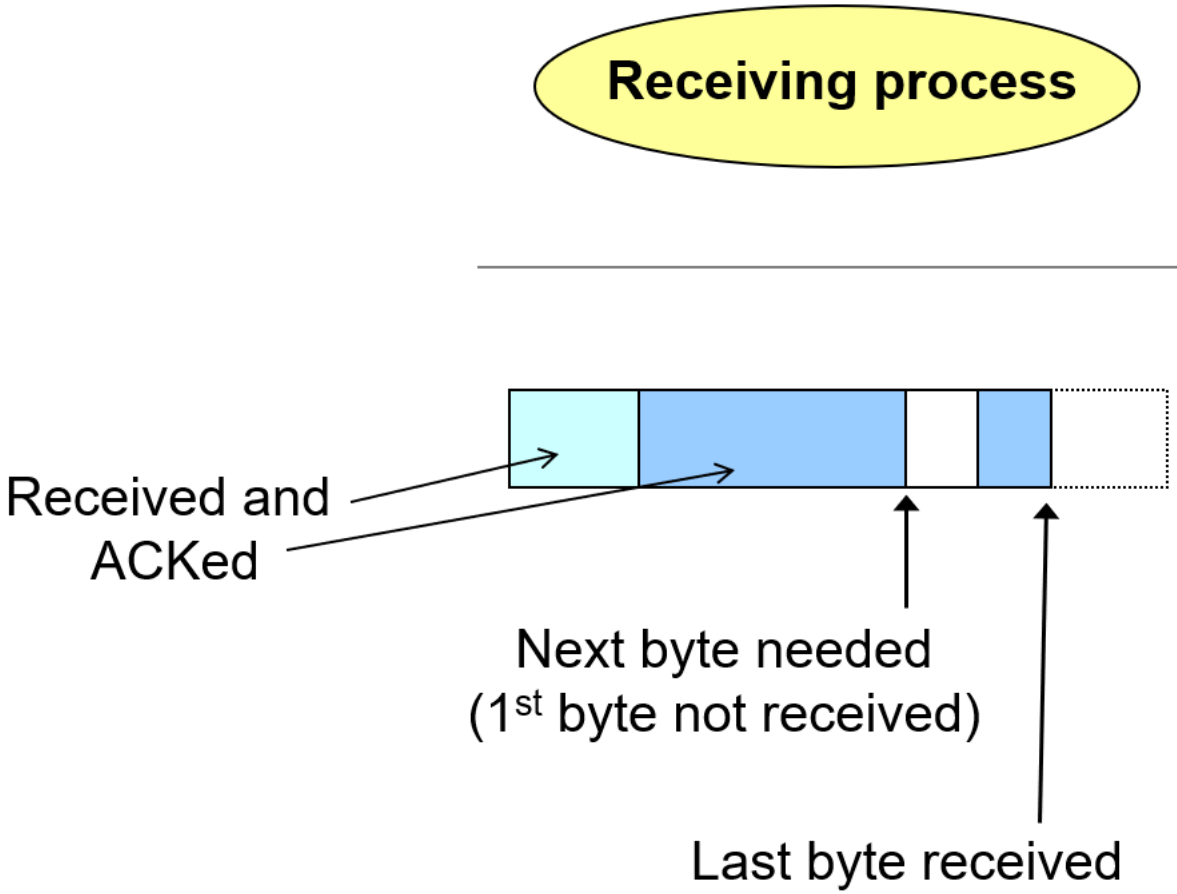
TCP SLIDING WINDOWS NEL DESTINATARIO



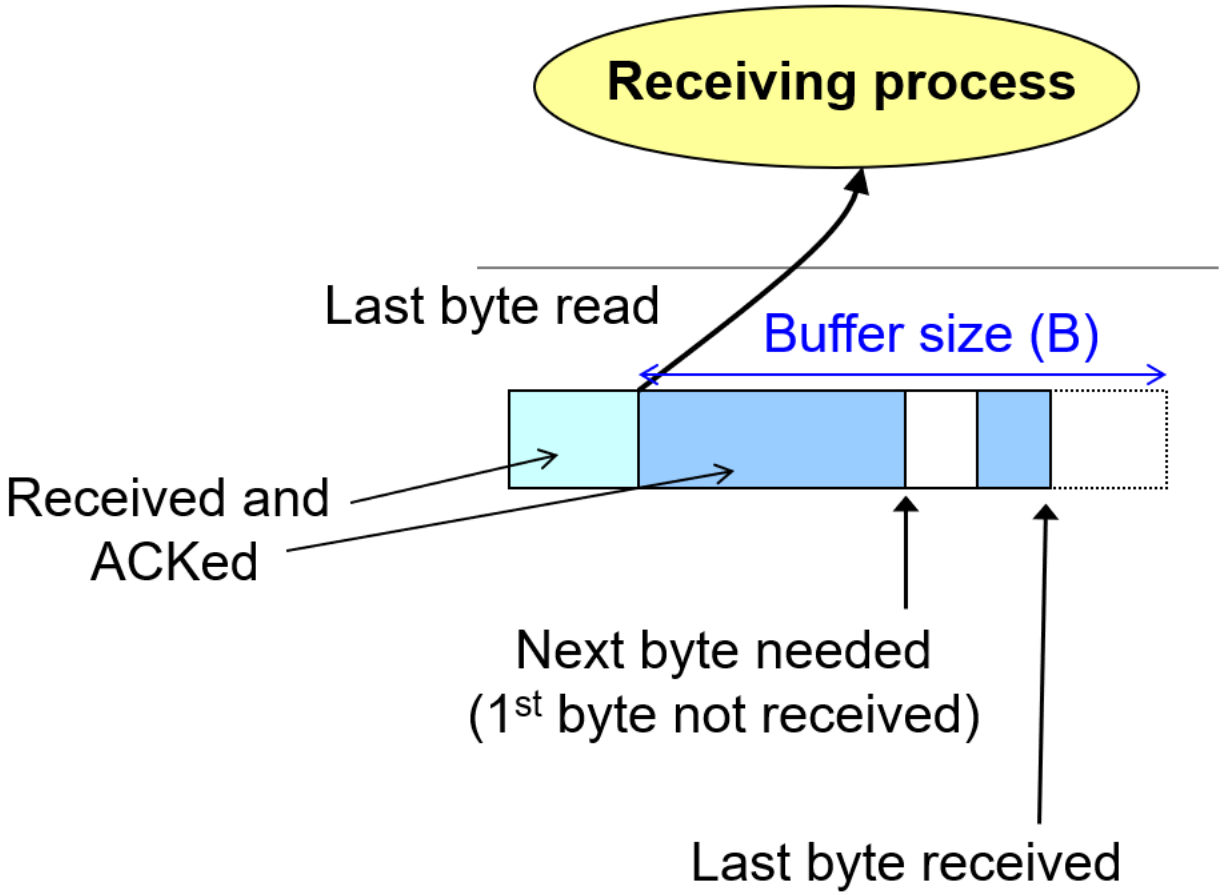
TCP SLIDING WINDOWS NEL DESTINATARIO



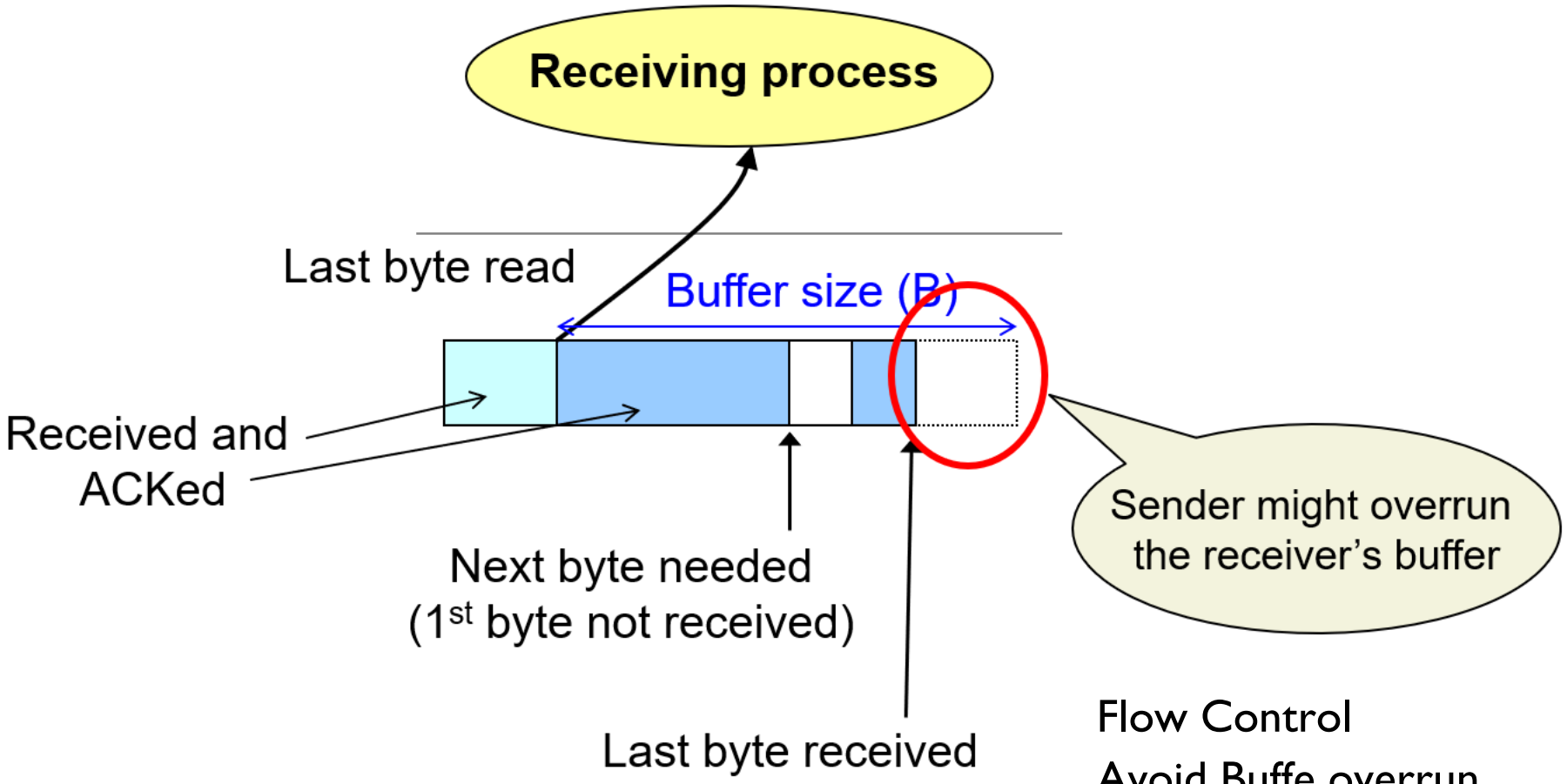
TCP SLIDING WINDOWS NEL DESTINATARIO



TCP SLIDING WINDOWS NEL DESTINATARIO



TCP SLIDING WINDOWS NEL DESTINATARIO

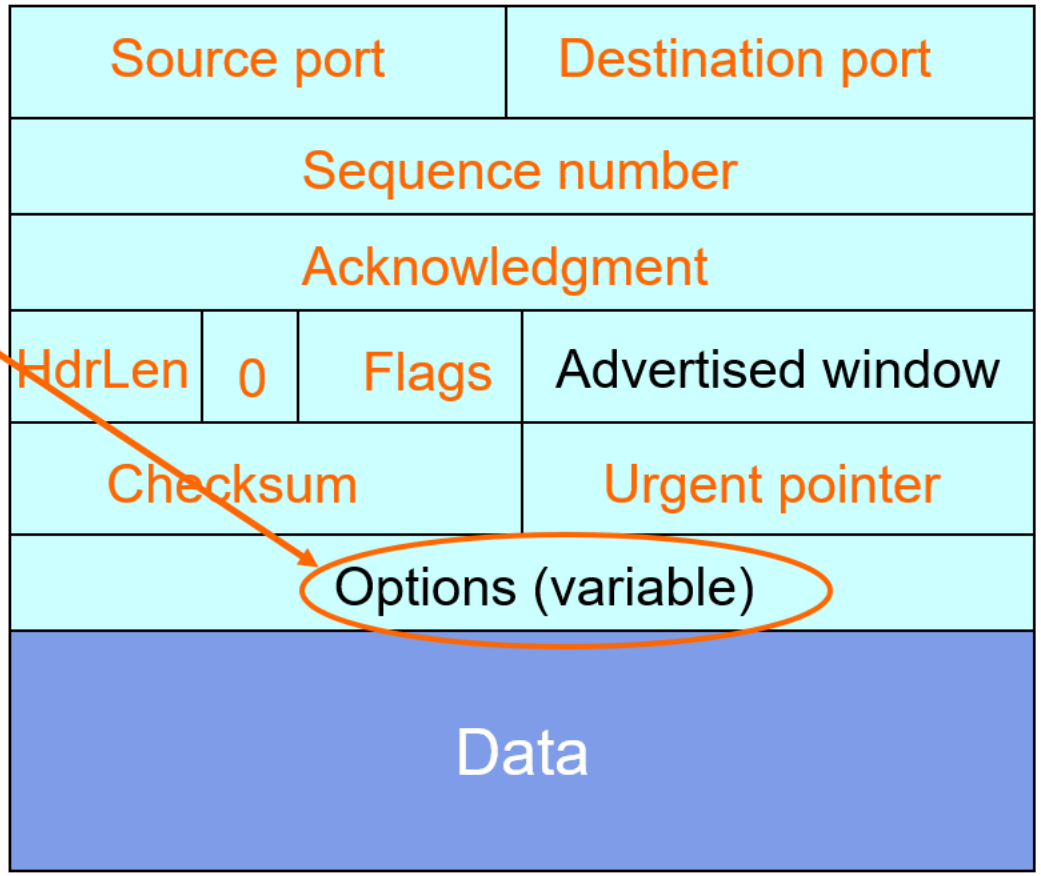


Flow Control

Avoid Buffe overrun

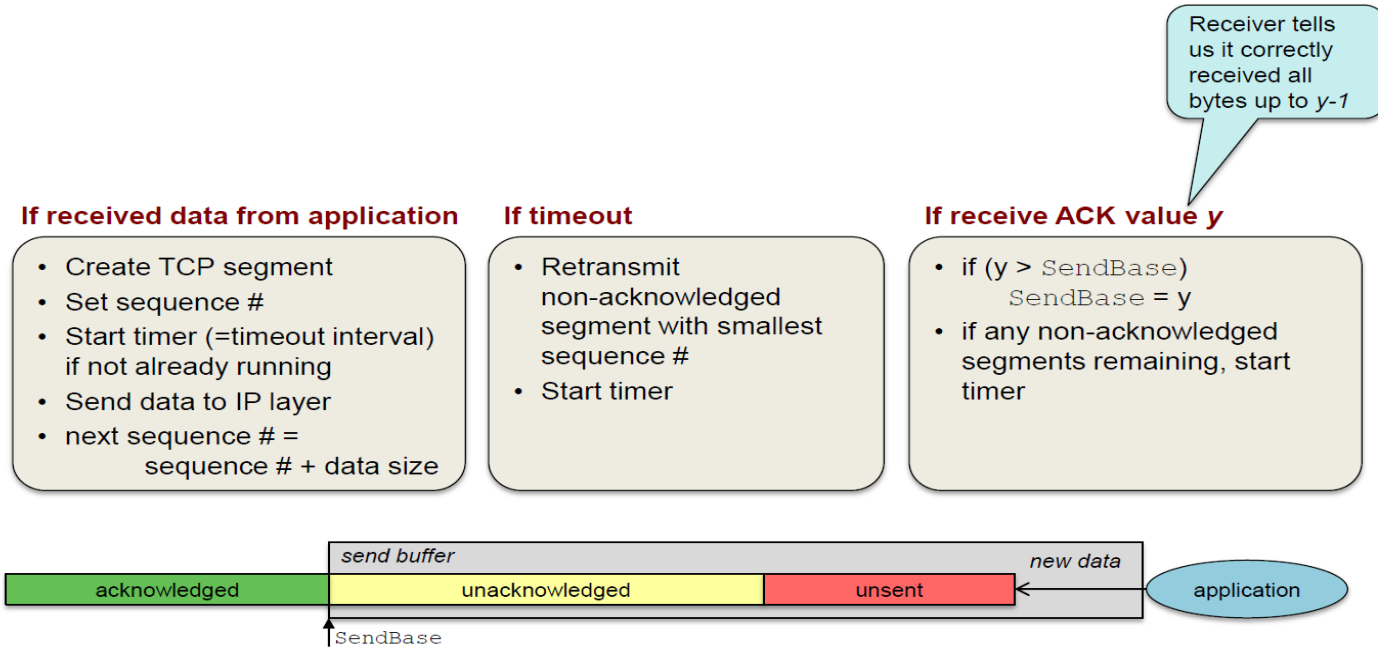
TCP HEADER

VUOTO IN MOLTI SEGMENTI. UTILIZZATO PER TIMESTAMPS, SUPPORTO FUNZIONI NON STANDARD, ETC.

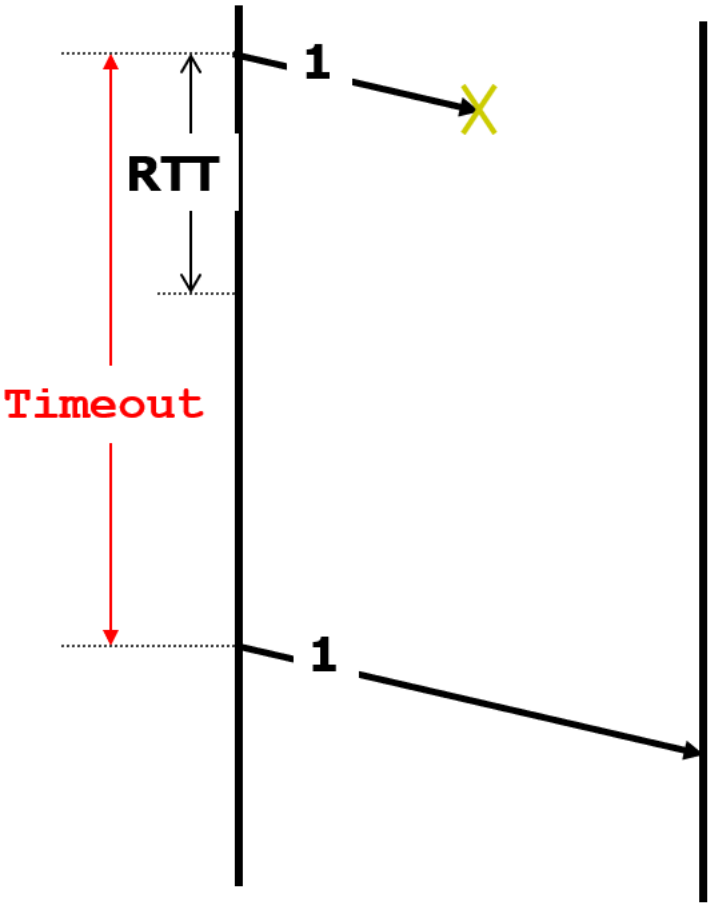


TCP TIMEOUT

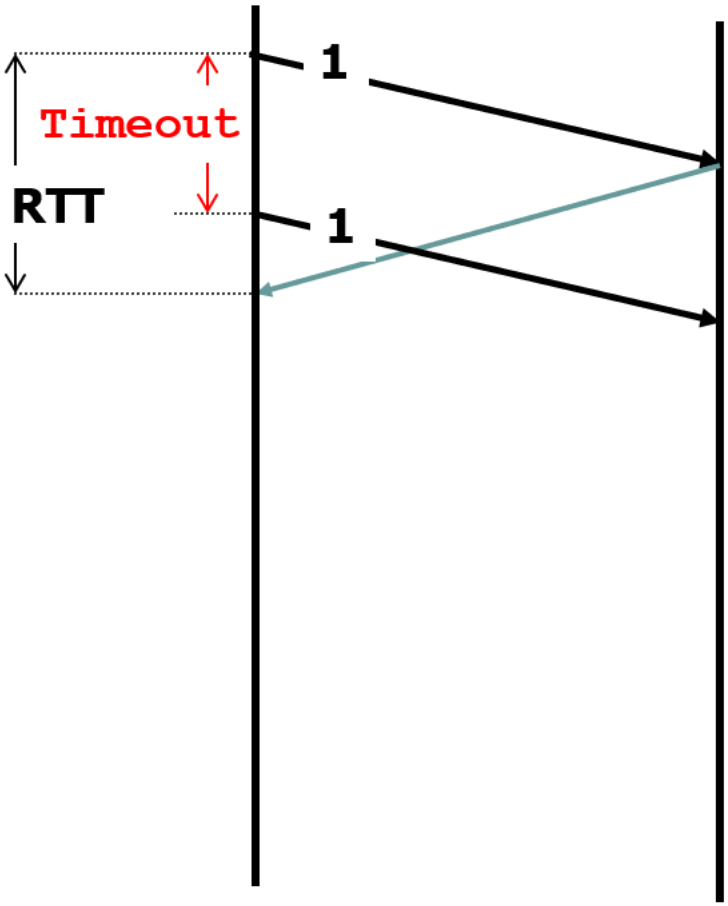
- TCP usa un singolo timer
  - anche se ci sono più segmenti trasmessi e non riscontrati
  - meno overhead di un timer per ogni segmento
- Il timer è associato al segmento più vecchio non riscontrato
  - allo scadere del timer si rispedisce solo questo segmento



TCP: TIMEOUT ADATTIVO



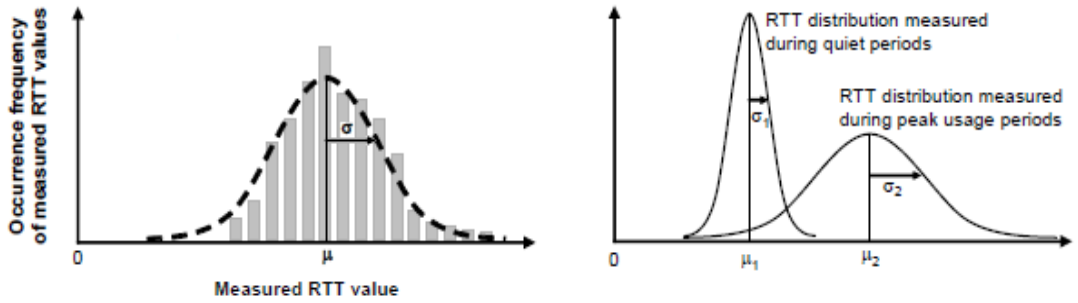
Timeout too long → inefficient



Timeout too short → duplicate packets

TCP: RETRASMISSION TIMEOUT

- calcolo del retrasmission time out (RTO)
  - relativamente semplice per reti single hop, perchè il tempo di propagazione rimane approssimativamente costante
  - più complesso in reti multi-hop
    - il delay nei routers
    - ritardi di propagazione variabili su cammini alternativi
- algoritmo usato da TCP
  - valutazione dinamica del valore di RTO [RFC-2988]
    - posto inizialmente a 3 secondi
  - aggiornato di seguito in base al Round Trip Time (RTT) calcolato su una certa connessione



TCP: CALCOLO DI UN RTT

- Round Trip Time
  - tempo trascorso tra l'invio del pacchetto e la ricezione dell'ack
- SampleRTT:
  - RTT calcolato per un segmento per cui si è ricevuto l'ACK prima della ritrasmissione del timer
  - calcolato nel seguente modo:
    - include il timestamp di ogni segmento nel campo Opzioni dell'header TCP
    - il destinatario rimanda indietro il valore del timestamp al mittente
    - il mittente calcola la differenza tra il tempo corrente ed il timestamp ricevuto

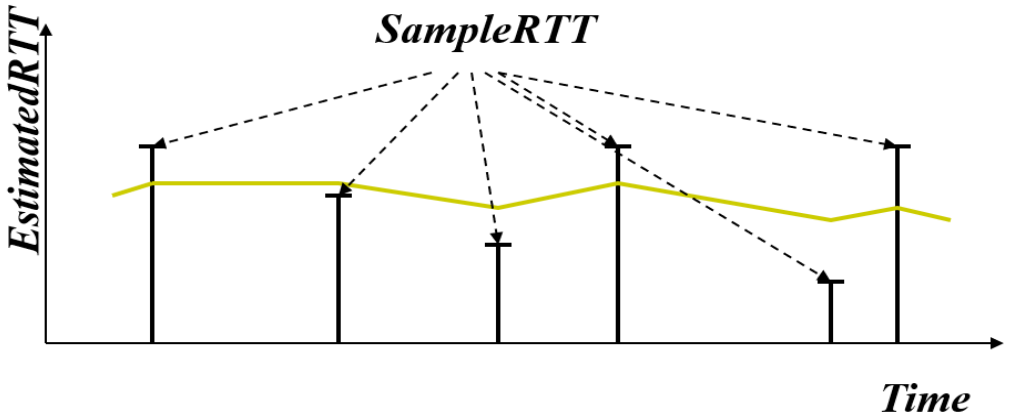
TCP: STIMA DI RTT

- il valore di RTT può fluttuare. Quale RTT?
- SRTT = "Smoothed Round Trip Time" = media pesata
- exponential weighted moving average (EWMA)
  - estimatedRTT(0) = SampleRTT(0)
  - α=0.875, maggiore importanza alle valutazioni passate

$$SampleRTT = AckRcvdTime - SendPacketTime$$

$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$

$$0 < \alpha \leq 1$$



STIMA DI RTO

- Timeout ≥ EstimatedRTT
  - altrimenti timeout precoce e occorre ritrasmettere troppo spesso
  - ma non vogliamo un valore troppo alto: ritardo eccessivo di ritrasmissione
- RTO = EstimatedRTT + x
  - x dovrebbe essere grande quando c'è molta variazione nel RTT
  - dovrebbe essere piccolo quando c'è piccola variazione del RTT
- A questo scopo occorre analizzare DevRTT
- RTO (TimeoutInterval) impostato al seguente valore:

$$TimeoutInterval(t) = EstimatedRTT(t) + 4 \times DevRTT(t)$$

con

  - EstimatedRTT(t) valore medio stimato del round-trip time al tempo t
  - DevRTT(t) varianza del round-trip time stimata al tempo t
- Valore iniziale di un secondo

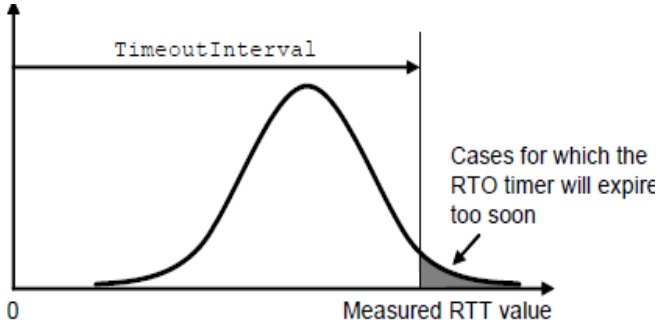
CALCOLO DI RTO

- In modo simile, la standard deviation di RTT, DevRTT(t), è stimata come

$$DevRTT(t) = (1-\beta) \cdot DevRTT(t-1) + \beta \cdot |SampleRTT(t) - EstimatedRTT(t-1)|$$

  - valore raccomandato $\beta = 0.25$.
  - valore iniziale DevRTT(0) = ½ SampleRTT(0)

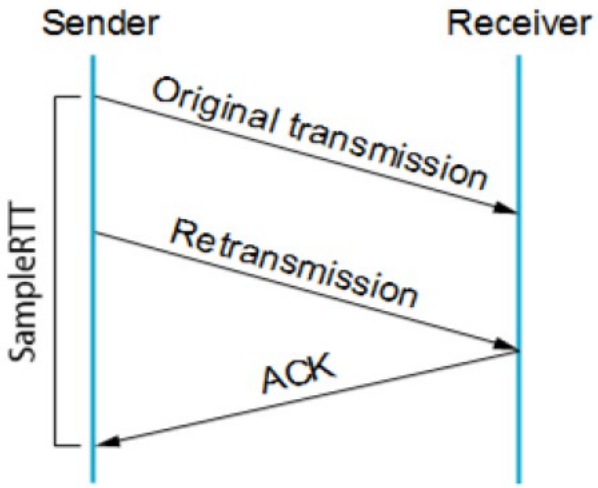
- fornisce una stima di quanto l'RTT misurato differisce da quello stimato



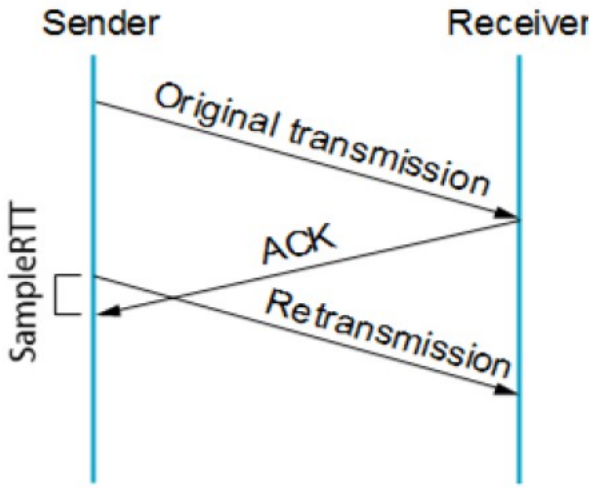
- impostando TimeoutInterval = $\mu + 4\sigma$ alta probabilità di impostare un time out opportuno
- … ma, come mostrato ci sarà sempre qualche caso per cui TimeoutInterval è troppo breve ed ACK arriva dopo il timeout

PRBLEMI NELLA STIMA DI RTO

- nel caso di ritrasmissione di segmenti, l'ACK può essere ambiguo
- problema:
  - se il mittente ha ritrasmesso un segmento, l'ACK per quel segmento può corrispondere alla trasmissione originale oppure alla ritrasmissione
- il calcolo di SampleRTT può essere errato



Calculated RTT is much greater than actual RTT



Calculated RTT is much lesser than actual RTT

ADAPTIVE RETRASMISSION: KARN PARTRIDGE

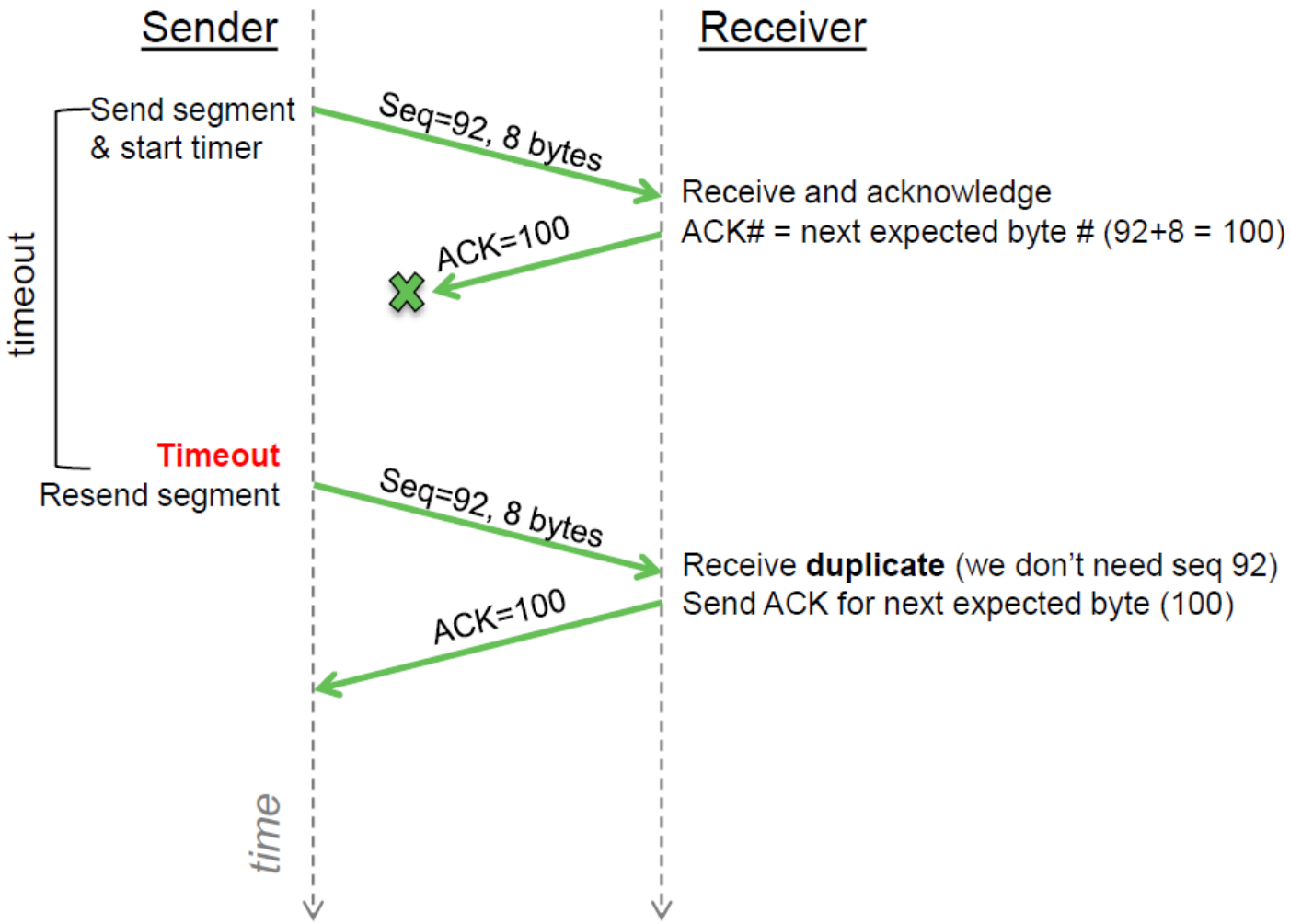
Soluzione semplice: ogni volta che un segmento è ritrasmesso (timeout scaduto):

- non si calcola la media pesata di quel SampleRTT con il valore stimato
- exponential backoff (come Ethernet)
  - quando il valore del timer si esaurisce, il valore dell'intervallo di timeout viene raddoppiato

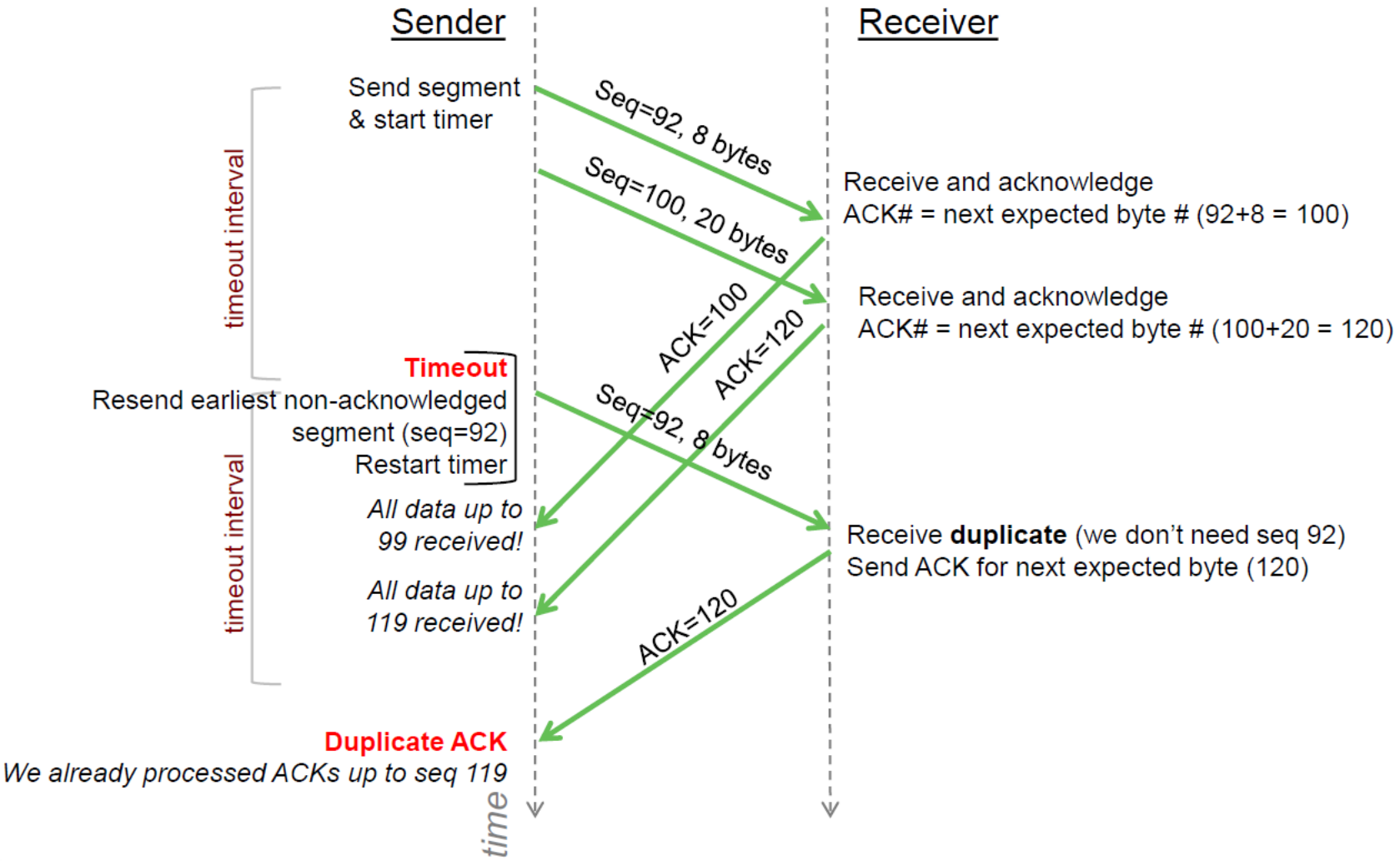
    $$\text{backoff}(t) = 2 \times \text{backoff}(t-1)$$

    $$\text{RTO timer} \leftarrow \text{backoff}(t) \times \text{TimeoutInterval}(t)$$
  - intuizione:
    - il time-out scade quando I pacchetti oppure gli ACK vengono perso
    - questo avviene generalmente se la rete è congestionata
    - meglio un valore conservativo di RTO per evitare di esacerbare la congestione, a causa di ritrasmissioni spurie
- quando il segmento viene riscontrato senza ritrasmissione, si rivaluta il valore di RTT

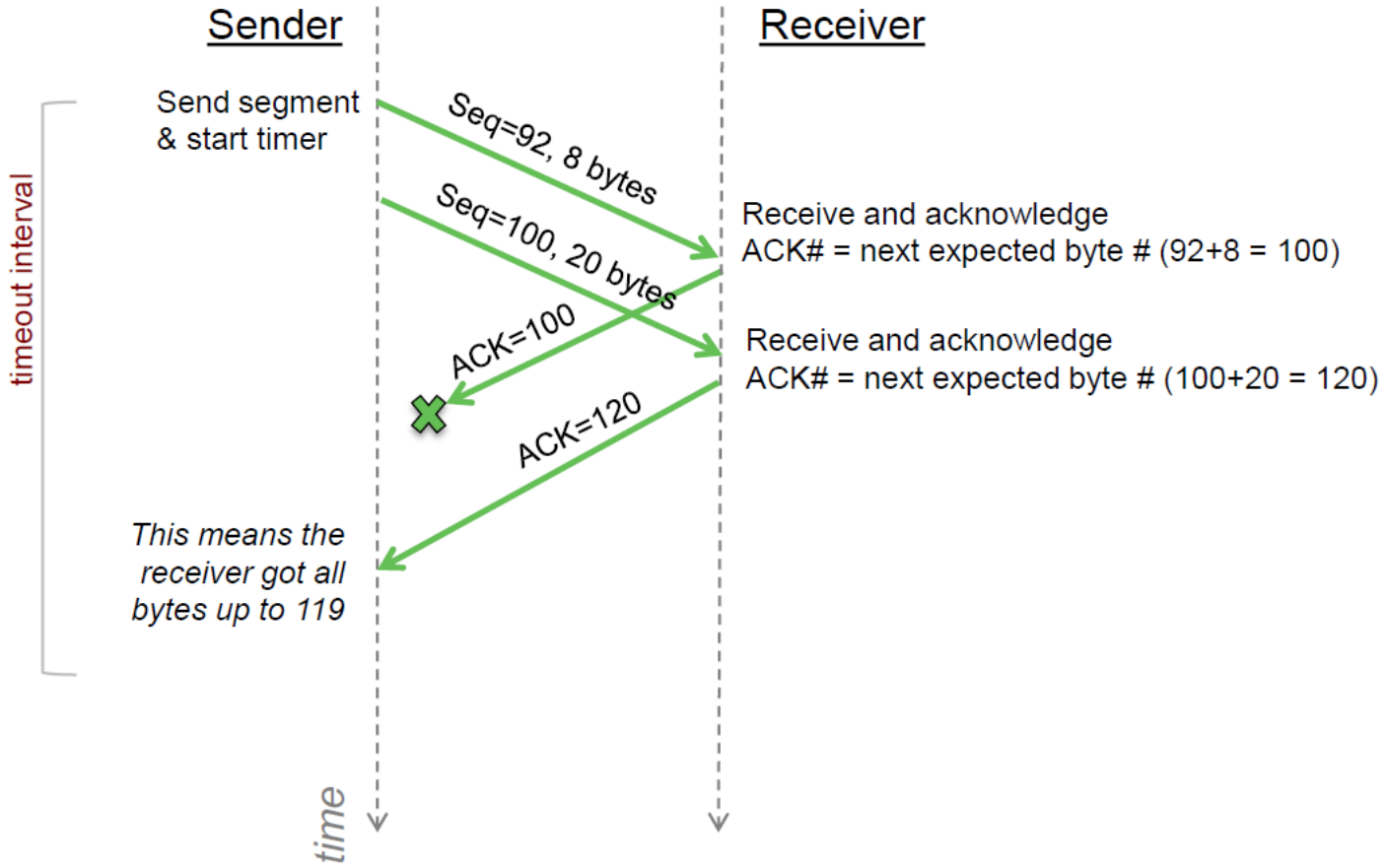
TIMEOUT: PERDITA DI ACK



TIMEOUT: DELAYED ACK



ACK CUMULATIVI



TCP: GO-BACK-N O SELECTIVE REPEAT?

- TCP simile al protocollo Go-Back-N (GBN)
  - il mittente tiene traccia della più sequenza di segmenti trasmessi, ma non riscontrata

- ma con delle differenze...
  - GBN ritrasmette tutti i segmenti nella finestra al momento del time out
    - TCP ritrasmette solo un segmento non riscontrato, quello con il numero di sequenza minore
    - TCP non ritrasmette alcun segmento se riceve un riscontro per segmenti maggiori prima di un time out
  - nella maggior arte delle implementazioni, il TCP memorizza in un buffer i segmenti fuori sequenza

SAK: SELECTIVE ACKNOLEDGMENT

- miglioramento rispetto a TCP standard con l'introduzione di un protocollo Selective Repeat
- RFC 2018: TCP Selective Acknowledgement Options
- quando il ricevente riceve un segmento fuori ordine:
  - invia ACK duplicati (come nella versione standard del protocollo)
  - ma inserisce nel campo TCP options un campo contenente i range dei dati ricevuti
    - lista di coppie (start byte, end byte)
- negoziazione tra gli end host all'inizio di una connessione
  - usato solo se entrambi devono supportare SACK

TCP/UDP: CONFRONTO

Characteristic / Description	UDP	TCP
General Description	Simple, high-speed, low-functionality "wrapper" that interfaces applications to the network layer and does little else.	Full-featured protocol that allows applications to send data reliably without worrying about network layer issues.
Protocol Connection Setup	Connectionless; data is sent without setup.	Connection-oriented; connection must be established prior to transmission.
Data Interface To Application	Message-based; data is sent in discrete packages by the application.	Stream-based; data is sent by the application with no particular structure.
Reliability and Acknowledgments	Unreliable, best-effort delivery without acknowledgments.	Reliable delivery of messages; all data is acknowledged.
Retransmissions	Not performed. Application must detect lost data and retransmit if needed.	Delivery of all data is managed, and lost data is retransmitted automatically.
Features Provided to Manage Flow of Data	None	Flow control using sliding windows; window size adjustment heuristics; congestion avoidance algorithms.
Overhead	Very low	Low, but higher than UDP
Transmission Speed	Very high	High, but not as high as UDP
Data Quantity Suitability	Small to moderate amounts of data (up to a few hundred bytes)	Small to very large amounts of data (up to gigabytes)
Types of Applications That Use The Protocol	Applications where data delivery speed matters more than completeness, where small amounts of data are sent; or where multicast/broadcast are used.	Most protocols and applications sending data that must be received reliably, including most file and message transfer protocols.
Well-Known Applications and Protocols	Multimedia applications, DNS, BOOTP, DHCP, TFTP, SNMP, RIP, NFS (early versions)	FTP, Telnet, SMTP, DNS, HTTP, POP, NNTP, IMAP, BGP, IRC, NFS (later versions)