



**La Sapienza**

Università degli Studi di Roma

Dipartimento di Informatica e Sistemistica

# RETI DI CALCOLATORI II

## TCP\_UDP recap

**Emiliano Trevisani**

**[trevisani@dis.uniroma1.it](mailto:trevisani@dis.uniroma1.it)**

**A.A. 2008/2009**

# Data Link Versus Transport

- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT -> what is RTT?
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

# Servizio di trasporto

- Underlying best-effort network (Internet)
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end service requirements
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

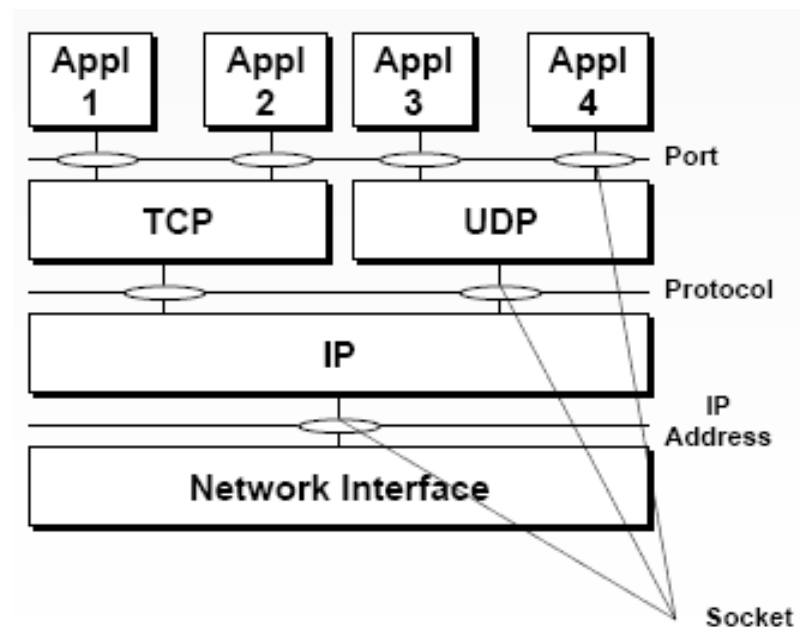
# Servizio di trasporto

- Lo strato di trasporto fornisce un servizio di trasferimento allo strato applicativo conforme ai requisiti di qualità richiesti dall'applicazione
- Protocolli di trasporto più diffusi:
  - User Datagram Protocol (UDP)
    - è utilizzato quando l'applicazione non richiede funzioni di controllo di flusso e controllo d'errore
  - Transport Control Protocol (TCP)
    - è utilizzato per applicazioni che generano flussi informativi di una certa complessità che richiedono funzioni di controllo d'errore e di flusso
  - Real-Time Transport Protocol (RTP)
    - è utilizzato quando l'applicazione è di tipo real-time (es. voce, video)
    - è supportato dal protocollo UDP

# Servizio di trasporto - indirizzamento

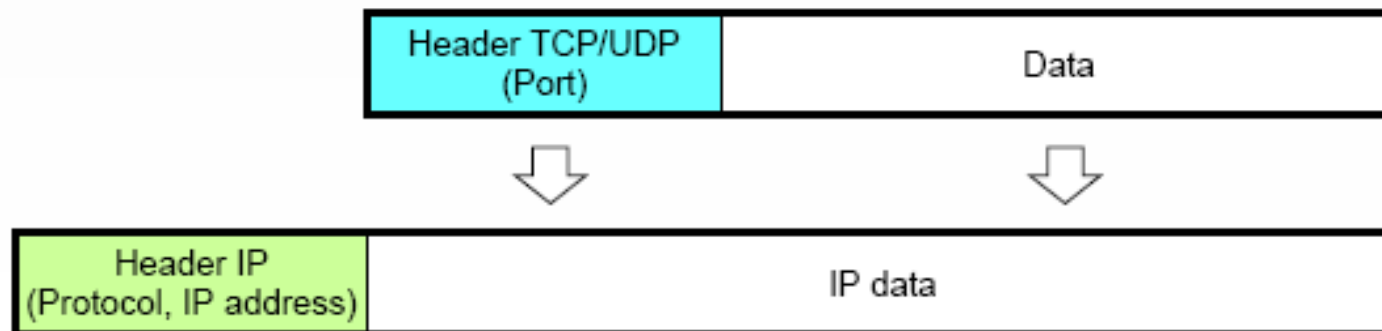
- Distingue tra i diversi processi applicativi ciascuno utente dello stesso servizio di trasporto

- Port
  - identifica un utente dello strato di trasporto
  - è rappresentato da un intero (16 bit)
- Socket
  - identifica l'interfaccia tra l'applicazione ed i protocolli di comunicazione
  - è rappresentata dalla terna (port; protocol; IP\_Address)



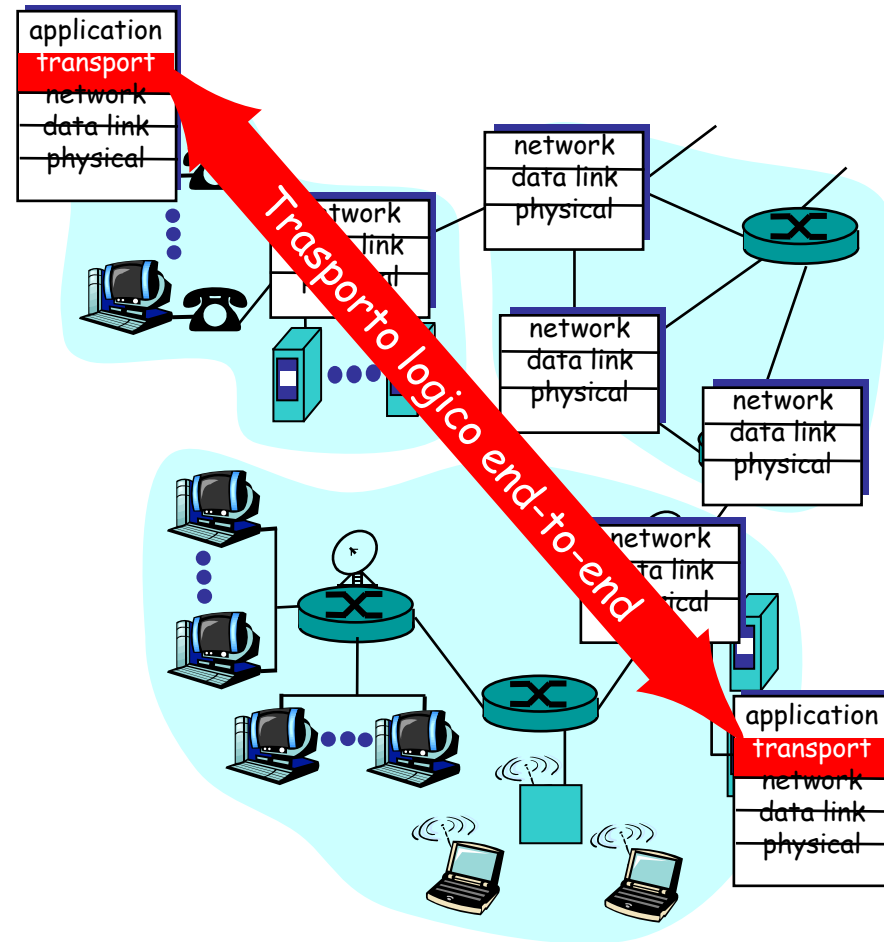
# Servizio di trasporto - indirizzamento

- La componente "Port" è contenuta nell'intestazione dell'unità dati di TCP/UDP
- Le componenti "Protocol" e "IP\_Address" sono contenute nell'intestazione dell'unità dati di IP (funzione di moltiplicazione)



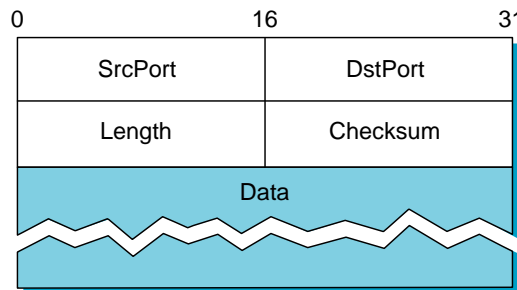
# Protocolli di trasporto in Internet

- Trasporto affidabile in ordine uno a uno (unicast): TCP
  - Congestione
  - Controllo di flusso
  - Instaurazione di connessione
- Trasporto non affidabile e non in ordine uno a uno o uno a molti (multicast): UDP
- Servizi non disponibili:
  - Real-time
  - Garanzia di banda



# Simple Demultiplexer (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see `/etc/services` on Unix
- Header format



- Optional checksum
  - pseudo header [IP header fields: source, dest, protocol, datagram length] + UDP header + data

# User Datagram Protocol (UDP)

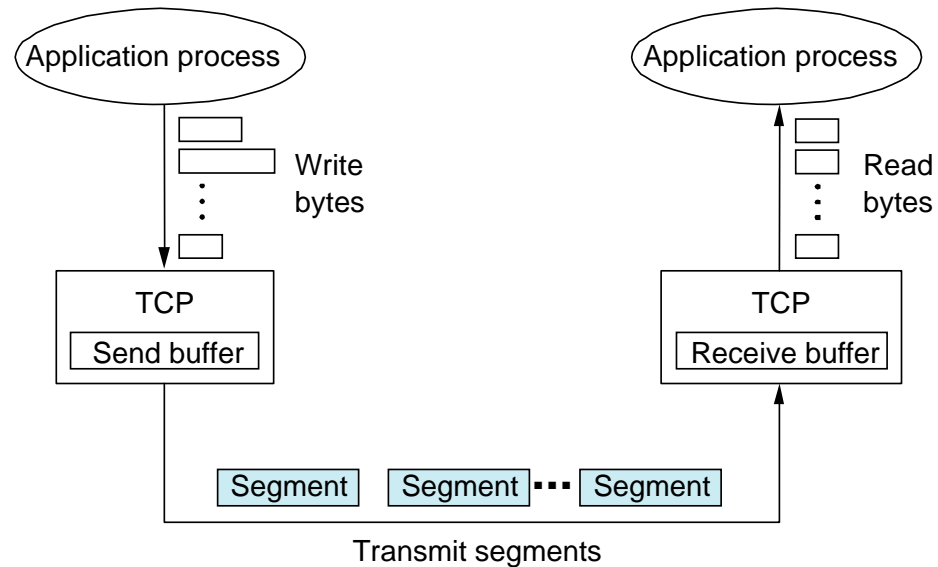
- E' un protocollo senza connessione
- Non supporta meccanismi di riscontro e di controllo d'errore
- È utilizzato per il supporto di transazioni semplici tra applicativi
  - interrogazioni di database
  - risoluzione di indirizzi [DNS]
  - messaggi di management
- Può essere usato come punto di partenza per sviluppare protocolli di trasporto “custom” dalle caratteristiche intermedie tra TCP e UDP

# TCP Overview

- Trasferisce un flusso informativo bi-direzionale non strutturato tra due host ed effettua operazioni di moltiplicazione e de-moltiplicazione
- E' un protocollo con connessione
- Funzioni eseguite
  - controllo e recupero di errore
  - controllo di flusso
  - ri-ordinamento delle unità informative
  - indirizzamento di una specifica applicazione all'interno di un host

# TCP Overview

- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes
- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



# TCP segment format

- Il TCP interpreta il flusso dati come sequenza di ottetti
- La sequenza di ottetti è suddivisa in **segmenti**

0	4	8	16	24	31
Source Port			Destination Port		
Sequence Number					
Acknowledgment Number					
HLEN	Reserved	Code bits	Adv. Window		
Checksum			Urgent Pointer		
Options (if any)				Padding	
Data					
Data					

# TCP segment

- **Source Port (16 bit) e Destination Port (16 bit)**
  - identificano i processi sorgente e destinazione dei dati
- **Sequence Number (32 bit)**
  - numero di sequenza in trasmissione
  - contiene il numero di sequenza del primo byte di dati contenuti nel segmento a partire dall'inizio della sessione TCP
- **Acknowledgement Number (32 bit)**
  - numero di sequenza in ricezione
  - se ACK=1, contiene il numero di sequenza del prossimo byte che il trasmettitore del segmento si aspetta di ricevere
  - è possibile la modalità piggybacking di riscontro
- **Obs: Sequence & ACK number both refer to byte sequence and not to segment sequence**

# TCP segment

- **HLEN (4 bit)**
  - contiene il numero di parole di 32 bit contenute nell'intestazione TCP
  - l'intestazione TCP non supera i 60 byte ed è sempre un multiplo di 32
- **Reserved (6 bit)**
  - riservato per usi futuri, per ora contiene degli zeri
- **Adv. Window (16 bit)**
  - larghezza della finestra in byte (controllo di flusso è orientato al byte)
  - è il numero di byte che, ad iniziare dal valore del campo Ack Number, il trasmettitore del segmento è in grado di ricevere
- **Checksum (16 bit)**
  - protegge l'intero segmento più alcuni campi dell'header IP (pseudo header, vedi UDP)

# TCP segment

- **Control bit (6 bit)**
  - URG
    - è uguale a 1 quando il campo urgent pointer contiene un valore significativo
  - ACK
    - è uguale a 1 quando il campo Ack Number contiene un valore significativo
  - PSH
    - è uguale a 1 se i dati devono essere consegnati all'applicazione ricevente prescindendo dal riempimento dei buffer di ricezione
  - RST
    - è uguale a 1 in caso di richiesta di reset della connessione
  - SYN
    - è uguale a 1 solo nel primo segmento inviato durante la fase di sincronizzazione fra le entità TCP [instaurazione della connessione]
  - FIN
    - è uguale a 1 quando la sorgente ha esaurito i dati da trasmettere

# TCP segment

- **Urgent Pointer (16 bit)**
  - contiene il numero di sequenza dell'ultimo byte dei dati che devono essere consegnati urgentemente al processo ricevente
  - tipicamente sono messaggi di controllo (out-of-band traffic)
- **Options (di lunghezza variabile)**
  - sono presenti solo raramente
  - Esempi:
    - End of Option List, No-operation, Maximum Segment Size (MSS)
- **Padding (di lunghezza variabile)**
  - impone che l'intestazione abbia una lunghezza multipla di 32 bit

# Indirizzamento TCP

- Il numero di porta può essere
  - statico (Well Known port)
    - sono identificativi staticamente associati ad applicazioni largamente utilizzate
    - sono utilizzati identificativi inferiori a 256
  - dinamico (Ephemeral)
    - sono identificativi assegnati direttamente dal sistema operativo al momento dell'apertura della connessione
    - si utilizzano valori maggiori di 1023

Numero	Applicazione	Numero	Applicazione
7	Echo	37	Time
21	FTP (File Transfer Protocol)	53	Domain Name Server
23	TELNET	103	X400 Mail Service
25	SMTP (Simple Mail Transport Protocol)	119	NNTP (USENET New Transfer Prot.)

# Trasferimento affidabile

- Tutti i pacchetti spediti giungono a destinazione in ordine
- Rete sottostante inaffidabile
- Strumenti
  - Numeri di sequenza / Riscontri
  - Connessione
  - Timer
  - Ritrasmissioni
  - Maximum Segment Lifetime

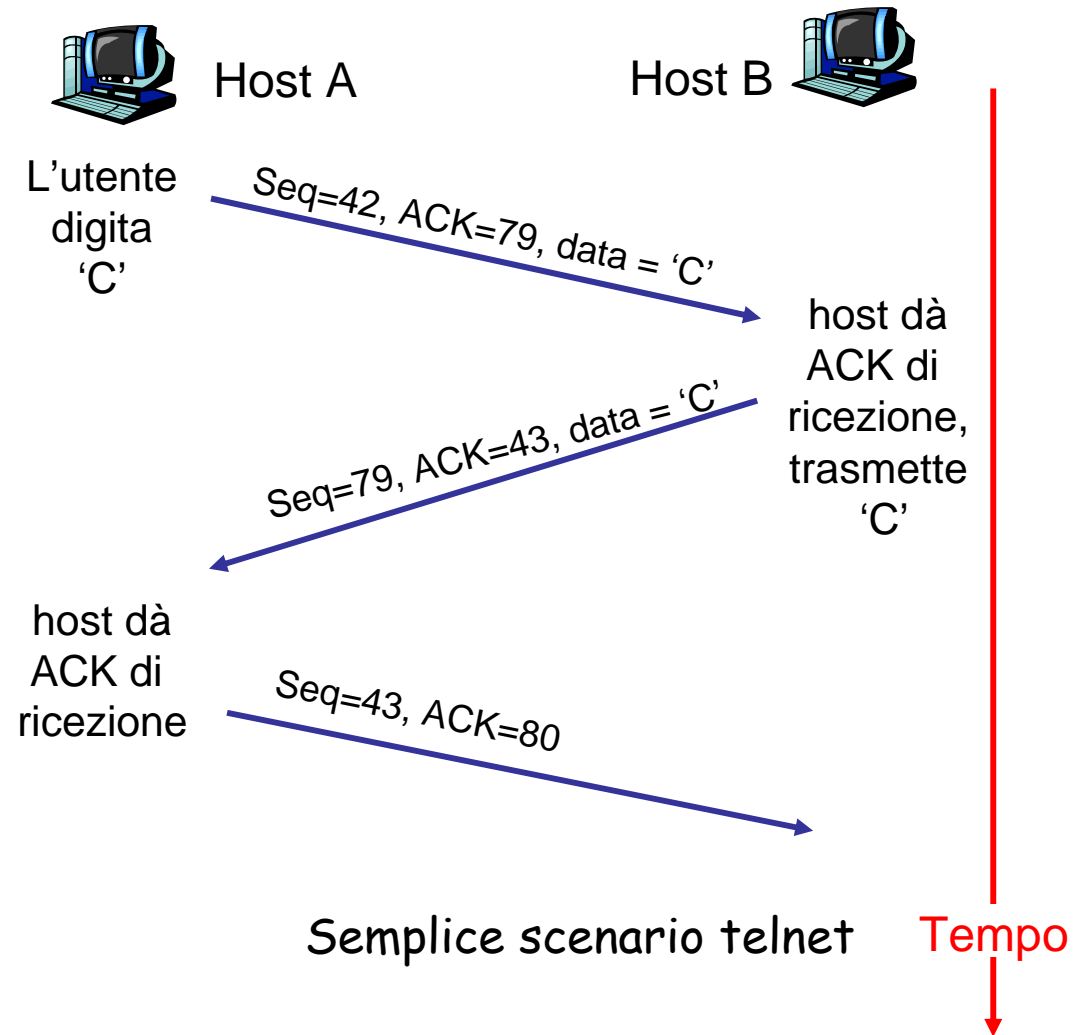
# # seq. e ACK in TCP

## Numeri di sequenza:

- Numero del primo byte presente nel segmento

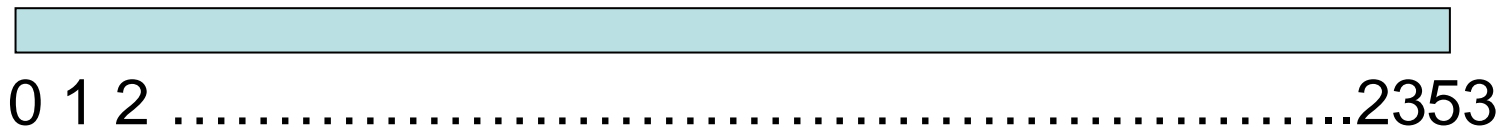
## ACK:

- # seq del prossimo byte atteso dal lato remoto
- ACK cumulativi
- **D.:** come il ricevente tratta segmenti fuori ordine
- **R.:** TCP non specifica, dipende dall'implementazione



# Numeri di sequenza/cont.

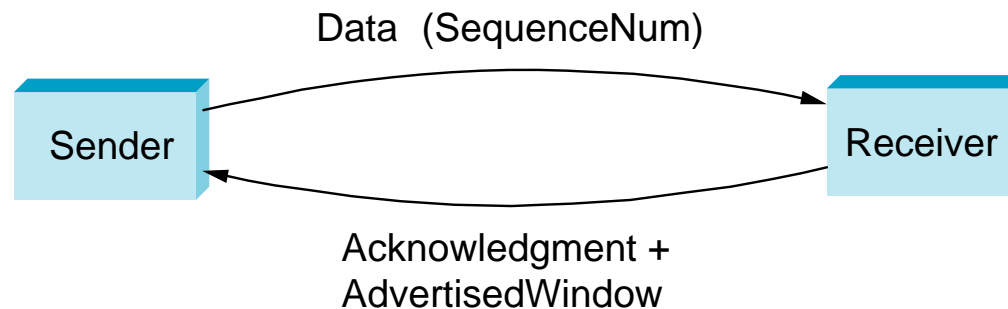
- File da trasmettere di 2354 byte
- Maximum Segment Size 500
- Numero di sequenza iniziale 181



Sono spediti 5 pacchetti con numeri di sequenza 181, 681, 1181, 1681 e 2181 rispettivamente

# TCP connection

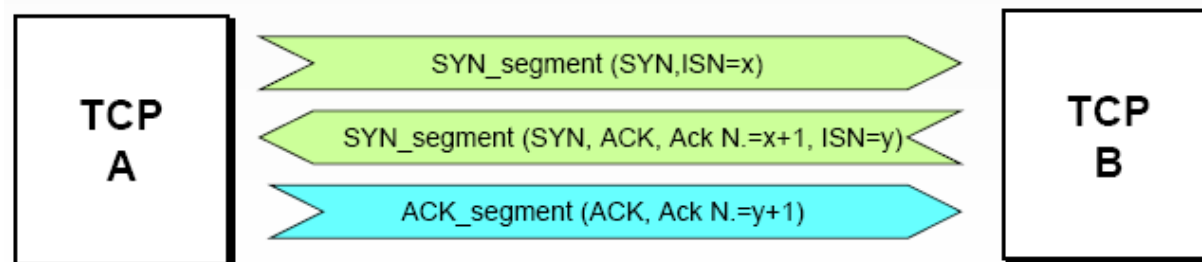
- Each connection identified by 4-tuple:
  - (SrcPort, SrcIPAddr, DstPort, DstIPAddr)
- Sliding window + flow control
  - acknowledgment, SequenceNum, AdvertisedWindow



- Numero di sequenza: numero del primo byte presente nel segmento
- ACK:
  - # seq del prossimo byte atteso dal lato remoto
  - ACK cumulativi

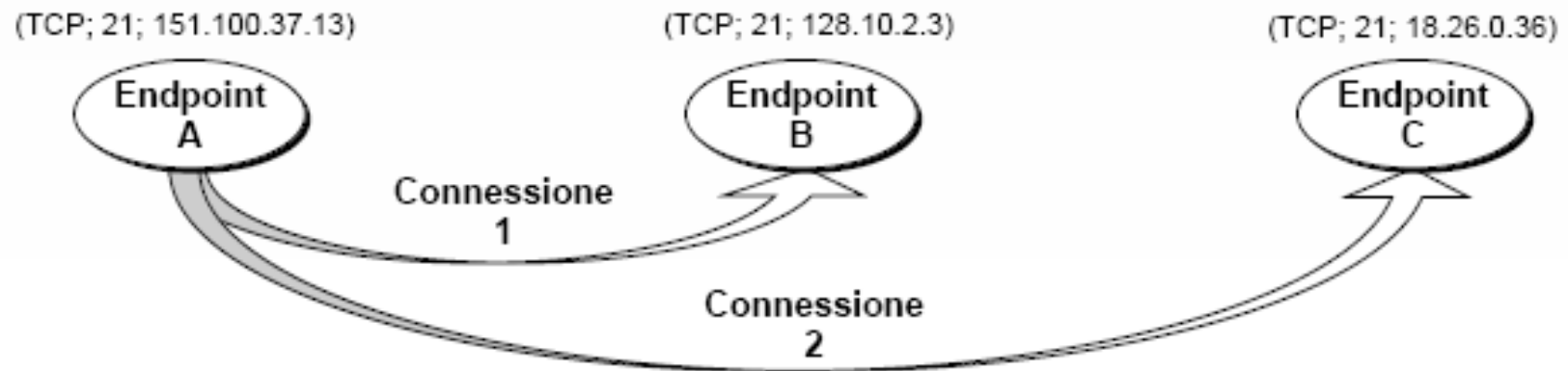
# TCP connection

- Durante la fase di instaurazione della connessione, le due entità TCP remote si sincronizzano scambiandosi il proprio numero di sequenza iniziale, che rappresenta il numero a partire dal quale tutti i byte trasmessi saranno numerati in sequenza
- La sincronizzazione è necessaria per risolvere potenziali situazioni anomale dovute alla non affidabilità del protocollo IP
- La sincronizzazione avviene con un meccanismo detto “**three way handshaking**”
- Nella fase di rilascio le due vie sono chiuse indipendentemente



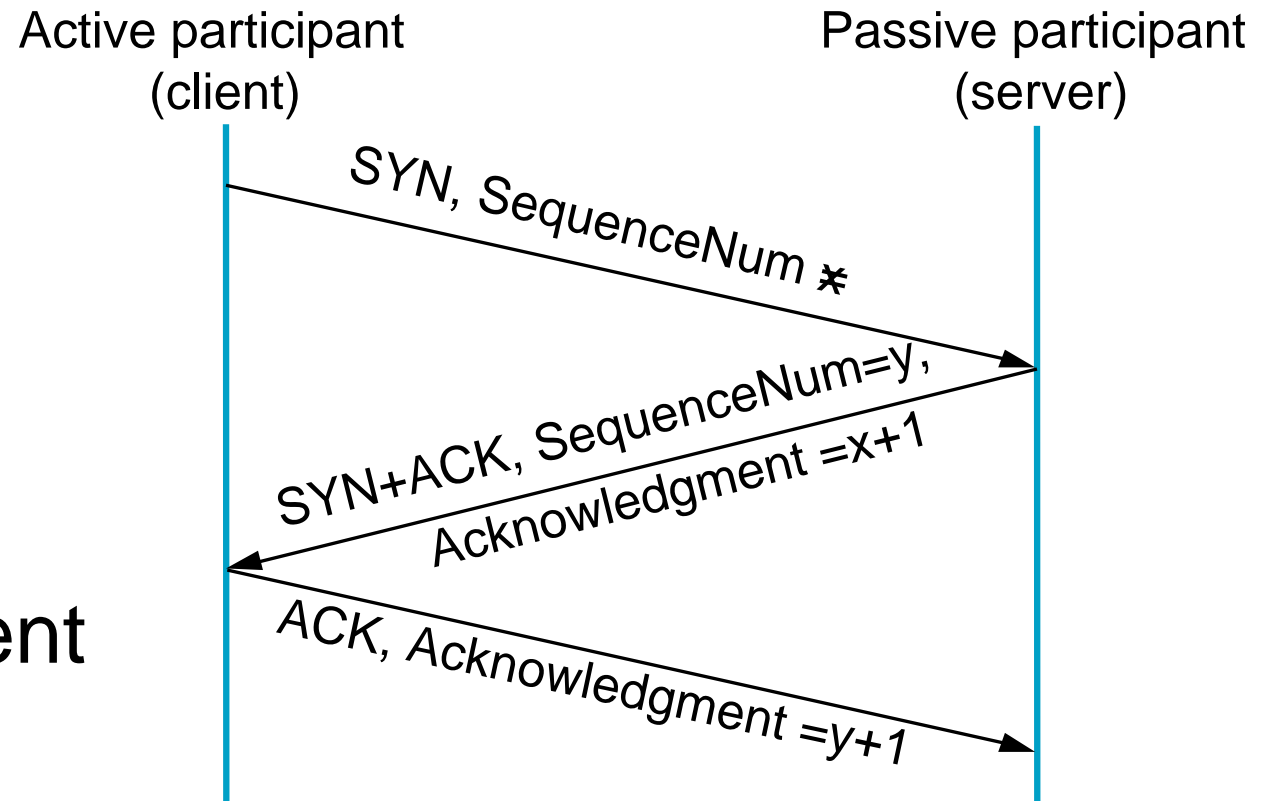
# TCP connection

- Una connessione TCP è identificata dalla coppia di socket associati agli endpoint tra i quali vengono scambiate informazioni
- Un endpoint può essere impegnato allo stesso tempo in più connessioni TCP

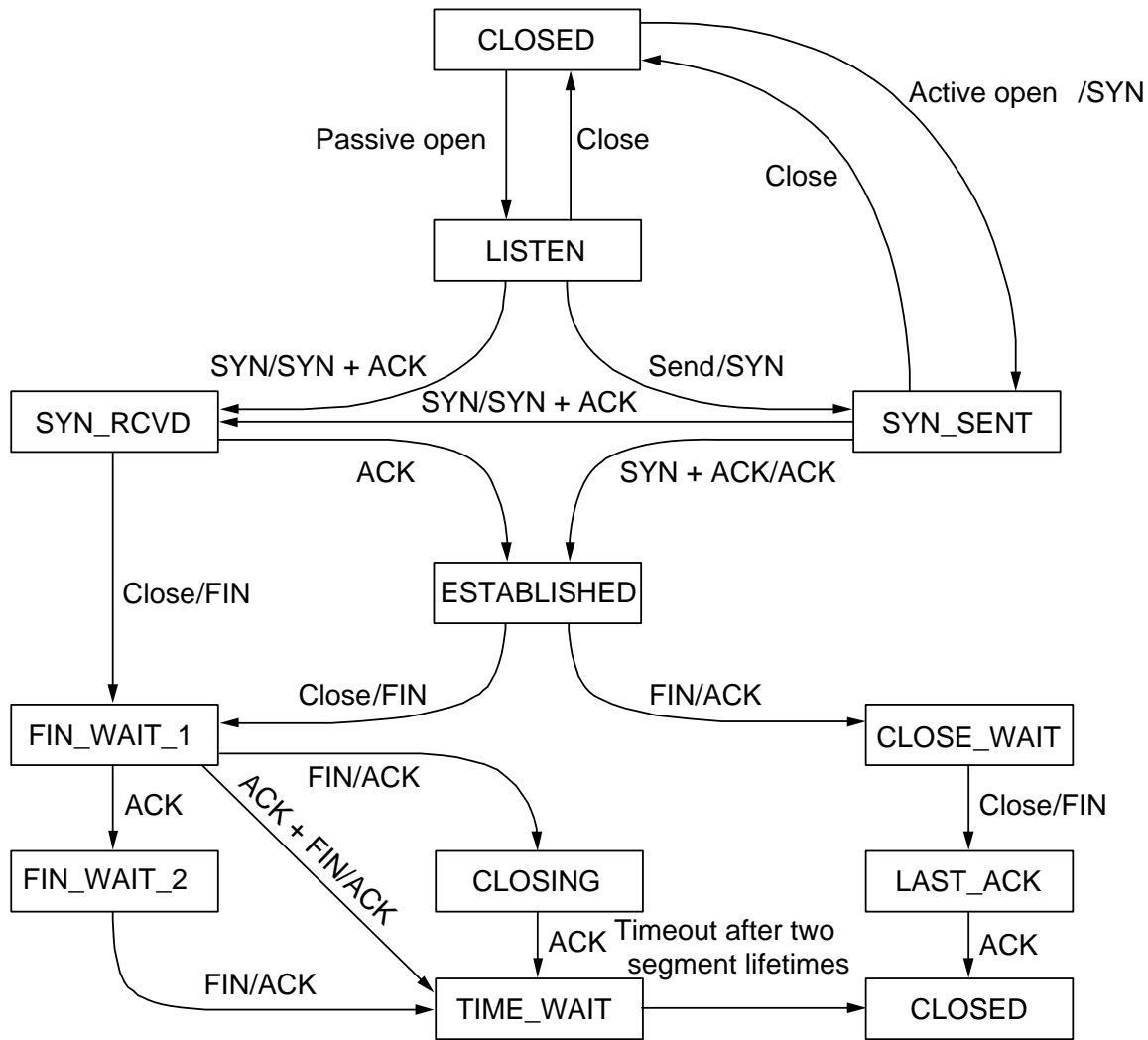


# Instaurazione connessione

- Modello Client - server

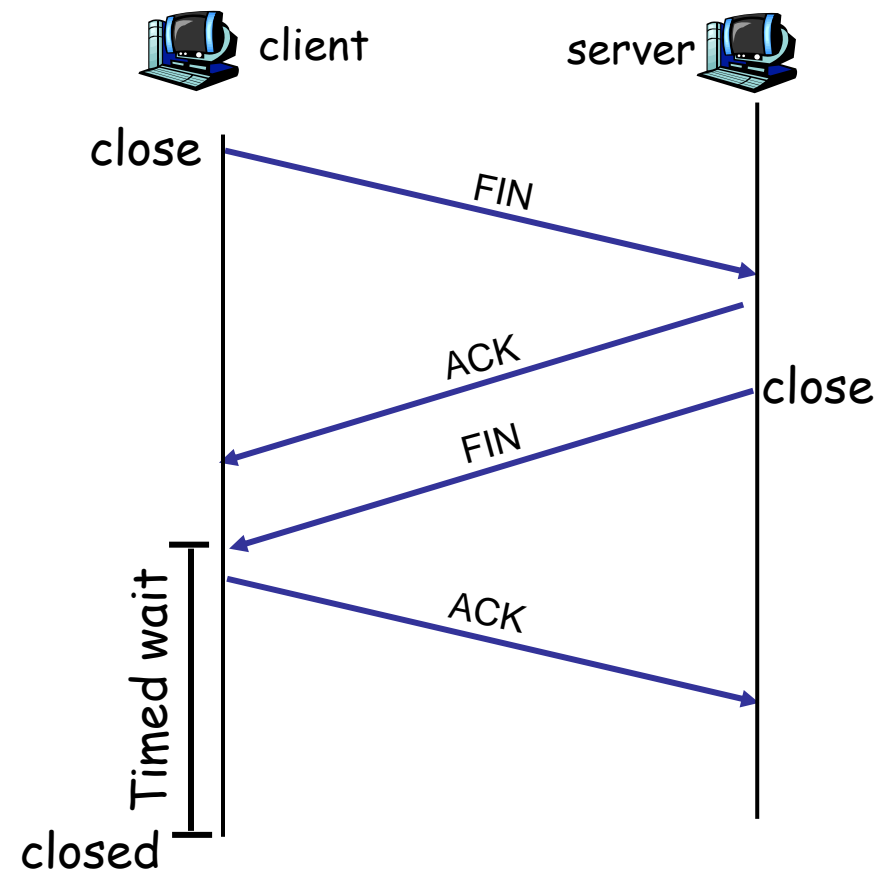


# State Transition Diagram



# Chiusura connessione - client inizia

- Passo 1: client invia il segmento di controllo TCP FIN al server
- Passo 2: server riceve FIN, risponde con ACK. Chiude la connessione, invia FIN.
- Passo 3: client riceve FIN, risponde con ACK.
  - Entra “Attesa” (“time wait”) - risponde con ACK ai segmenti FIN ricevuti
- Passo 4: server riceve ACK. Connessione chiusa



# Controllo di errore

- TCP prevede esclusivamente riscontri positivi (ACK)
- La ritrasmissione dei segmenti è innescata dalla mancata ricezione degli ACK entro un fissato tempo limite (Timeout)
- Il dimensionamento del timeout è un aspetto critico nelle prestazioni di TCP
  - se il suo valore è troppo piccolo, alcuni segmenti in ritardo a causa di congestione, potrebbero considerati persi e quindi ri-trasmessi con conseguente perdita di efficienza
  - se il suo valore è troppo grande, la risposta ad un evento di perdita sarebbe troppo lenta con conseguente perdita di efficienza

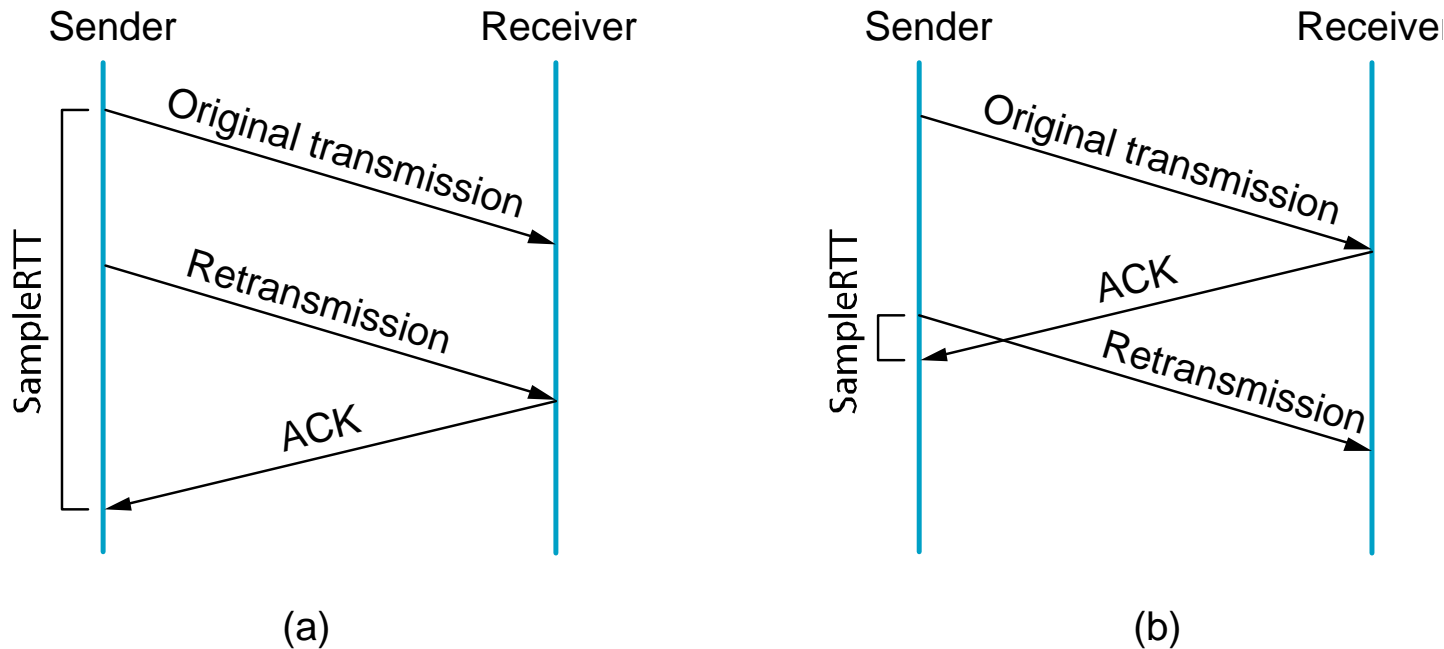
# Controllo di errore

- Il **Retransmission TimeOut (RTO)** è determinato con uno schema adattativo
- Il TCP misura dinamicamente il Round Trip Time (RTT)
  - RTT = ritardo tra l'invio di un segmento e la ricezione del relativo ACK
- Il valore di RTO è scelto maggiore del valore medio osservato del RTT
- La misura del RTT è affetta dai seguenti errori:
  - l'emissione degli ACK da parte del ricevente può essere non immediata
  - se è stata effettuata una ritrasmissione è impossibile distinguere se l'ACK si riferisce alla trasmissione iniziale o alla ritrasmissione
  - lo stato di congestione della rete può cambiare molto rapidamente

# Adaptive Retransmission (Original Algorithm)

- Measure `sampleRTT` for each segment / ACK pair
- Compute weighted average of RTT
  - $\text{EstRTT} = \alpha \times \text{EstRTT} + (1 - \alpha) \times \text{SampleRTT}$
  - $\alpha$  between 0.8 and 0.9
- Exponential Weighted Moving Average -> perche'?
- Set timeout based on `EstRTT`
  - $\text{TimeOut} = 2 \times \text{EstRTT}$

# Karn/Partridge Algorithm

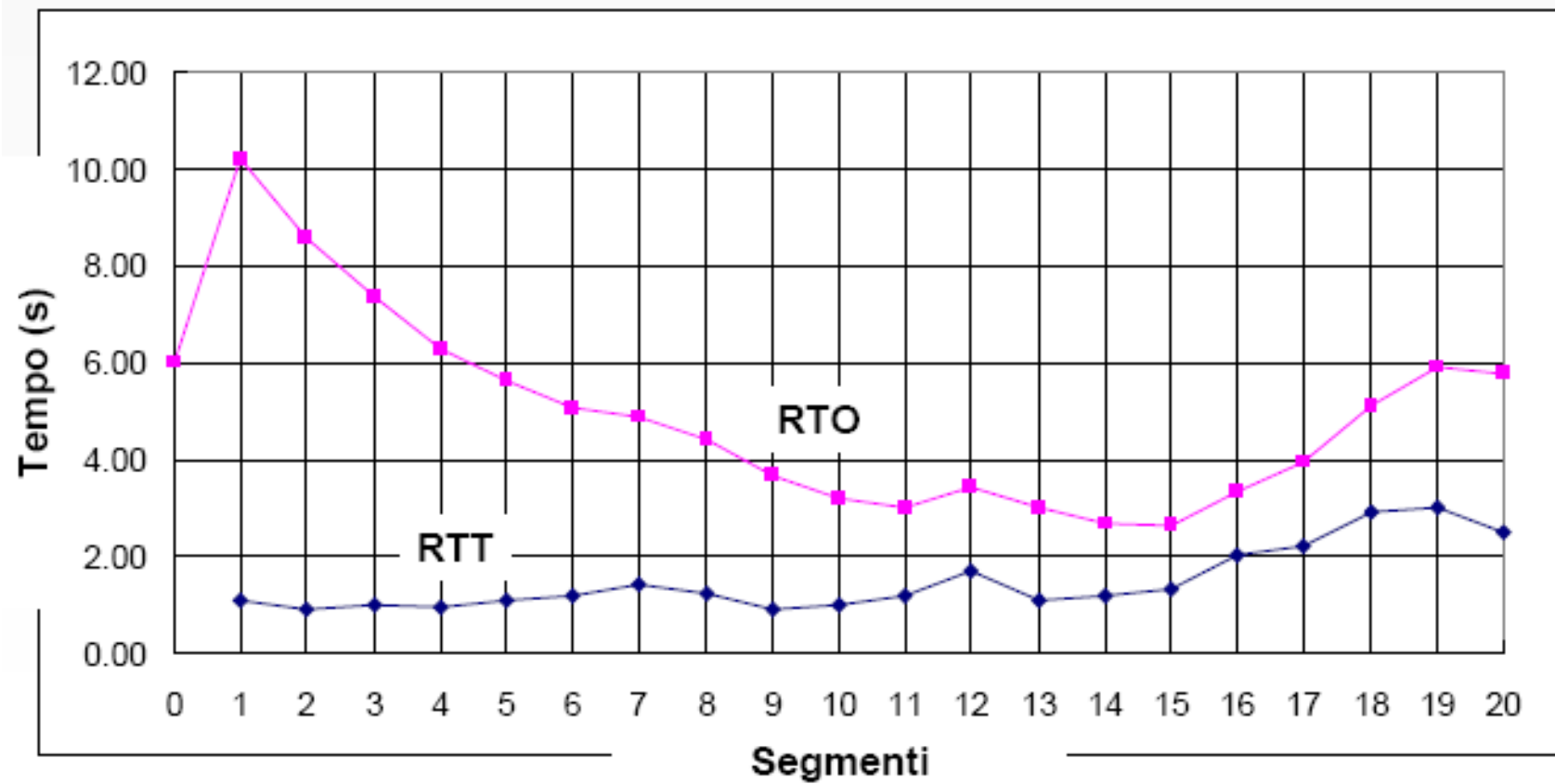


- Do not sample RTT when retransmitting
- Double timeout after each retransmission

# Jacobson/ Karels Algorithm

- New Calculations for average RTT
- $\text{Diff} = \text{SampleRTT} - \text{EstRTT}$
- $\text{EstRTT} = \text{EstRTT} + (\delta \times \text{Diff})$ 
  - Decrease se  $\text{Diff} < 0$
- $\text{Dev} = \text{Dev} + \delta(|\text{Diff}| - \text{Dev})$ 
  - where  $\delta$  is a factor between 0 and 1
- Consider variance when setting timeout value
- $\text{TimeOut} = \mu \times \text{EstRTT} + \phi \times \text{Dev}$ 
  - where  $\mu = 1$  and  $\phi = 4$
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control

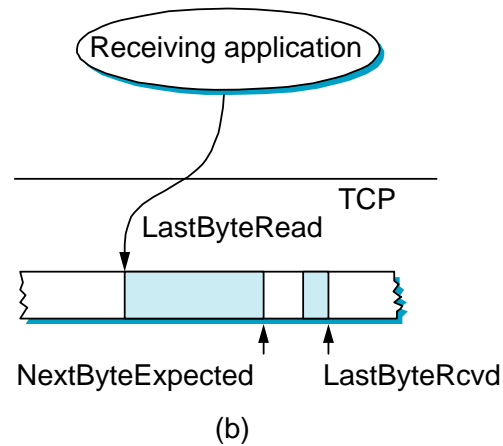
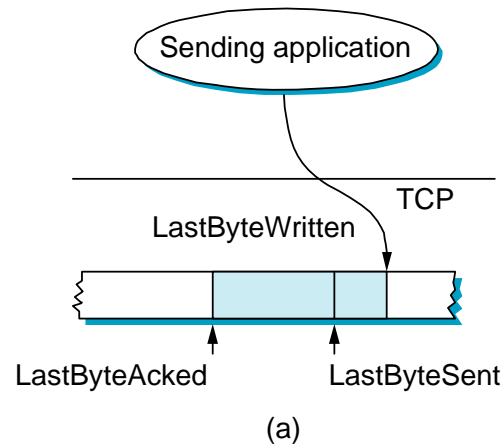
# RTO calculation example



# Controllo di Flusso

- Il controllo di flusso ha lo scopo di limitare il tasso di generazione dei dati da parte di un host
  - tale meccanismo è indispensabile in Internet dove sono presenti host di potenzialità molto diverse
- TCP utilizza un controllo di flusso a finestra basato su
  - finestra scorrevole di ampiezza variabile
  - credit allocation schema
- Il controllo di flusso opera a livello di ottetti (byte)
- Gli ottetti sono numerati sequenzialmente a partire dal numero scelto durante il 3-way handshaking
- Un riscontro (ACK Number= $X$  e Window= $W$ ) significa che
  - sono riscontrati tutti gli ottetti ricevuti fino a quello numerato con  $X-1$
  - il trasmittente è autorizzato a trasmettere fino a ulteriori  $W$  ottetti, ovvero fino all'ottetto numerato con  $X+W-1$

# Sliding Window Revisited



- Sending side

- $\text{LastByteAcked} \leq \text{LastByteSent}$
- $\text{LastByteSent} \leq \text{LastByteWritten}$
- buffer bytes between  $\text{LastByteAcked}$  and  $\text{LastByteWritten}$

- Receiving side

- $\text{LastByteRead} < \text{NextByteExpected}$
- $\text{NextByteExpected} \leq \text{LastByteRcvd} + 1$
- buffer bytes between  $\text{LastByteRead}$  and  $\text{LastByteRcvd}$

# Flow Control

- Send buffer size: `MaxSendBuffer`
- Receive buffer size: `MaxRcvBuffer`
- Receiving side
  - `LastByteRcvd - LastByteRead <= MaxRcvBuffer`
  - `AdvertisedWindow = MaxRcvBuffer - ((NextByteExpected - 1) - LastByteRead)`
- Sending side
  - `LastByteSent - LastByteAcked <= AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)`
  - `LastByteWritten - LastByteAcked <= MaxSendBuffer`
  - block sender if `(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer`
- Always send ACK in response to arriving data segment
- Persist when `AdvertisedWindow = 0`

# Controllo di congestione

- Ha lo scopo di recuperare da situazioni di sovraccarico nella rete limitando il traffico offerto alla rete stessa [approccio indiretto]
- Difficoltà:
- il protocollo IP (protocollo di rete) non possiede alcun meccanismo per rivelare e controllare la congestione
- TCP è un protocollo end-to-end e può rivelare e controllare la congestione solo in modo indiretto
- la rete non coopera con gli host per il controllo della congestione
- la conoscenza dello stato della rete da parte delle entità TCP è imperfetta a causa dei ritardi di rete
- le entità TCP che usano la rete non cooperano tra loro, anzi competono per l'uso delle risorse distribuite

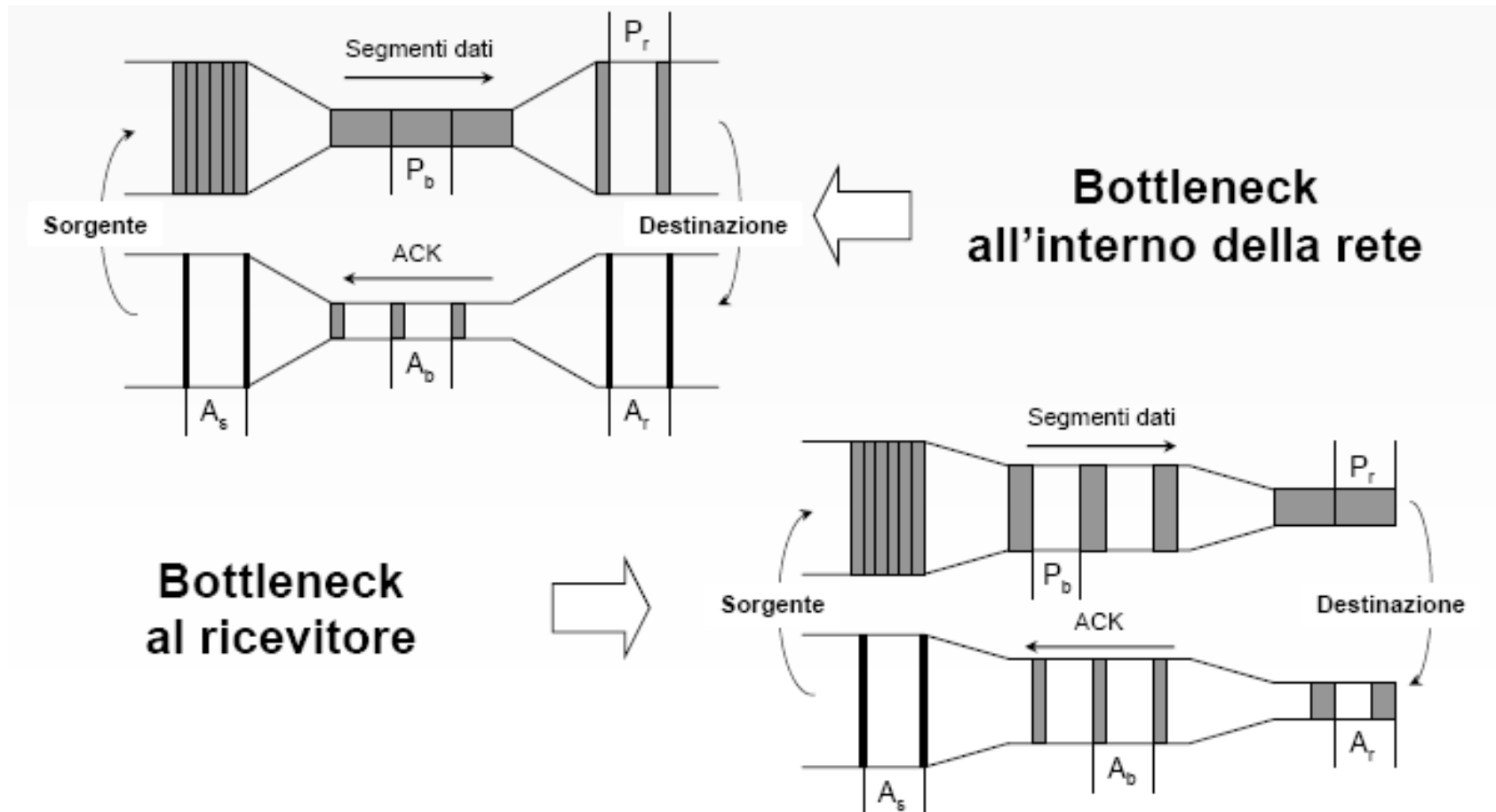
# Controllo di congestione

- In caso di congestione, il controllo di flusso a finestra protegge implicitamente, oltre al destinatario, anche la rete
  - se la rete è congestionata arriveranno meno riscontri e quindi saranno emessi un numero minore di segmenti
  - il meccanismo adattativo di timeout evita ritrasmissioni che porterebbero ad un aumento della congestione invece che ad una sua diminuzione
- Il controllo di flusso end-to-end riesce ad adattare il rate di emissione della sorgente a quello corrispondente al bottleneck della rete (proprietà di self-clocking)

# Controllo di congestione

- I bottleneck in rete possono essere
  - logici, causati dalla congestione nei router
  - fisici, causati dalla limitazione della banda nei collegamenti fisici
  - dovuti al ricevitore, per la limitata capacità elaborativa del ricevitore
- Il controllo di flusso di TCP
  - non è in grado di distinguere il tipo di bottleneck di rete
  - non può stabilire il tipo di contromisura più adatta
- TCP utilizza la stima di RTT come misura di congestione, lo scadere del timeout di ritrasmissione è considerato un sintomo di congestione

# Controllo di congestione



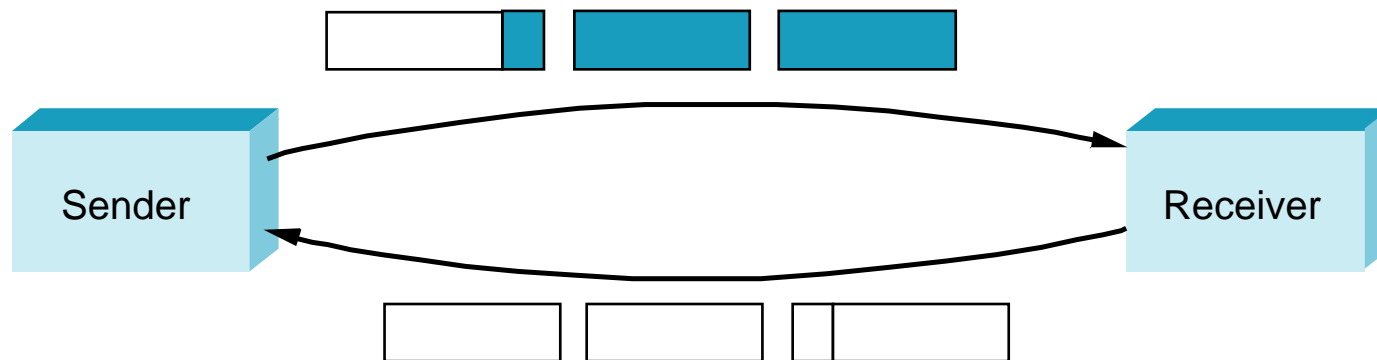
# Controllo di congestione

- Sono definiti dei meccanismi aggiuntivi
- Esistono varie implementazioni di TCP
  - Berkeley
  - Tahoe
  - Reno

	Meccanismo	TCP Berkeley	TCP Tahoe	TCP Reno
Window	Slow Start	◆	◆	◆
	Congestion Avoidance	◆	◆	◆
	Fast retransmit		◆	◆
	Fast recovery			◆

# Silly Window Syndrome

- Slow sender could transmit several TCP segment with few data [high overhead]
- Slow receiver could trigger “little” TCP segment sending once its buffer frees space
- How aggressively does sender exploit open window?



- Receiver-side solutions [Clark solution]
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements
- Sender side solution [Nagle' algorithm]

# Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking
- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack'ed data in transit: buffer it until ACK arrives
    - else: send it

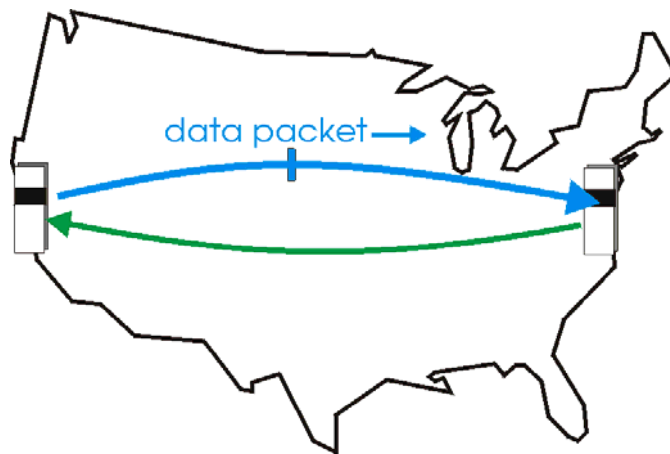
# Protection Against Wrap Around

- 32-bit SequenceNum

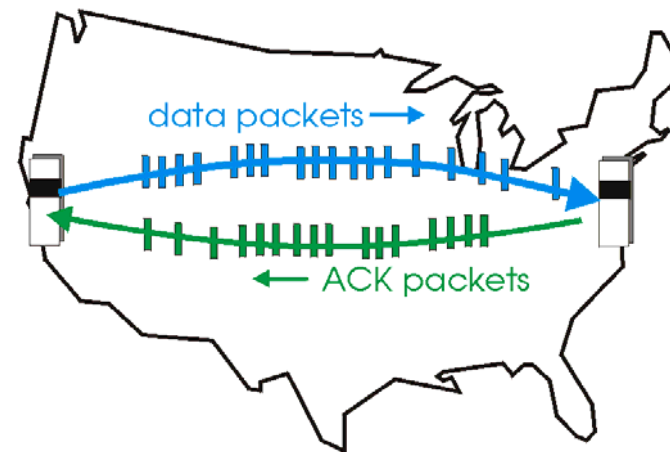
Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

# Keeping the Pipe Full/1

- Delay-bandwidth product:  
 $B \times RTT$
- B is bandwidth



(a) a stop-and-wait protocol in operation



(b) a pipelined protocol in operation

# Keeping the Pipe Full/2

- 16-bit **AdvertisedWindow**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

# Riferimenti

- Testo di Kurose e Ross
  - Cap.3, in particolare 4.1 - 4.5
- Testo di Peterson e Davie
  - Cap. 5, in particolare 5.1 e 5.2