Università degli Studi del Sannio
Facoltà di Ingegneria

Corso di Laurea Specialistica in
Ingegneria delle Telecomunicazioni
A.A. 2009/2010

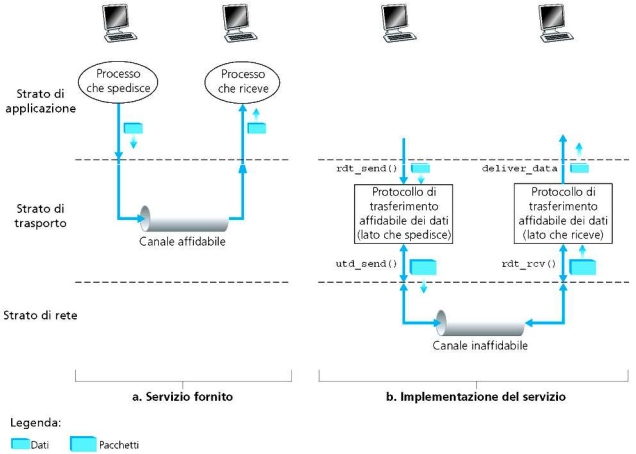
Il livello trasporto

Trasmissione affidabile dei dati
Protocolli ARQ

Ing. Nadia Ranaldo

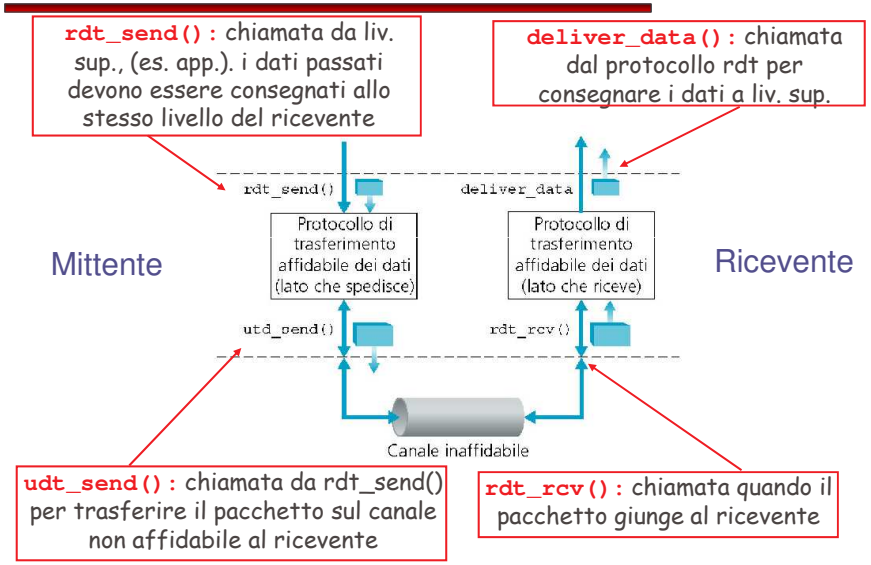
Trasmissione affidabile dei dati

- Questo problema riguarda il livello del trasporto, il livello data-link, ed il livello applicazione



- Le caratteristiche del canale non affidabile determinano la complessità dei protocolli per il trasferimento affidabile dei dati

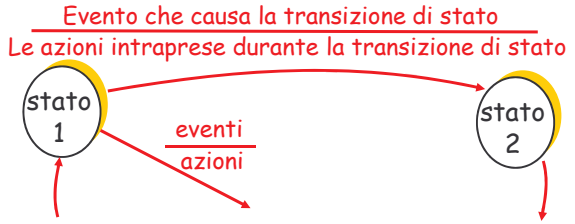
Introduzione (1)



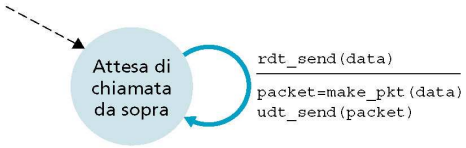
Introduzione (2)

- Le componenti mittente e ricevente del protocollo di trasferimento affidabile dei dati (rdt) saranno sviluppate in modo incrementale
- Si considereranno soltanto trasferimenti unidirezionali
  - Anche se le informazioni di controllo viaggiano in entrambe le direzioni!
- Si impiegheranno le macchine a stati finiti (MSF) per descrivere il comportamento del mittente e del ricevente
- La lettera Λ indica nessuna azione al verificarsi di un evento (sotto) o l'intraprendere di un'azione senza il verificarsi di un evento (sopra)
- Lo stato iniziale è indicato dalla freccia tratteggiata

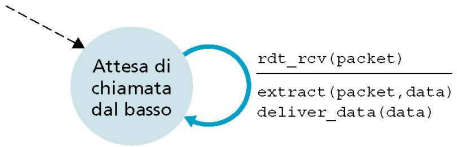
stato: quando è in questo stato, lo stato successivo è determinato univocamente dall'evento successivo



rdt1.0 – un protocollo per canale affidabile



a. rdt1.0: lato che spedisce



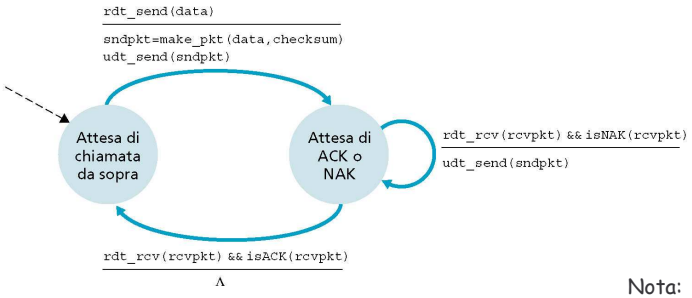
b. rdt1.0: lato che riceve

- Si assume che il canale sottostante sia affidabile
  - Non si presentano errori di bit
  - Non ci sono perdite di pacchetti
- Due MSF separate per il mittente ed il ricevente:
  - Il mittente invia i dati nel canale sottostante
  - Il ricevente legge i dati dal canale sottostante

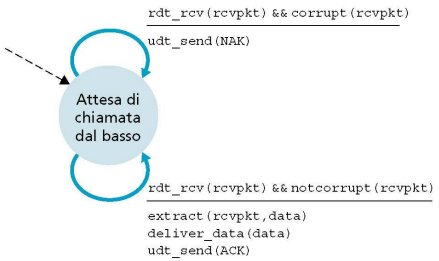
rdt2.0 – canale che introduce errori di bit

- Il canale sottostante può introdurre errori su alcuni bit
  - Nei componenti fisici delle reti quando il pacchetto viene trasmesso, propagato o inserito nei buffer
  - È necessario usare un qualche meccanismo per la rilevazione degli errori
- Continuiamo ad assumere che tutti i pacchetti trasmessi vengano ricevuti nell'ordine di invio
- *Problema*: come recuperare da situazioni d'errore?
  - *Positive acknowledgement (ACK):* il ricevente dice esplicitamente al mittente che il pacchetto spedito è stato ricevuto con successo
  - *Negative acknowledgement (NAK):* il ricevente dice esplicitamente al mittente che il pacchetto spedito presenta qualche errore in ricezione
    - Il mittente ritrasmette il pacchetto alla ricezione di un NAK
- Meccanismi introdotti in `rdt2.0` non presenti in `rdt1.0`
  - Rilevamento degli errori
  - Riscontro da parte del ricevente: impiego di messaggi di controllo (ACK, NAK) dal ricevente al mittente
- I protocolli basati su ritrasmissioni sono detti protocolli ARQ (Automatic Repeat reQuest) e richiedono: rilevamento di errore, feedback del dest. e ritrasmissione

rdt2.0 – specifica mediante MSF



a. rdt2.0: lato che spedisce

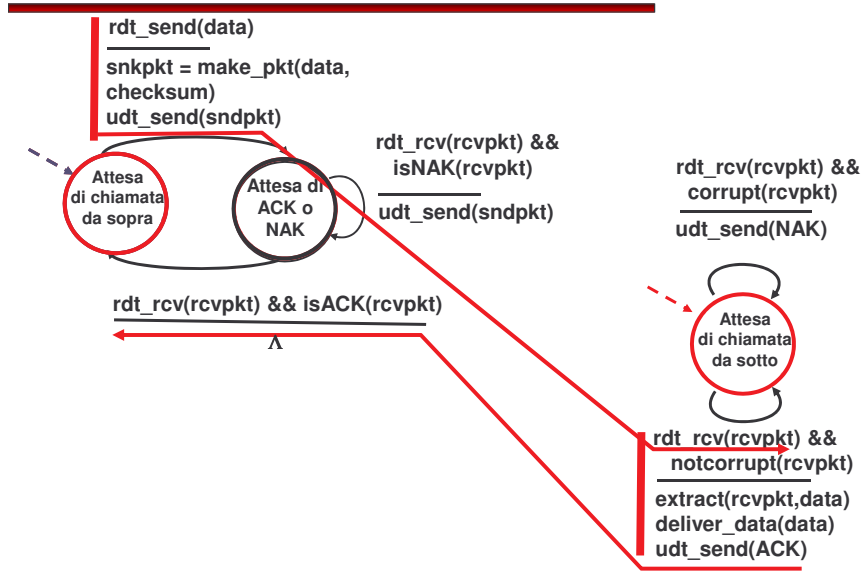


b. rdt2.0: lato che riceve

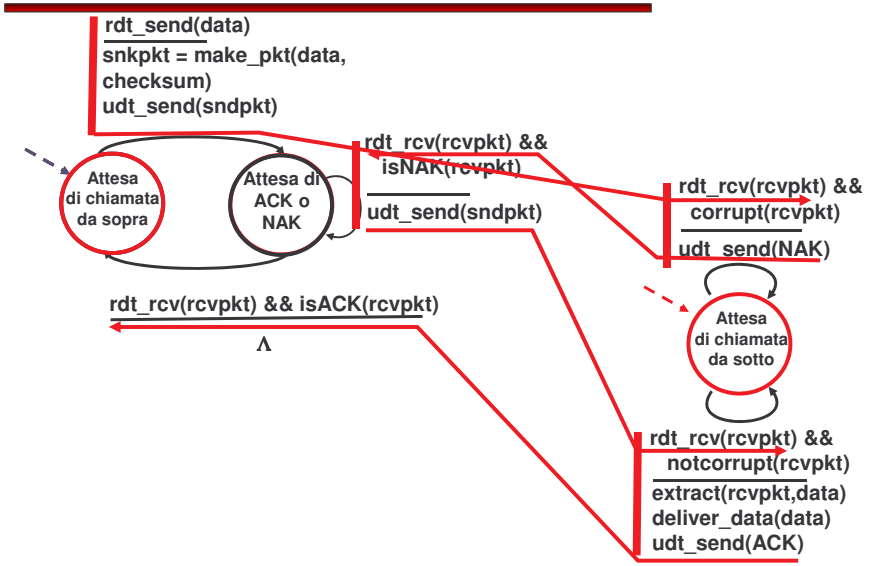
Nota: Il mittente non può inviare nuovi dati finchè non è certo che il destinatario abbia ricevuto il pacchetto corrente

stop and wait
Il mittente invia un pacchetto, quindi aspetta la risposta del ricevente

rdt2.0: funzionamento senza errori



rdt2.0: funzionamento con errori



rdt2.0 ha un punto debole!

Cosa accade se il canale altera un ACK/NAK?

- Il mittente non sa cosa è realmente accaduto al ricevente!

Problemi

- Cosa succede se il riscontro (ACK/NAK) viene alterato?

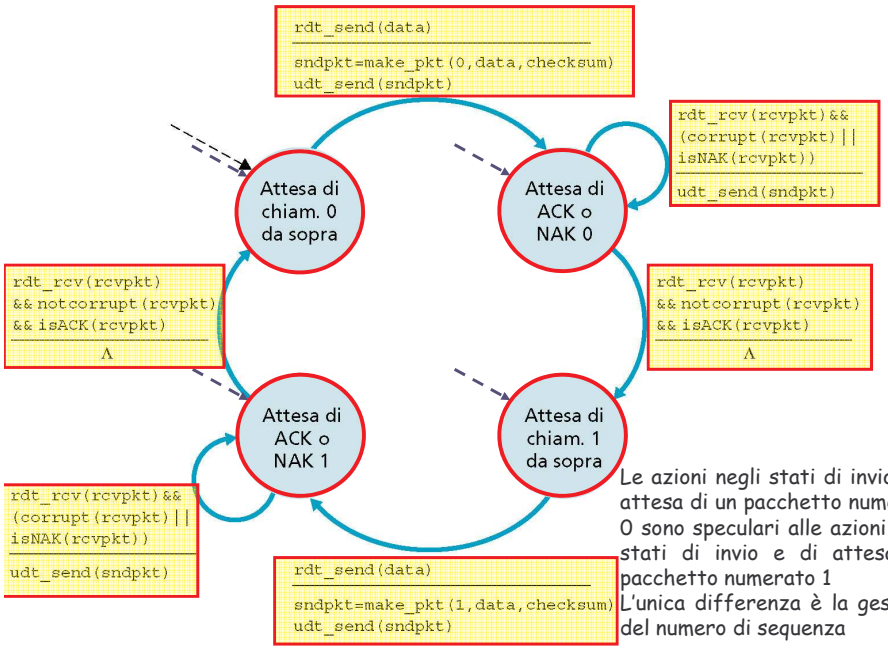
Soluzione

- Bit di checksum ai pacchetti di ACK/NAK
- Il mittente ritrasmette in ogni caso quando riceve un ACK o un NAK alterati
- …ma questo potrebbe causare la ritrasmissione di pacchetti ricevuti correttamente!

Gestione dei duplicati:

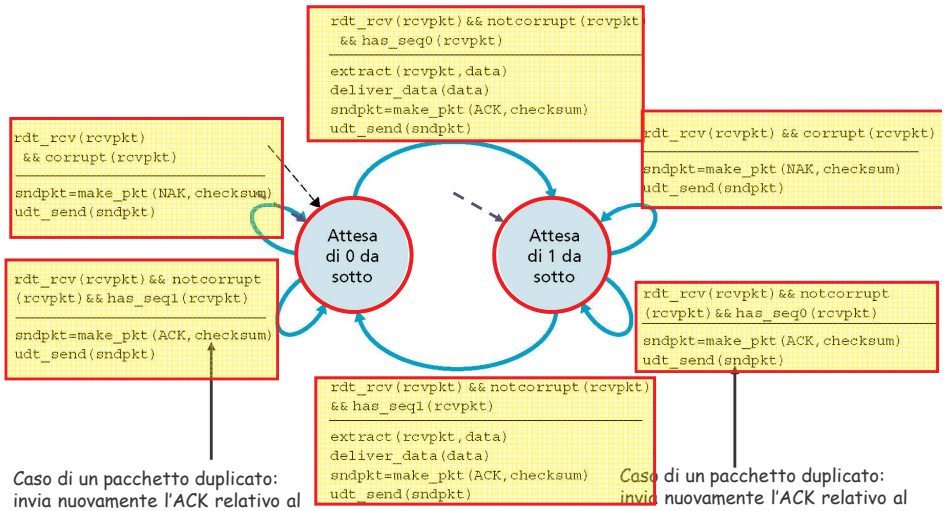
- Il ricevente non sa "a priori" se un pacchetto in arrivo contenga dati nuovi o rappresenti una ritrasmissione
- Il mittente aggiunge un numero di sequenza ad ogni pacchetto
- Il mittente ritrasmette il pacchetto corrente se l' ACK/NAK è danneggiato
- Il ricevente scarta il pacchetto duplicato, non consegnandolo al livello superiore
- Per questo protocollo stop and wait è sufficiente un numero di sequenza da 1 bit (occorre solo distinguere tra un pacchetto ed il precedente)
- Poiché per ipotesi il canale non perde pacchetti, i pacchetti ACK/NAK non devono indicare il numero di sequenza del pacchetto a cui si riferiscono

rdt2.1 – Gestione ACK/NAK errati, mitt.



Le azioni negli stati di invio e di attesa di un pacchetto numerato 0 sono speculari alle azioni negli stati di invio e di attesa del pacchetto numerato 1
L'unica differenza è la gestione del numero di sequenza

rdt2.1 – Gestione ACK/NAK errati, ricev.



Caso di un pacchetto duplicato: invia nuovamente l'ACK relativo al pacchetto precedente ma non lo consegna allo strato superiore

Caso di un pacchetto duplicato: invia nuovamente l'ACK relativo al pacchetto precedente ma non lo consegna allo strato superiore

Alcune considerazioni

Mittente:

- Aggiunta di # di sequenza ai pacchetti
- Sono sufficienti solo due numeri (0,1) => un bit
- Deve controllare se l'ACK/NAK ricevuto è corrotto
- Duplicazione del numero degli stati
  - Lo stato deve "ricordare" se il pacchetto corrente ha come # di sequenza 0 o 1

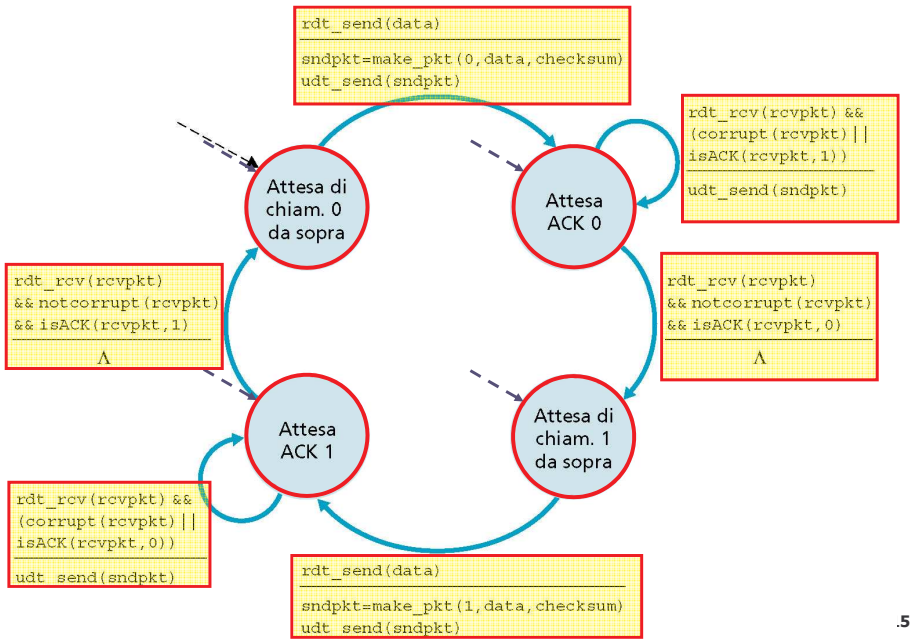
Ricevente:

- Deve controllare se il pacchetto ricevuto è un duplicato
  - Lo stato indica se il numero di sequenza atteso è 0 o 1
- Nota: il ricevente non sa se il suo ultimo ack è stato ricevuto correttamente dal mittente

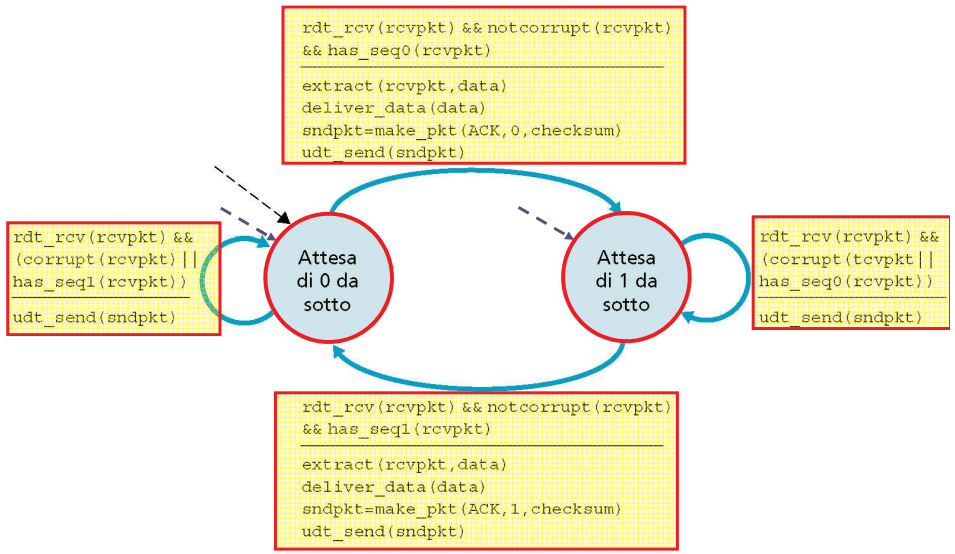
rdt2.2 – un protocollo senza NAK

- Stesse funzionalità di rdt2.1, usando solo ACK
- Invece di usare i NAK, il ricevente invia un ACK per l'ultimo pacchetto ricevuto correttamente
  - Il ricevente deve includere esplicitamente il numero di sequenza del pacchetto che si sta riscontrando
  - Nel caso precedente (uso di ACK e NAK) il numero di sequenza negli ACK non era necessario
- La duplicazione di un ACK al mittente è interpretata come un NAK: si ritrasmette il pacchetto corrente

rdt 2.2 - mittente



rdt 2.2 - ricevente



rdt3.0 – canale con errori e perdite di pacchetti

Nuova ipotesi:

Il canale sottostante può perdere pacchetti (dati o ACK)

- Rilevazione degli errori, # di seq. (0 e 1), ACK, ritrasmissione sono sicuramente di aiuto, ma non sono sufficienti

Problema:

Come individuare la perdita di un pacchetto?

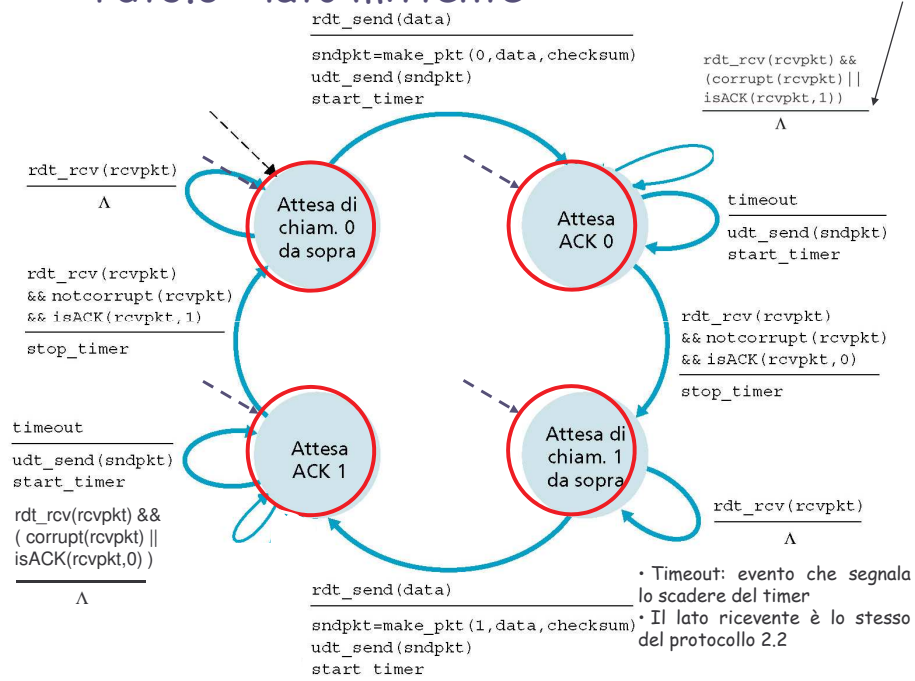
- Una soluzione prevede di operare dal lato mittente: se il mittente non riceve un ACK in un tempo ragionevole è molto probabile che o il pacchetto o l'ACK siano stati persi

Cosa fare quando un pacchetto viene perso?

Approccio:

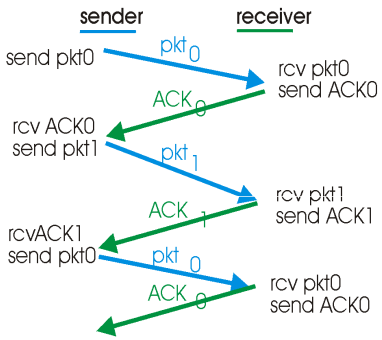
- Il mittente aspetta di ricevere un ACK per un tempo ragionevole
- Ritrasmette se in questo intervallo temporale non viene ricevuto nessun ACK
- Se un pacchetto (o ACK) arriva in ritardo:
  - Il pacchetto sarà ritrasmesso e duplicato in ricezione. L'uso dei # di sequenza risolve il problema
  - Il ricevente deve specificare il # di sequenza del pacchetto di cui il mittente richiede il riscontro
- È richiesto l'impiego di timer
- Quanto tempo deve attendere il mittente? minimo ritardo di andata e ritorno tra mitt. e dest. + il tempo per l'elab. di un pacchetto da parte del destinatario
  - Difficile da stimare – tempo ragionevole oltre il quale la perdita è probabile

rdt3.0 - lato mittente

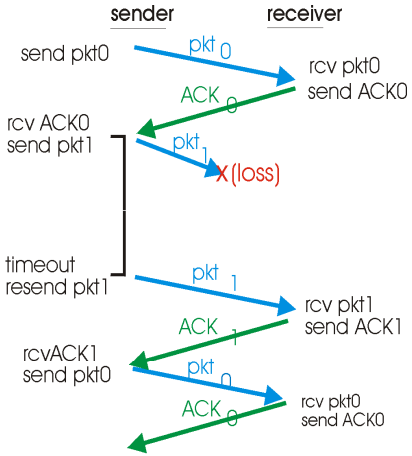


- Timeout prematuro: vedi es. d)
- Timeout: evento che segnala lo scadere del timer
- Il lato ricevente è lo stesso del protocollo 2.2

rdt3.0 - funzionamento

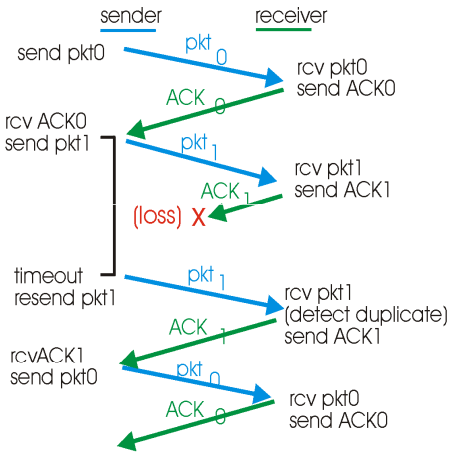


(a) operation with no loss

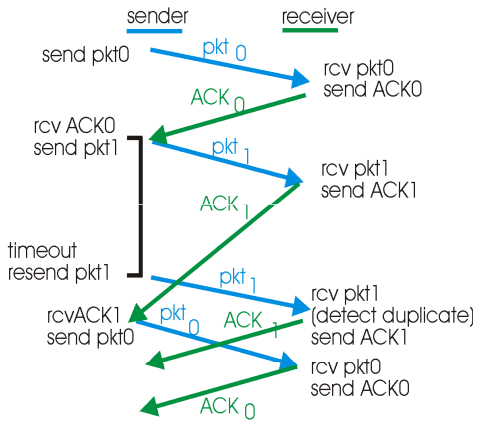


(b) lost packet

rdt3.0 - funzionamento



(c) lost ACK

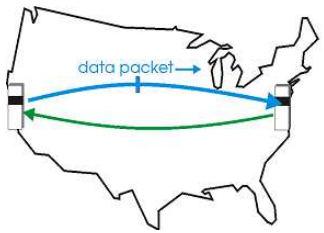


(d) premature timeout

Prestazioni di rdt3.0

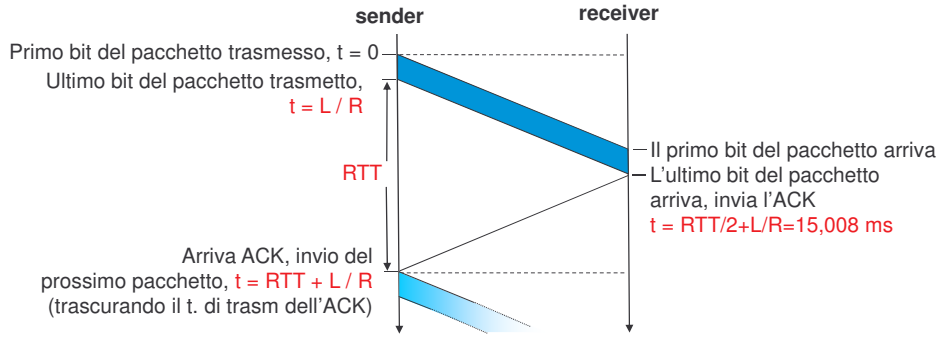
- rdt3.0 funziona, ma le prestazioni non sono soddisfacenti
- Esempio à dato un link da 1 Gbps, un ritardo end-to-end di 15 ms, e pacchetti da 1KB (considerando 1K=10^3)
  - RTT = 15ms+15ms = 30 ms

$$T_{trasm.} = L/R = \frac{8kbit/pkt}{10^9 bit/s} = 8\mu s$$



Utilizzo del canale = U = Frazione del tempo per cui il mittente è impegnato a trasmettere bit sul canale

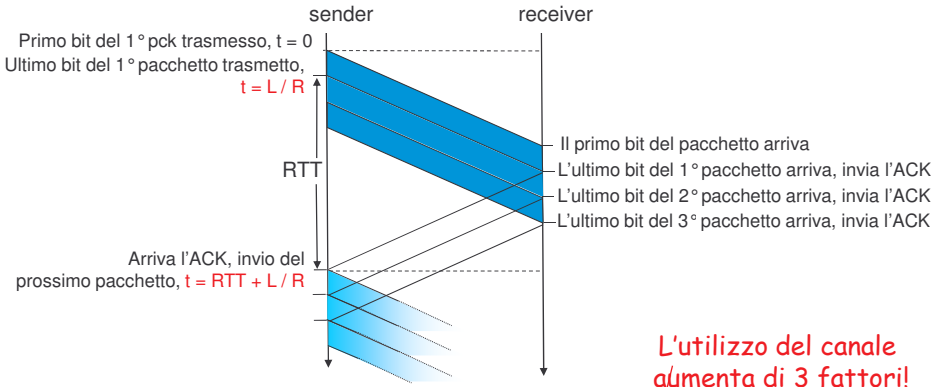
rdt3.0: funzionamento stop-and-wait



$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

- 1 pacchetto da 1KB ogni 30 ms -> throughput = 33KB/s (267 Kbps) su un link da 1 Gbps link
- Il protocollo di rete limita le capacità delle risorse fisiche!

Pipelining: miglioramento dell'utilizzo del canale



L'utilizzo del canale aumenta di 3 fattori!

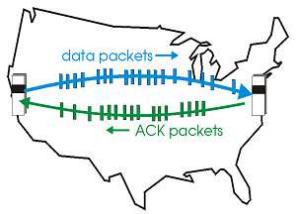
$$U_{sender} = \frac{3 * L/R}{RTT + L/R} = \frac{.024}{30.008} = 0.0008$$

Impiego del pipelining

Pipelining: la componente mittente del protocollo consente di inviare pacchetti anche se non sono stati ancora ricevuti i riscontri dei precedenti

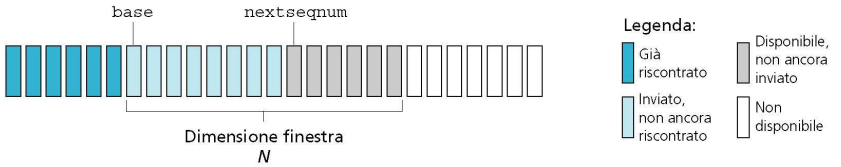
- L'intervallo dei numeri di sequenza deve essere ampliato
  - Ogni pacchetto in transito deve avere un n. di sequenza univoco e ci potrebbero essere più pacchetti in transito non ancora riscontrati
- Viene impiegata la bufferizzazione dal lato mittente e dal lato ricevente
  - Il mittente dovrà bufferizzare i pacchetti che sono stati trasmessi ma non ancora riscontrati
  - La bufferizzazione dal lato del ricevente potrebbe essere richiesta e dipende dal protocollo impiegato per il recupero di pacchetti alterati o persi o troppo in ritardo

§ Due sono le forme di pipelining impiegate:
  § *Go-Back-N*
  § *Selective Repeat*



Go-Back-N (1)

- Numeri di sequenza a k bit nell'intestazione dei pacchetti, che determina lo spazio dei numeri di sequenza (0, 2^k-1) (operazioni modulo 2^k)
- È consentita una finestra di N pacchetti consecutivi non ancora "riscontrati"

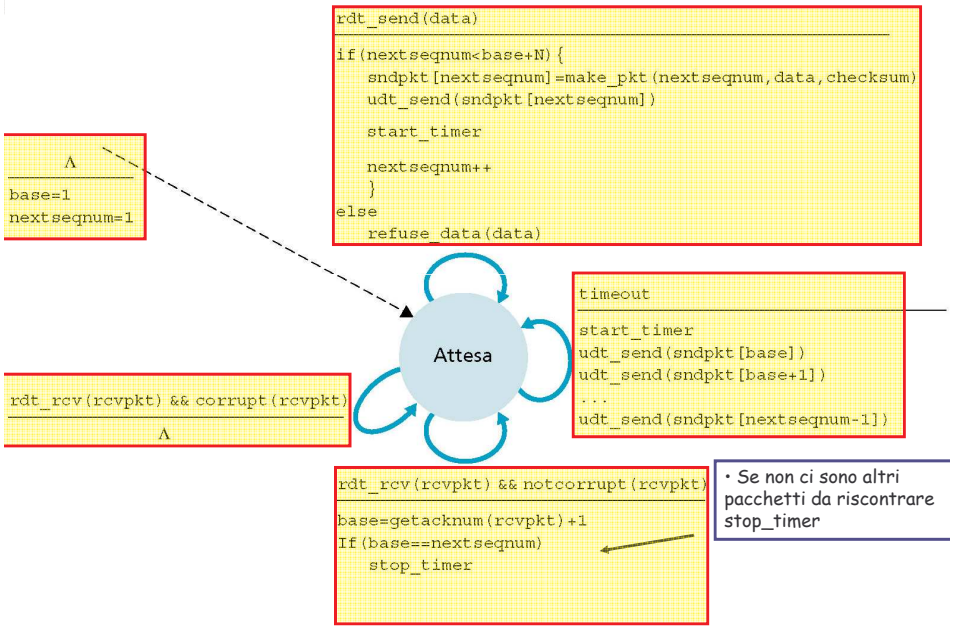


- ACK(n): invia riscontri per tutti i pacchetti fino al numero n incluso - "ACK cumulativo" (altri possibili protocolli utilizzano NAK o ACK non cumulativi)
- È necessario un timer per OGNI pacchetto inviato e di cui non è stato ancora ricevuto un riscontro (si può utilizzare un unico timer che segnala differenti timeout)
- Al verificarsi del timeout si ritrasmettono il pacchetto **base** e tutti i pacchetti con numero di sequenza più alto, presenti nella finestra fino a **nextseqnum** escluso
- I numeri di sequenza da **nextseqnum** e **base+N−1** possono essere utilizzati per i pacchetti da inviare immediatamente nel caso di richiesta dal livello superiore

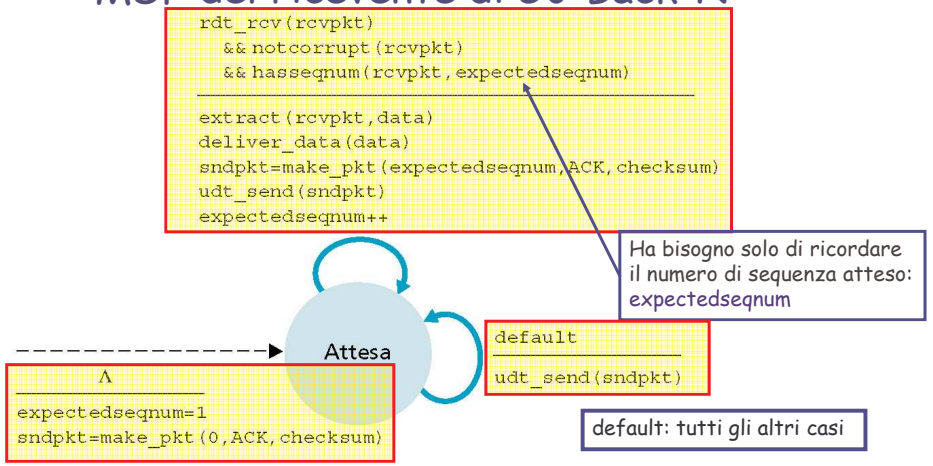
Go-Back-N (2)

- Durante il funzionamento del protocollo, la finestra trasla lungo lo spazio dei numeri di sequenza verso numeri di sequenza più grandi mano a mano che i pacchetti vengono riscontrati attraverso la ricezione di ACK
- Le ritrasmissioni avvengono solo allo scadere di un timeout
- Normalmente la finestra coincide con il buffer a disposizione
- Per questo motivo il protocollo go-back-N (così come il selective repeat) è detto anche protocollo sliding window (a finestra scorrevole)
- La dimensione della finestra (N) può essere utilizzata per limitare la velocità di trasmissione del mittente
  - Utile quindi per il controllo di flusso ed il controllo della congestione
- Mittente
  - Per inviare inizialmente il primo pacchetto si utilizza come numero di sequenza 1, in modo da interpretare un pacchetto di ACK ricevuto con numero di sequenza 0 come indicazione dal ricevente che non è stato ricevuto correttamente il primo pacchetto

MSF del mittente di Go-Back-N

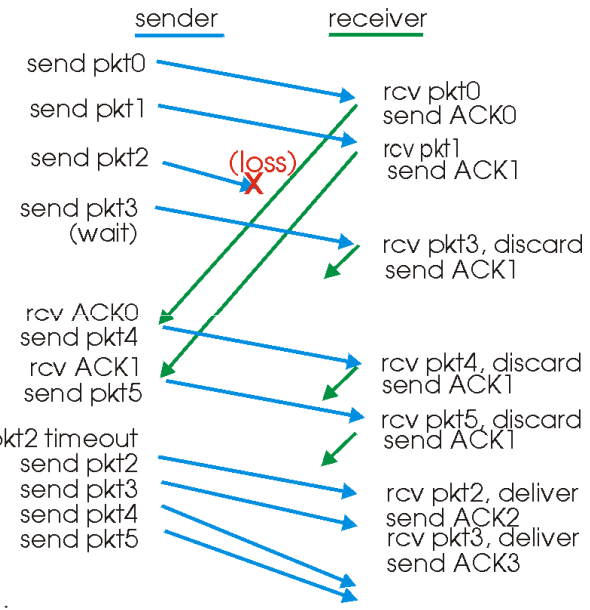


MSF del ricevente di Go-Back-N



- Impiego solo di ACK: invia sempre un ACK per i pacchetti ricevuti correttamente e con numero di sequenza in ordine
  - Può generare ACK duplicati
- Quando i pacchetti arrivano fuori ordine:
  - Sono scartati, non sono usati buffer in ricezione! (vantaggio è la semplicità)
  - Invia un riscontro per il pacchetto con numero di sequenza più alto ricevuto in ordine

Go-Back-N

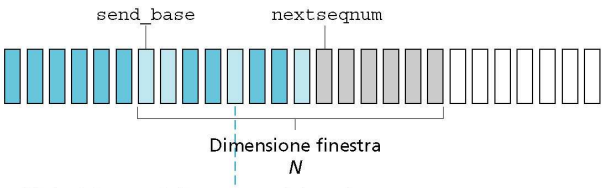


- Finestra di 4 pacchetti
- Perdita del pacchetto 2
- Timeout per il pacchetto 2
- Ritrasmissione dei pacchetti 2, 3, 4 e 5

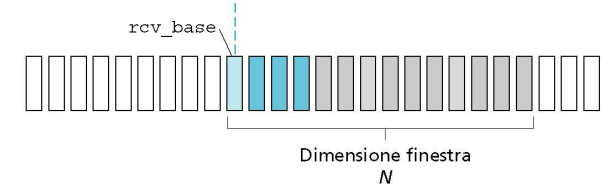
Ripetizione selettiva

- Problema con il go-back-N
  - Un errore su un solo pacchetto può causare la ritrasmissione di un elevato numero di pacchetti
- Protocollo selective repeat permette di evitare ritrasmissioni inutili
- Il mittente rispedisce soltanto i pacchetti per i quali non è stato ricevuto un ACK
  - È richiesto un timer (lato mittente) per ogni pacchetto non riscontrato (si può utilizzare un unico timer che segnala differenti timeout)
- Il ricevente invia riscontri individuali per ogni pacchetto ricevuto correttamente sia in ordine sia fuori ordine
  - Accetta pacchetti che hanno fino ad un certo numero di sequenza individuato dalla dimensione della finestra in ricezione
  - Bufferizza i pacchetti, se è necessario, per consegnarli in ordine al livello superiore
- Finestra del mittente (come in go-back-N)
  - N numeri di sequenza consecutivi
  - Limita i numeri di sequenza dei pacchetti non riscontrati

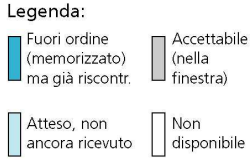
Finestra del mittente e del ricevente



a. Vista dei numeri di sequenza dal sender



b. Vista dei numeri di sequenza dal receiver



Ripetizione selettiva

Lato mittente

Dati provenienti dal liv. sup.:

- Se è disponibile un numero di sequenza in finestra, il pacchetto viene spedito

timeout(n):

- Rispedisce il pacchetto n; riavvia il timer per il pacchetto n

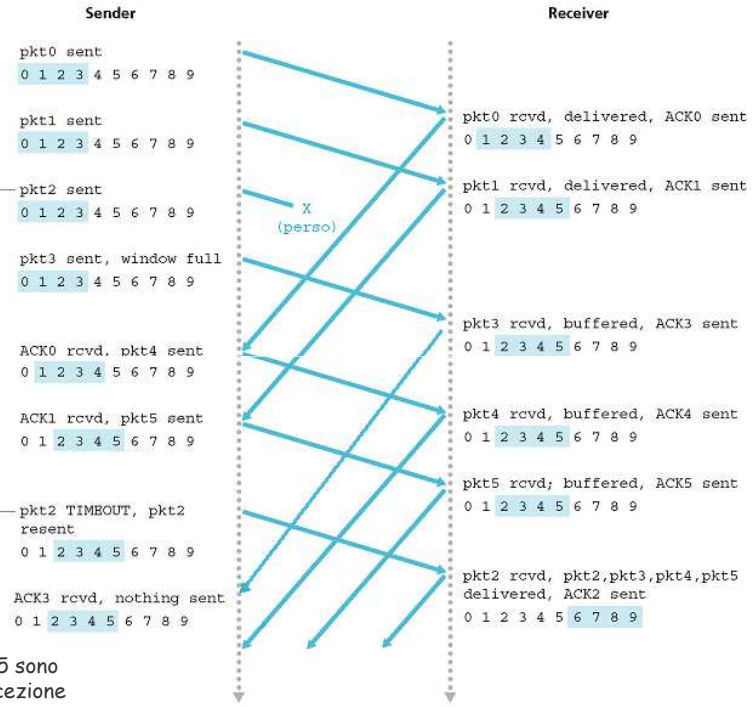
ACK(n) in `[sendbase, sendbase+N-1]`:

- Marca il pacchetto come ricevuto
- Se n è il numero di sequenza più basso in finestra, la base della finestra viene fatta avanzare fino al primo numero di sequenza relativo ad un pacchetto non riscontrato

Lato ricevente

- # di seq. del pacchetto n in `[rcvbase, rcvbase+N-1]`
- Invia l'ACK(n)
- Se fuori ordine: bufferizza
- Se in ordine: consegna (insieme ai pacchetti bufferizzati con # di seq. in ordine); avanza la finestra fino al # di seq. del pacchetto non ancora ricevuto
- # di seq. del pacchetto n in un intervallo al di sotto di rcvbase `[rcvbase-N, rcvbase-1]`
- Invia l'ACK(n), anche se si tratta di un pacchetto già riscontrato
- Caso in cui si perde l'ACK relativo al pacchetto fino a `rcvbase-N`

Altrimenti: Ignora il pacchetto



- I pacchetti 3,4,5 sono bufferizzati in ricezione

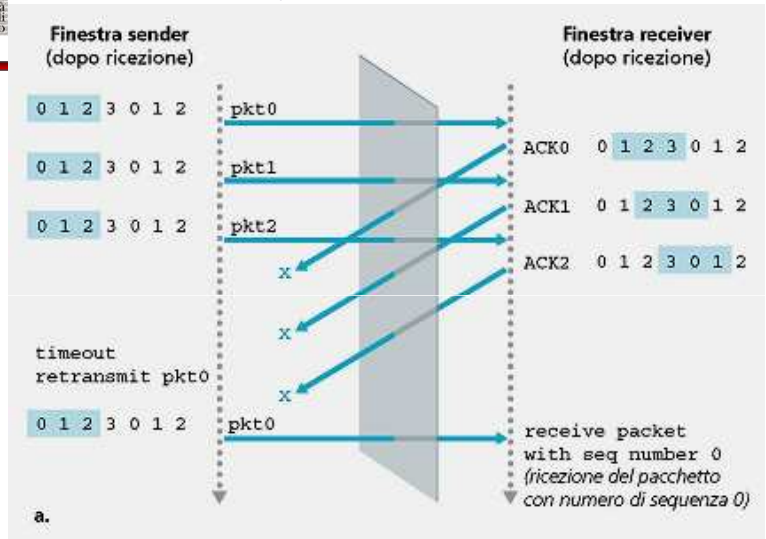
Numeri di sequenza – possibili problemi

- La finestra del mittente e quella del ricevente non sempre coincidono
  - Non sempre mittente e destinatario hanno la stessa visuale su cosa è stato ricevuto correttamente
- La mancanza di sincronizzazione tra finestre del mittente e del destinatario può creare problemi a causa di intervalli finiti per i numeri di sequenza

Esempio:

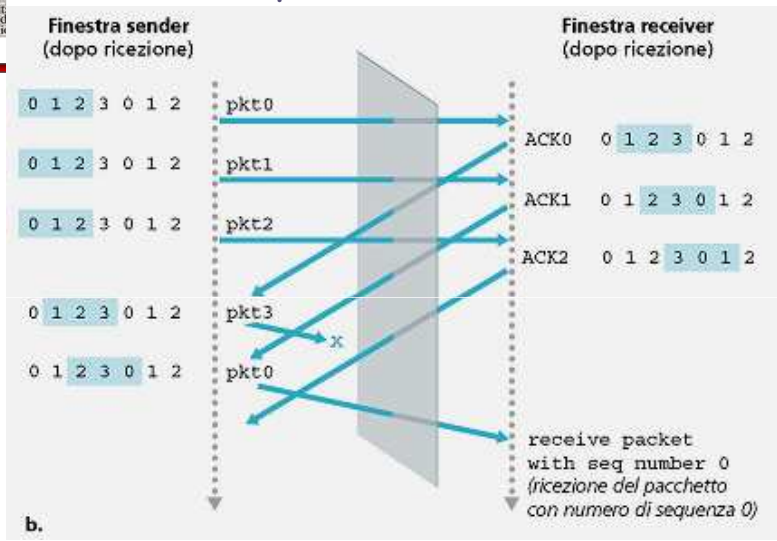
- Numeri di seq.: 0, 1, 2, 3
- Dimensione finestra = 3
- Supponiamo che inizialmente vengano trasmessi i pacchetti 0, 1 e 2 e che questi vengano ricevuti correttamente dal destinatario che quindi li riscontra
- Quando il mittente riceverà gli ack, potrà spostare avanti la propria finestra, e quindi potrà spedire altri pacchetti (il quarto, il quinto ed il sesto) che avranno numero di sequenza 3, poi 0, 1, ecc.

Numeri di sequenza – 1° scenario



• Quando il destinatario riceve il pacchetto con numero di sequenza 0 (ritrasmissione del 1° pacchetto per il mittente) lo interpreta come nuovo pacchetto (come se fosse il 5°)!!!

Numeri di sequenza – 2° scenario



Per il destinatario i due scenari sono gli stessi!
Non c'è modo di distinguere tra la ritrasmissione del 1° pacchetto dalla trasmissione del quinto (che ha nuovamente numero di sequenza 0)

Quale deve essere la dimensione della finestra mittente per evitare questi problemi?

- Si deve evitare di avere che l'estremo superiore della finestra ricevente (il numero di sequenza più alto che può essere accettato) si sovrapponga con l'estremo inferiore della finestra mittente (numero di sequenza di cui si attendono ancora gli ack)
- Che intervallo di numero di sequenza le due finestre possono al massimo coprire?
- Sia k lo spazio dei num. di sequenza e w la dimensione della finestra
- Sia m il num. di seq. del primo pacchetto atteso dal ricevente
  - La finestra del ricevente è [m, m+w-1]
  - Il ricevente ha ricevuto i precedenti pacchetti e ha inviato gli ack
  - Il mittente potrebbe non aver ricevuto nessuno di questi ack quindi la sua finestra può essere al limite [m-w, m-1]
- Quindi lo spazio dei numeri di sequenza deve essere abbastanza ampio da includere numeri di sequenza differenti per la finestra del mittente e quella del ricevente che in totale sarà #[m-w, m+w-1] =2w

=> **k>=2w**

Esercizo

- Consideriamo il protocollo go-back-N con dimensione di finestra pari a N. Supponiamo che all'istante t il successivo pacchetto nella sequenza atteso dal ricevente abbia num. di seq. k (base della finestra di ricezione)
- Quali sono i possibili num. di seq. all'interno della finestra del mittente all'istante t?
  - Indichiamo con N la dimensione della finestra
  - Il ricevente ha ricevuto i pacchetti con numero di sequenza k-1 e quelli ancora precedenti fino a k-N
    - Se il mittente ha ricevuto questi ack, allora la sua finestra sarà [k, k+N-1]
    - Se il mittente non ha ricevuto questi ack, allora la sua finestra sarà [k-N, k-1]
  - In generale, la finestra del mittente inizierà con un num. di sequenza incluso nell'intervallo [k-N, k]
  - Quindi al più si troverà nell'intervallo [k-N, k+N-1]