



Reti di Calcolatori

Il livello data-link

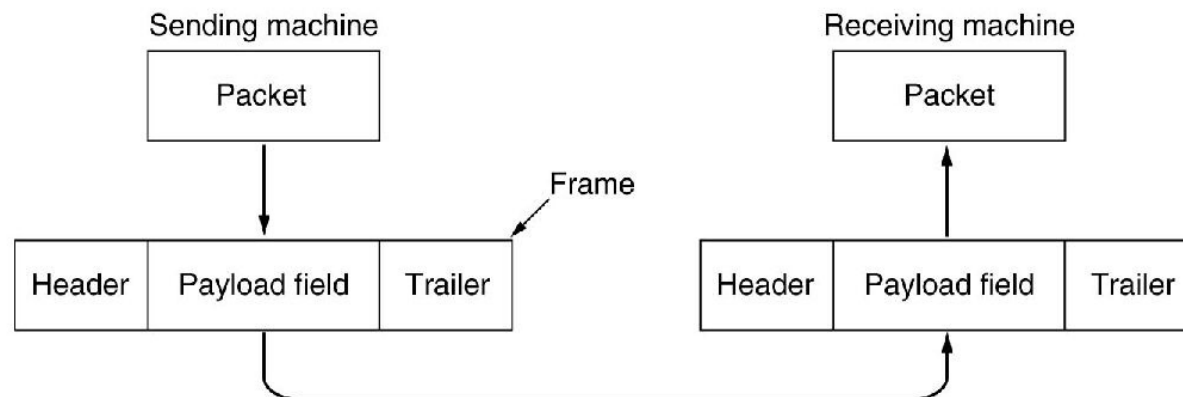


Il data link layer

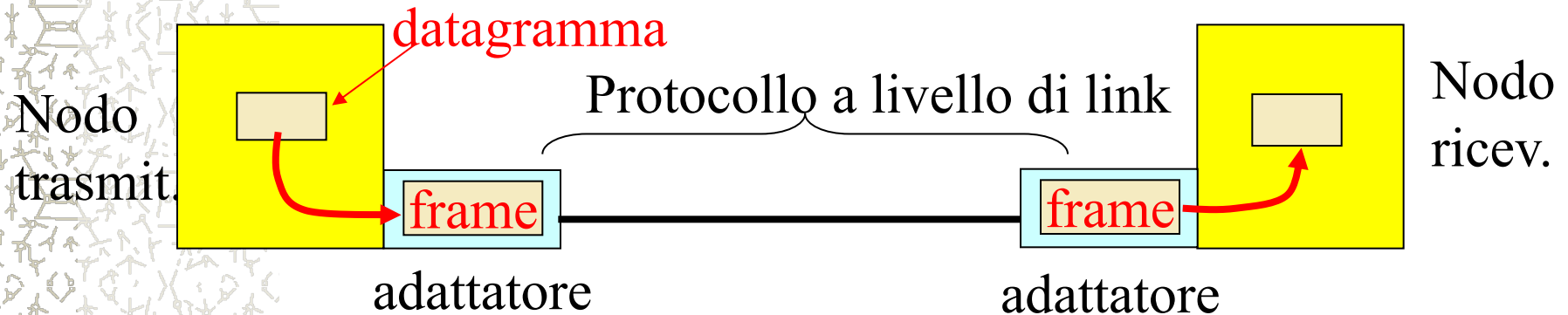
- Il Data Link Layer (anche **livello di collegamento dati**, o piu' semplicemente: **livello 2**) ha la funzione principale di fornire allo strato di rete servizi per il **recapito di dati** al nodo direttamente **adiacente** sulla rete
- Il compito del data link layer e' quindi quello di **organizzare** il trasferimento dei dati tra **due apparati adiacenti**, e di fornire una interfaccia definita per consentire allo strato di rete di **accedere** ai **servizi** offerti
- Apparati adiacenti significa **logicamente connessi** da un "**canale**" che trasmette i bit da una parte e li riceve dall'altra, **nell'ordine** di trasmissione
- Il data link layer utilizzerà i **servizi dello strato fisico** per il recapito dei dati al suo processo paritario sul calcolatore ricevente, ma logicamente la comunicazione avverrà **direttamente** con il processo di data link layer **remoto**
- come sia fatto il "canale" **non e'** argomento che riguardi il **data link layer**, ma lo **strato fisico**: non importa se ci sia un cavo, una fibra, una sequenza di mezzi differenti con interposti ripetitori, convertitori elettrico/ottici, modem, multiplexer, antenne o altro

Il data link layer (cont.)

- Per realizzare le sue funzioni il data link layer riceve i dati dallo strato di rete (**pacchetti**), li organizza in frame (**frame**) eventualmente **spezzando** in più frame il blocco di dati ricevuto dal livello 3, aggiunge ad ogni frame una intestazione ed una coda (**header** e **trailer**), e passa il tutto allo **strato fisico** per la trasmissione
- In ricezione il data link layer **riceve** i dati dallo strato fisico, effettua i **controlli** necessari, **elimina** header e trailer, **ricombina i frame** e passa i dati ricevuti allo **strato di rete**



Adattatori di interfaccia



- Il protocollo a livello di link è realizzato da un adattatore (NIC, scheda di interfaccia di rete)
 - Adattatori Ethernet, adattatori PCMCIA e adattatori 802.11
- Lato trasmittente:
 - Incapsula un datagramma in un frame.
 - Imposta il bit rilevazione degli errori, trasferimento dati affidabile, controllo di flusso, etc.

- Lato ricevente:
 - Individua gli errori, trasferimento dati affidabile, controllo di flusso, etc.
 - Estrae i datagrammi e li passa al nodo ricevente
- L'adattatore è un'unità semi-autonoma.

Servizi del DLL

- Normalmente la progettazione dello strato 2 fornisce allo strato di rete i servizi
 - trasmissione dati **senza riscontro e senza connessione**
 - trasmissione dati **affidabile senza connessione**
 - trasmissione **affidabile con connessione**
- La classe di servizio non affidabile senza connessione è adatta su linee di **elevata** qualità
 - il controllo sugli errori e la ritrasmissione di frame errati comporta una **inefficienza** in termini di **numero di bit trasmessi** rispetto ai dati, con riduzione del **tasso utile** ed aumento della **probabilità** di errore
 - il controllo può essere **demandato** ai livelli **superiori** a vantaggio della efficienza del livello di data link
 - generalmente questi servizi sono utilizzati su **rete locale**
 - servizi non affidabili sono utilizzati anche per il traffico **voce e video**

Servizi del DLL (cont.)

- La classe di servizio **affidabile** con connessione e' adatta su linee piu' frequentemente soggette ad **errori**
 - demandare il controllo e la **ritrasmissione** ai livelli superiori (che generalmente trasmettono pacchetti costituiti da **piu' frame**) in caso di elevata probabilita' di errore potrebbe causare la **ritrasmissione** di molti pacchetti, mentre al livello due puo' essere sufficiente la ritrasmissione del **singolo frame**
 - Implementa meccanismi di **riscontro** per verificare la necessita' di **ritrasmissioni**
 - tipicamente utilizzata su linee a **grande distanza** (connessioni WAN), anche se la fibra ottica riduce notevolmente questo problema
- Il data link layer deve quindi poter offrire le **diverse classi** di servizio, per soddisfare le diverse esigenze conseguenti alle diverse circostanze

Problematiche del livello 2

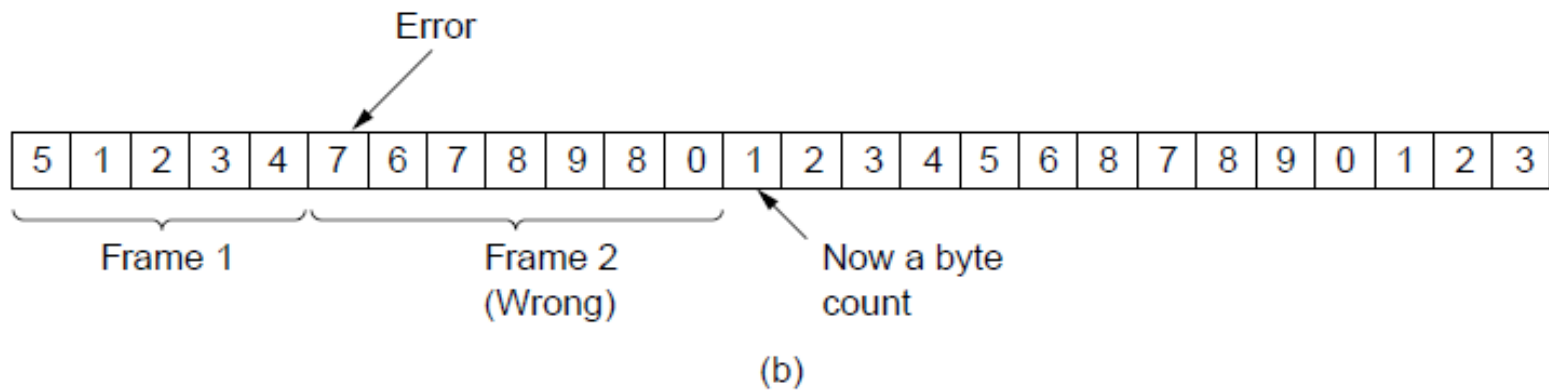
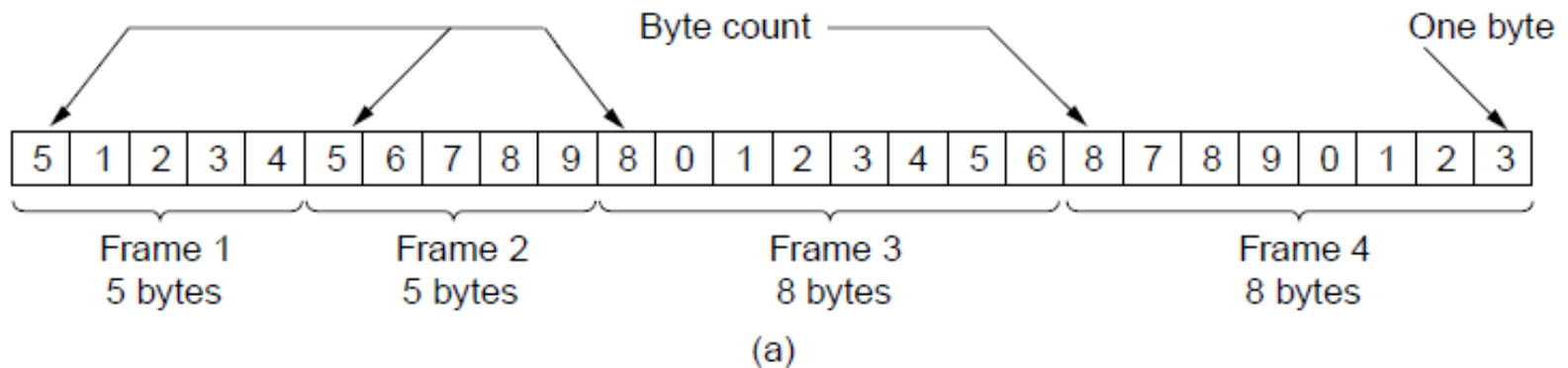
- Per poter svolgere le sue funzioni il data link layer dovrà **curare** i seguenti aspetti:
 - la **organizzazione** del flusso di bit in **frame**, con controllo per la **sincronizzazione**, inserimento e rimozione di **header** e **trailer**, riordinamento dei frame in ricezione
 - organizzare il trasferimento dei dati in modo da **gestire** eventuali **errori di trasmissione**, utilizzando codici di **correzione** degli errori o codici di **identificazione** degli errori e gestendo la **ritrasmissione** dei frame errati
 - realizzare il **controllo di flusso**, per utilizzare in modo efficiente il canale trasmissivo impedendo al contempo ad un **trasmettitore veloce** di sovraccaricare un **ricevitore lento**

Framing e sincronizzazione

- Per trasportare i bit il Data Link Layer utilizza i servizi dello strato fisico
- Lo strato fisico non può **garantire** il trasferimento privo di errori, che dovranno essere gestiti dal DLL
- Per fare ciò il DLL organizza i bit in **frame**, ed effettua i controlli **per ogni** frame
- La gestione del frame deve prevedere in primo luogo la possibilità del ricevente di **identificare** il frame, quindi si devono adottare regole per **delimitarlo** e poterne identificare i **limiti** in ricezione
- Esistono diverse tecniche
 - conteggio dei caratteri
 - byte di flag, e byte stuffing
 - bit(s) di flag di inizio, e fine e bit stuffing

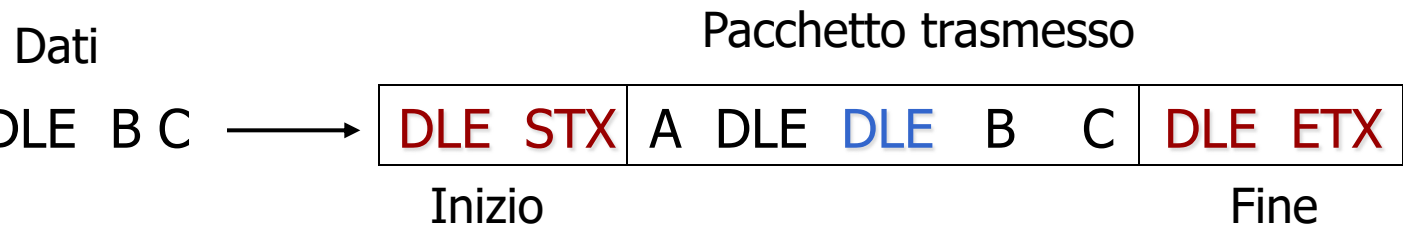
Conteggio dei caratteri

- Un campo dell'intestazione indica il numero di caratteri nel pacchetto
- Se si perde il sincronismo non si riesce a trovare l'inizio di un pacchetto successivo



Caratteri di inizio e fine

- I pacchetti di dati sono chiamati **STX** e **ETX**.
 - Ci si aspetta che i dati siano **incontrati** in un **ricevitore**.
 - Un **STX** è un **pacchetto** di dati.
- A D



Byte Delimitatore

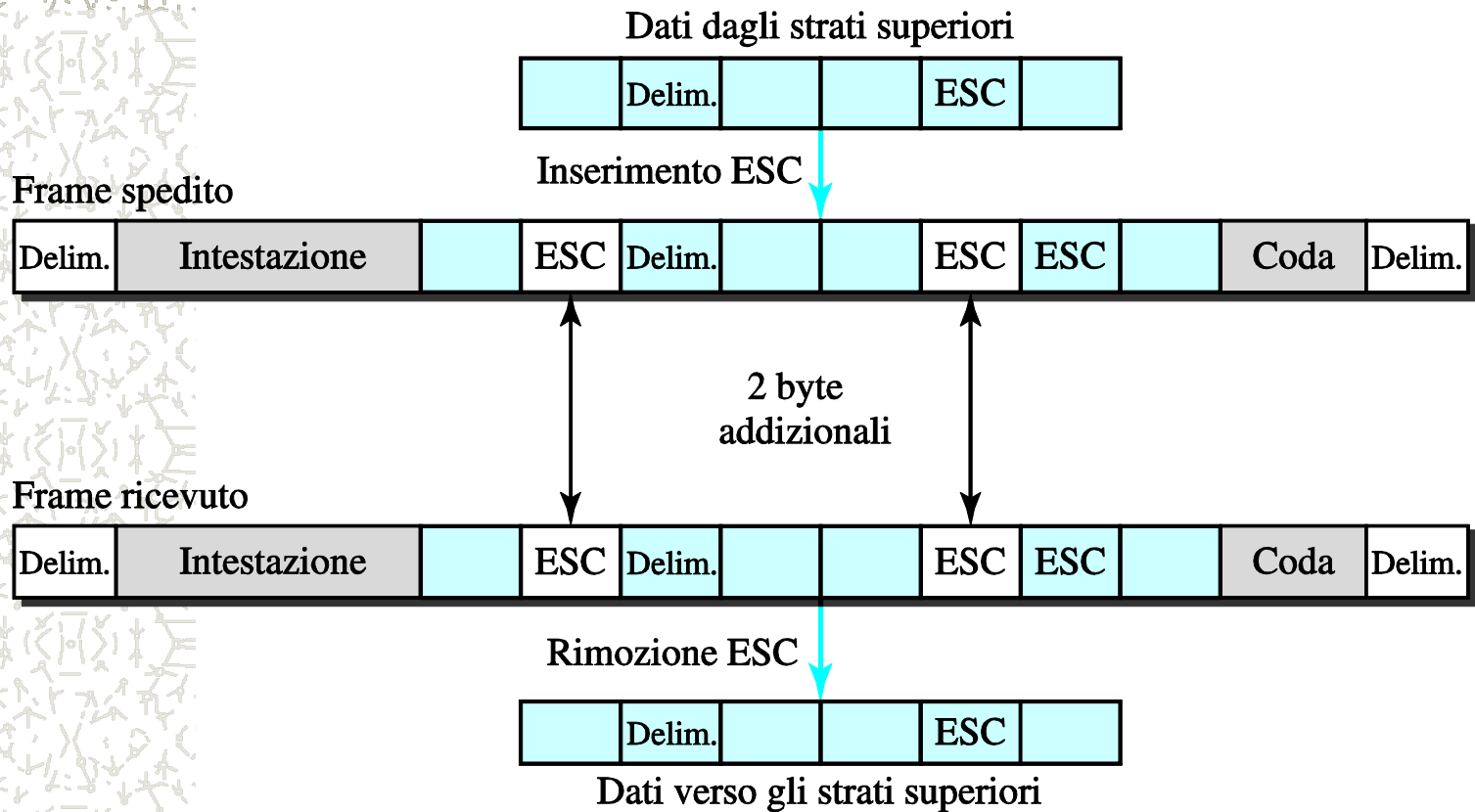
- I pacchetti sono iniziati e terminati con una sequenza speciale di bit detta delimitatore o flag-byte

Flag byte = **01111110**

- Per evitare che il flag byte possa trovarsi all'interno dei dati del pacchetto, possiamo fare ricorso a 2 differenti tecniche:
 - **Riempimento di caratteri (Byte Stuffing)**
 - **Riempimento di bit (Bit Stuffing)**

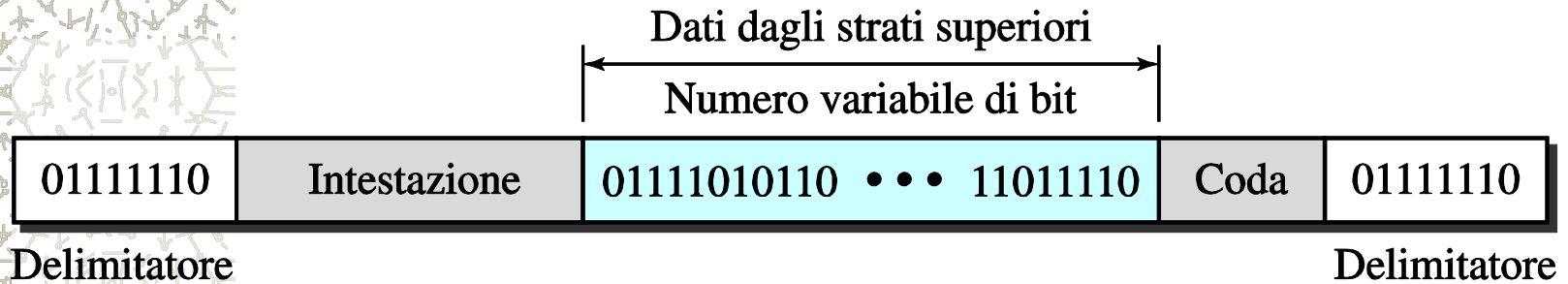
Byte stuffing

12



La tecnica di byte stuffing consiste nell'inserimento di 1 byte (ESC) aggiuntivo ogni qual volta nei dati compare il carattere delimitatore o il carattere ESC stesso

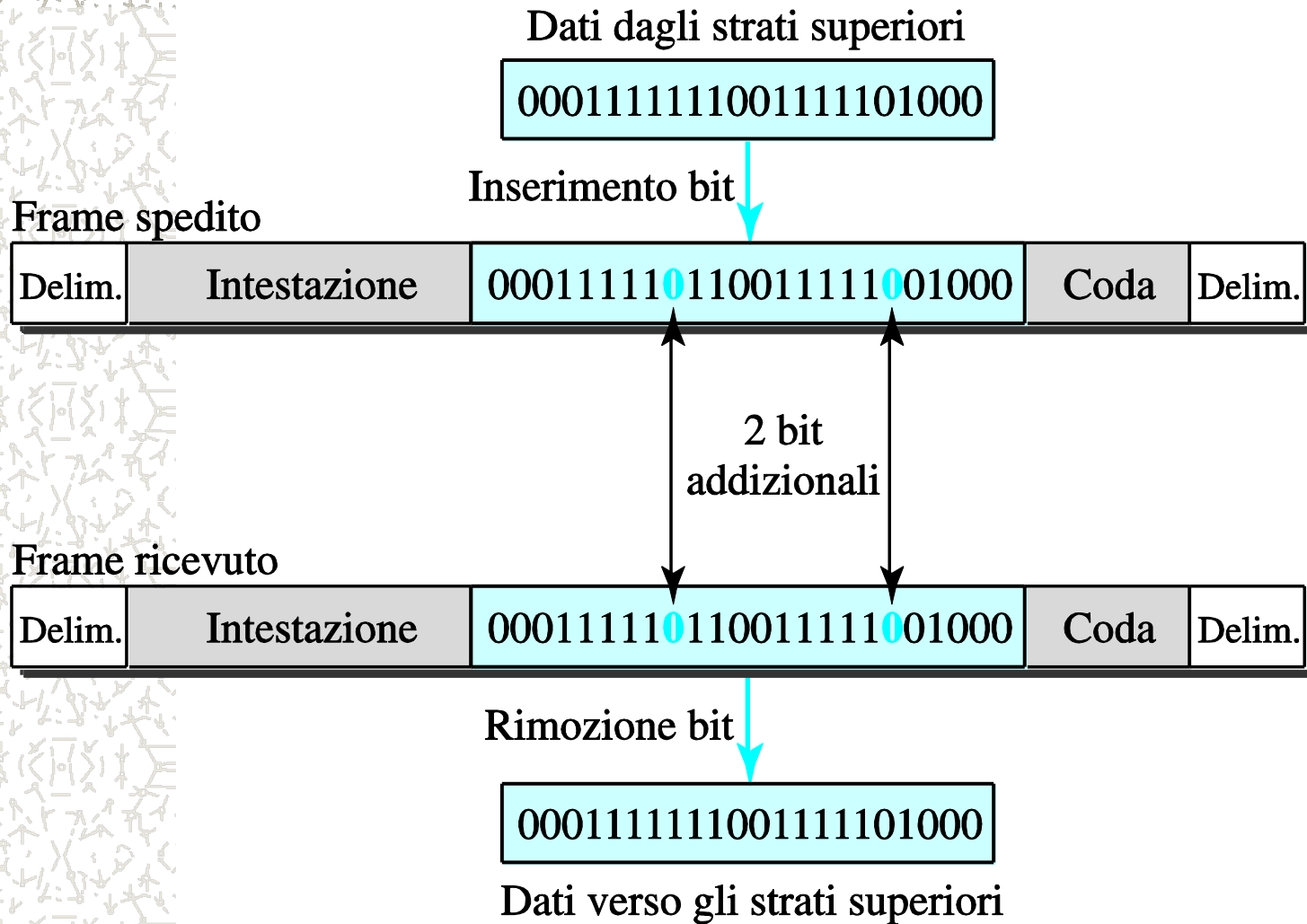
Bit stuffing



- I bit all'interno del frame non sono considerati come sequenza di byte
- Bit stuffing
 - Mittente: dopo una sequenza di uno 0 e cinque 1 consecutivi, inserisce uno 0
 - Destinatario: se riceve 01111110, elimina l'ultimo zero

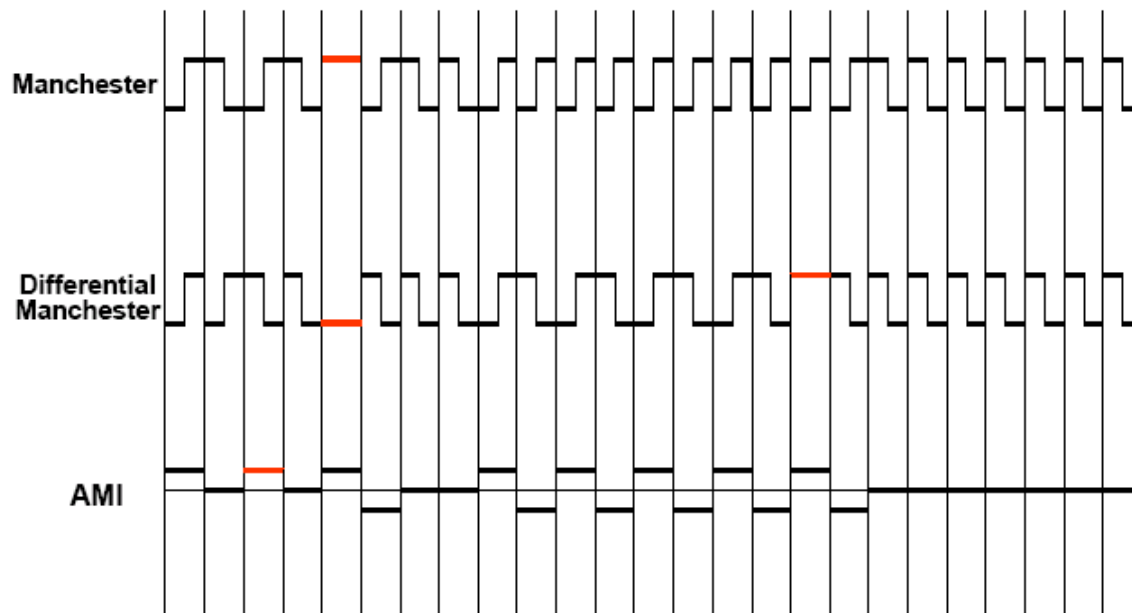
La tecnica di bit stuffing consiste nell'inserire uno 0 aggiuntivo ogni qualvolta cinque 1 consecutivi seguono uno 0 nei dati, per evitare che il destinatario interpreti una possibile sequenza di 01111110 nei dati come delimitatore.

Bit stuffing



Violazioni di codifica

- È possibile segnalare l'inizio o la fine di una trama con una deliberata violazione delle regole di codifica del segnale
- Ad esempio, usando la codifica Manchester differenziale è possibile ottenere una violazione omettendo la transizione da 1 a 0 o da 0 a 1 nel mezzo di un impulso per indicare rispettivamente la fine o l'inizio di una trama



Controllo di flusso

- Può capitare che una sorgente sia in grado di trasmettere ad un tasso **piu' alto** della capacita' di **ricevere** a destinazione
- Senza controllo, questo implica che la destinazione inizierebbe a **scartare frame** trasmessi correttamente per **mancanza di risorse** (tempo di processamento, buffer)
- Il protocollo deve poter gestire questa situazione e prevedere **meccanismi** per **rallentare** la trasmissione
- Tipicamente il protocollo prevedera' dei **frame di controllo** con cui il ricevente puo' **inibire** e **riabilitare** la trasmissione di frame, cioe' il protocollo stabilisce **quando** il trasmittente puo' inviare frame
- Vedremo **diverse tecniche**, che si differenziano per complessita' ed **efficienza** di utilizzo della linea

Controllo di flusso

- L'implementazione del data link layer prevederà la realizzazione della **interfaccia** con i livelli adiacenti, ad esempio due **procedure** *from-network-layer()* e *to-network-layer()* per **scambiare dati** con il livello superiore, e due procedure analoghe per scambiare dati con lo strato fisico
- In aggiunta sarà prevista una procedura *wait-for-event()* che metterà il data link layer in **attesa** di un evento
- Questo evento sarà in generale la **segnalazione**, da parte di uno dei due **livelli adiacenti**, che sono disponibili **dei dati**
- Infine, saranno definite procedure per il **trattamento dei dati** (inserimento/rimozione di header, calcolo di checksum, ...)

Controllo di flusso (cont.)

- In ricezione, il data link layer verrà **svegliato** per **prelevare dati** dallo strato fisico, **processarli**, e **passarli** allo strato di rete
- Di fatto il DDL in ricezione **non sarà** in grado di rispondere ad eventi per il tempo che intercorre tra la chiamata alla procedura *from-physical-layer()* e la fine della procedura *to-network-layer()*
- In questo intervallo di tempo, dati in arrivo saranno messi in **buffer**, in **attesa** di essere processati
- Poiché il tempo di elaborazione **non è nullo**, si deve gestire l'eventualità che i dati arrivino **troppo velocemente**

Controllo di flusso a priori

- Un semplice meccanismo puo' essere quello di **valutare i tempi di risposta** del ricevente, ed inserire dei **ritardi** nel processo di trasmissione per adattarlo alla capacita' di ricezione
- Il problema e' che il tempo di processamento in ricezione non e' una **costante** e puo' dipendere dal numero di linee che il nodo ricevitore deve gestire
- Basarsi sul **caso peggiore** comporta un grosso limite di **efficienza**
- Vedremo esempi di protocolli che implementano un controllo di flusso di **complessita' crescente** al fine di utilizzare al meglio la **banda**

Il frame data link

- Il **frame** data link prevede un'intestazione (**header**) e una coda (**trailer**) aggiunti al pacchetto passato dal livello di rete

Start flag	type	seq	ack	Pacchetto (livello rete)	Check sum	End flag
------------	------	-----	-----	--------------------------	-----------	----------

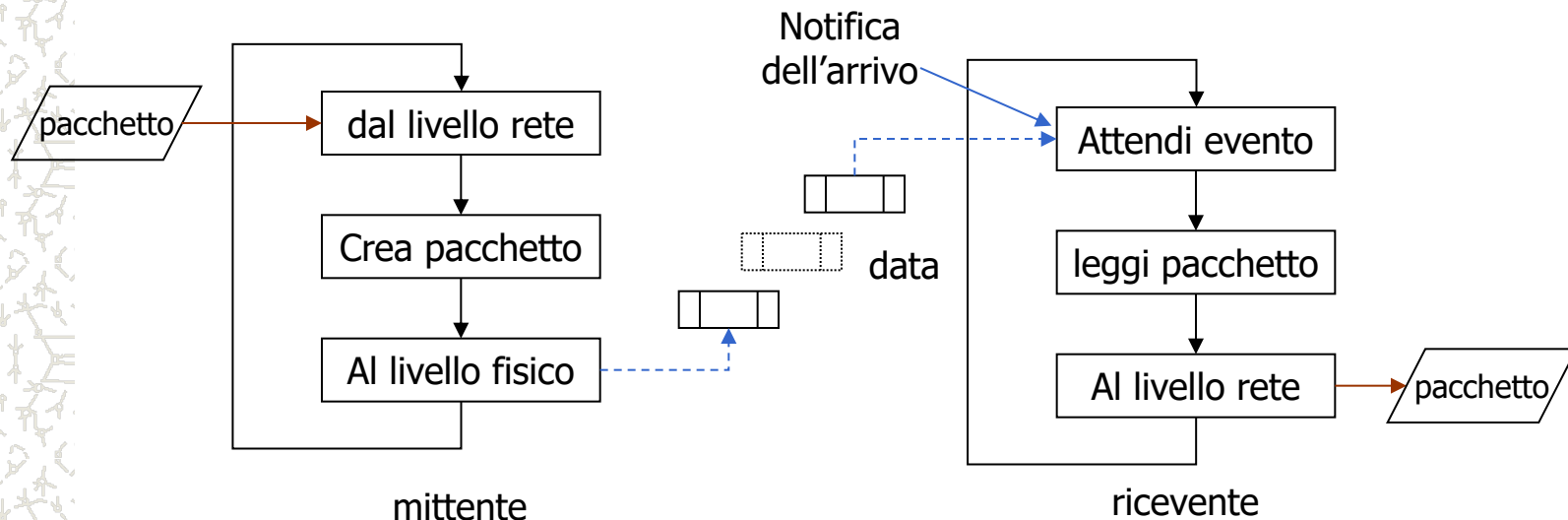
- Le informazioni di framing e di checksum sono gestite in hardware
- La presenza di campi di controllo dipende dal **protocollo** di comunicazione utilizzato nel livello data link

- **Tipo del pacchetto (**type**)** (es. data, ack, nack)
- **Numero di sequenza del pacchetto (**seq**)**
- **Numero di riscontro (**ack**)**

Protocolli data link

- Definizione della modalità di scambio dei pacchetti fra mittente e ricevente (ack, stop-and-wait, ritrasmissione,...)

Protocollo non limitato



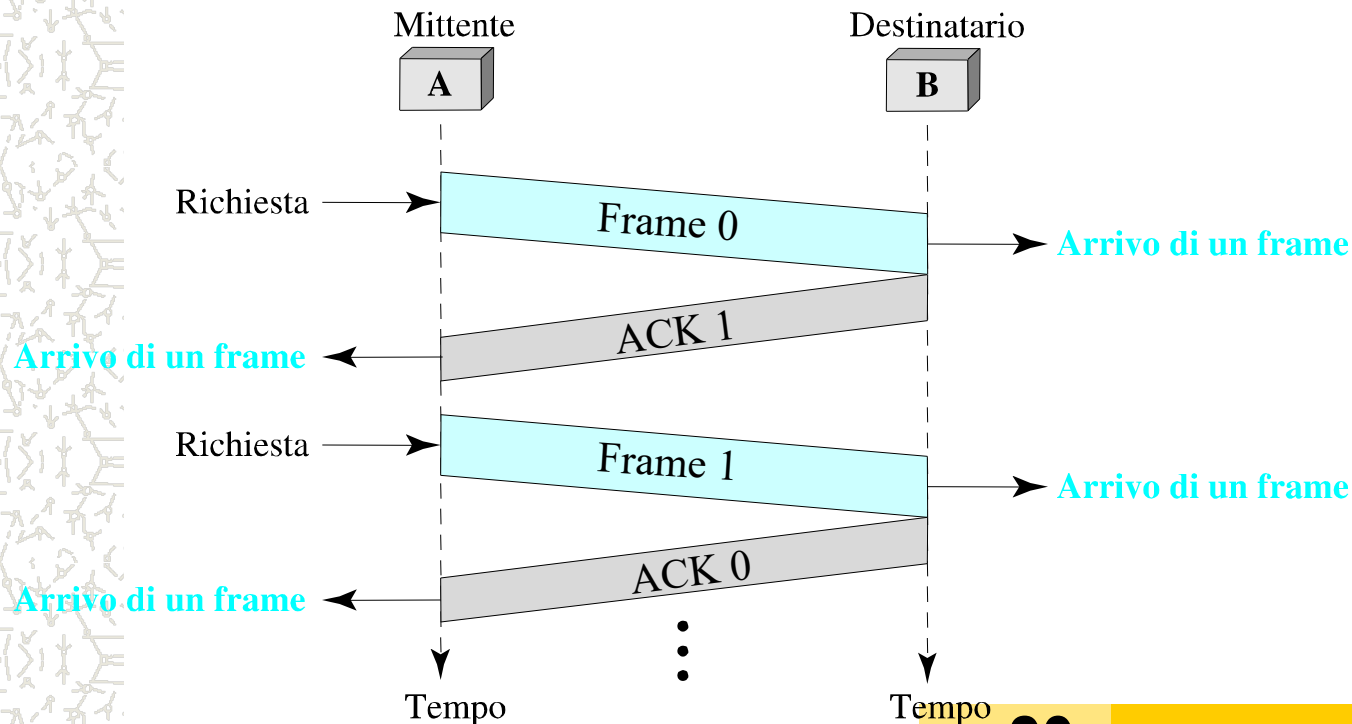
Protocollo senza conferma e senza connessione
I pacchetti possono andare persi o essere accettati anche se corrotti

Errori in trasmissione

- Il canale sottostante potrebbe confondere i bit nei pacchetti
- Possibilità di rilevare gli errori nei bit (controlli di parità e checksums)
- feedback del destinatario: messaggi di controllo (ACK, NAK) ricevente->mittente
 - per notificare lo stato della trasmissione si usa:
 - *notifica positiva (ACK)*: il ricevente comunica espressamente al mittente che il pacchetto ricevuto è corretto
 - *notifica negativa (NAK)*: il ricevente comunica espressamente al mittente che il pacchetto contiene errori
 - il mittente ritrasmette il pacchetto se riceve un NAK
- I protocolli che gestiscono tali meccanismi sono conosciuti come Protocollo PAR (Positive Acknowledgement with Retransmission) o anche ARQ (Automatic Repeat reQuest)

Riscontri

- La conferma di avvenuta trasmissione avviene attraverso l'invio di una trama di riscontro ACK
- L'identificativo della trama riscontrata fa riferimento al suo sequence number:
 - I numeri di sequenza nei frame di ACK annunciano il prossimo frame che il destinatario si aspetta di ricevere



Problemi

Perdita dei riscontri

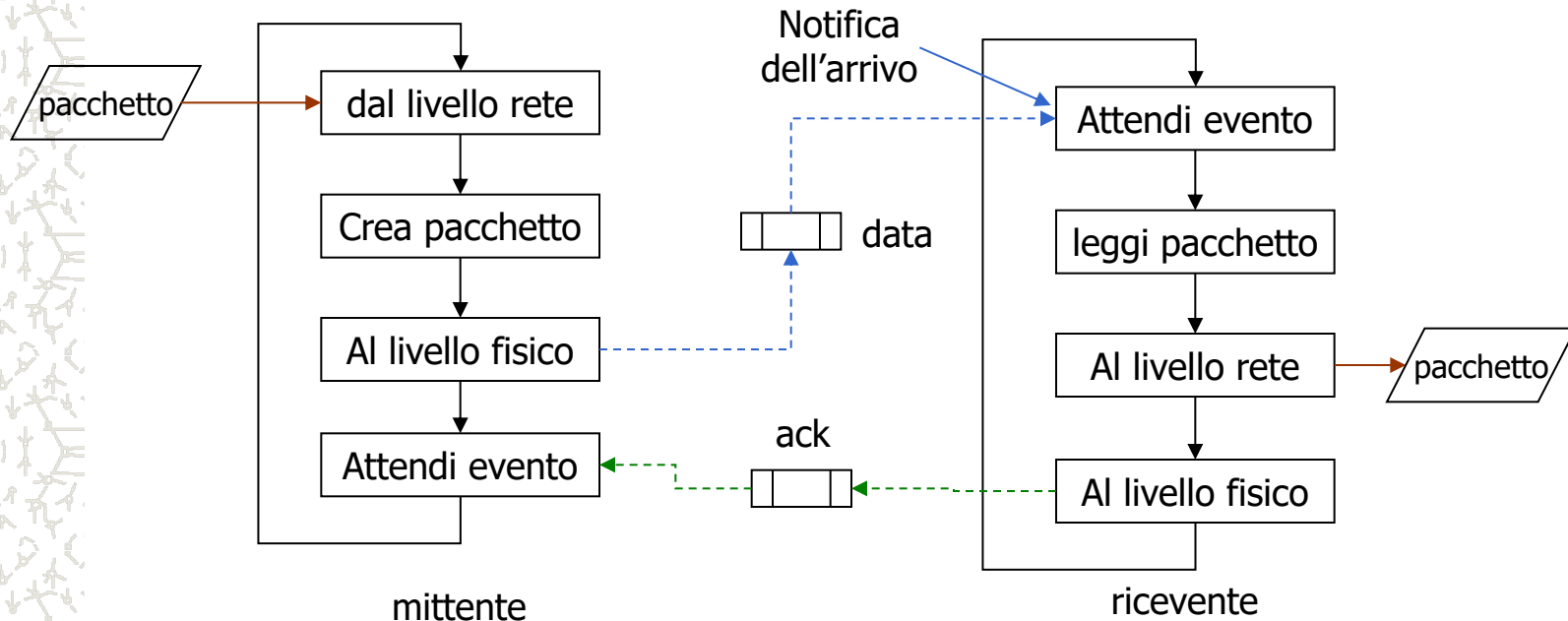
- Che cosa accade se i pacchetti ACK/NAK vengono danneggiati?
 - Il mittente non sa che cosa sia accaduto al destinatario!
 - Non sa se il destinatario ha ricevuto correttamente i dati trasmessi
- Una possibile soluzione:
 - Bisogna aggiungere il checksum a ACK/NAK
 - Il sender reinvia i pacchetti quando riceve un ACK/NAK alterato

Duplicazione dati

- Il sender reinvia i pacchetti quando riceve un ACK/NAK alterato
- Questo metodo introduce duplicati dei pacchetti nel canale tra sender e receiver
- Soluzione:
 - Si aggiunge un nuovo campo ai pacchetti dati: Numero di Sequenza
 - Il receiver deve solo controllare questo numero per comprendere se il pacchetto che gli è arrivato è nuovo o un duplicato.

Protocollo ARQ stop-and-wait

- Sincronizzazione della trasmissione con ack



Evita il flooding del ricevente

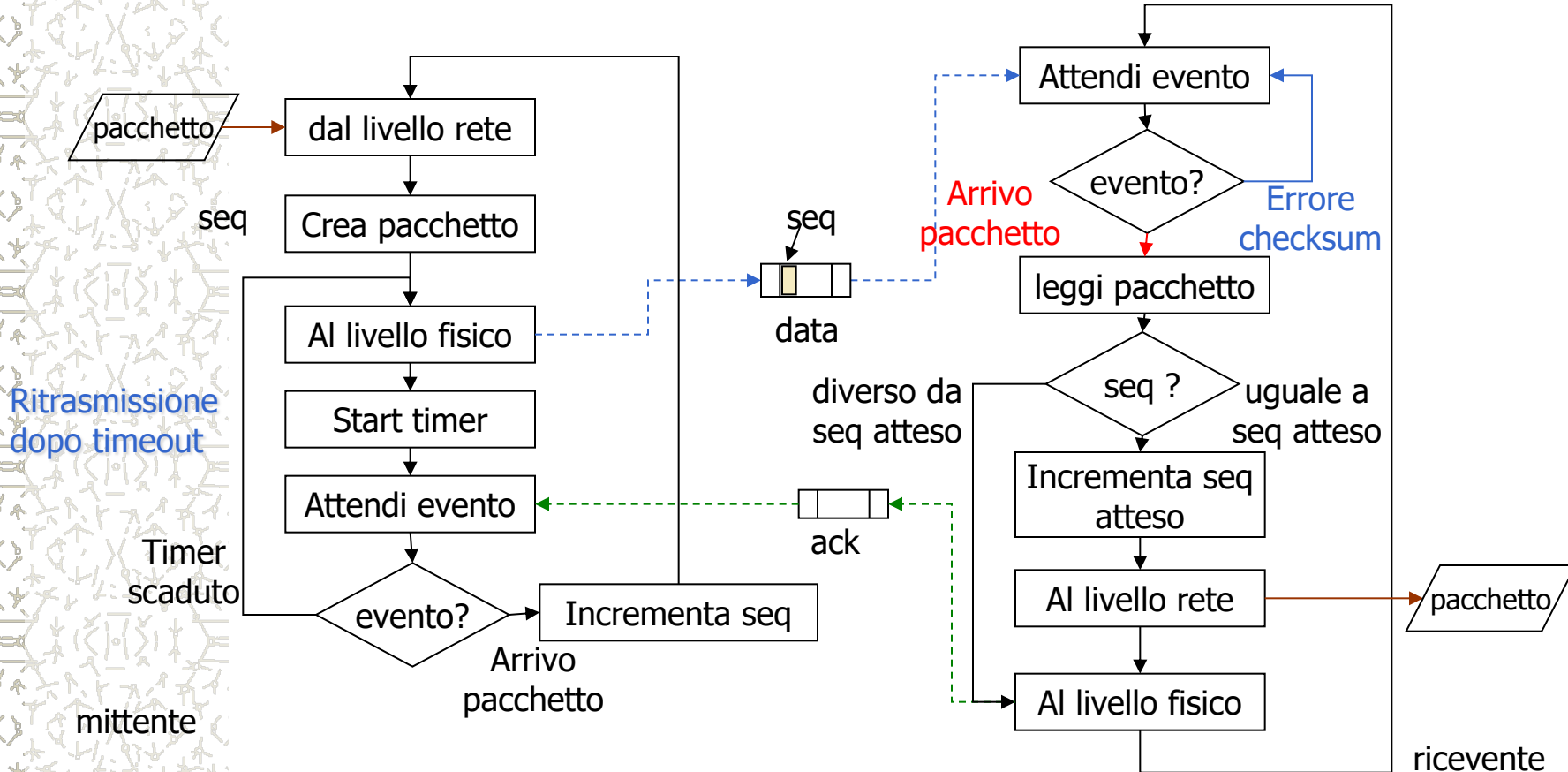
Funziona solo se c'è garanzia dell'arrivo dei pacchetti di ack

Protocollo stop-and-wait

- Ipotizziamo che
 - il canale sia **privo di errori**
 - il traffico dati scorra in **una direzione sola**, dal trasmittente (A) al ricevente (B), cioè' protocollo **simplex**
- Il protocollo **stop-and-wait** prevede che A, dopo aver inviato il frame, si **fermi** per attendere un **riscontro**
- B, una volta **ricevuto il frame**, inviera' ad A un **frame di controllo**, cioè' un frame privo di dati, allo scopo di avvisare A che **puo'** **trasmettere** un nuovo frame
- Il frame di riscontro di indica generalmente con il termine ACK (ACKnowledge) o RR (Receiver Ready)
- Va osservato che il **traffico dati e' simplex**, ma i frame devono viaggiare nelle **due direzioni**, quindi il canale fisico deve essere almeno half-duplex

Protocollo ARQ con timeout

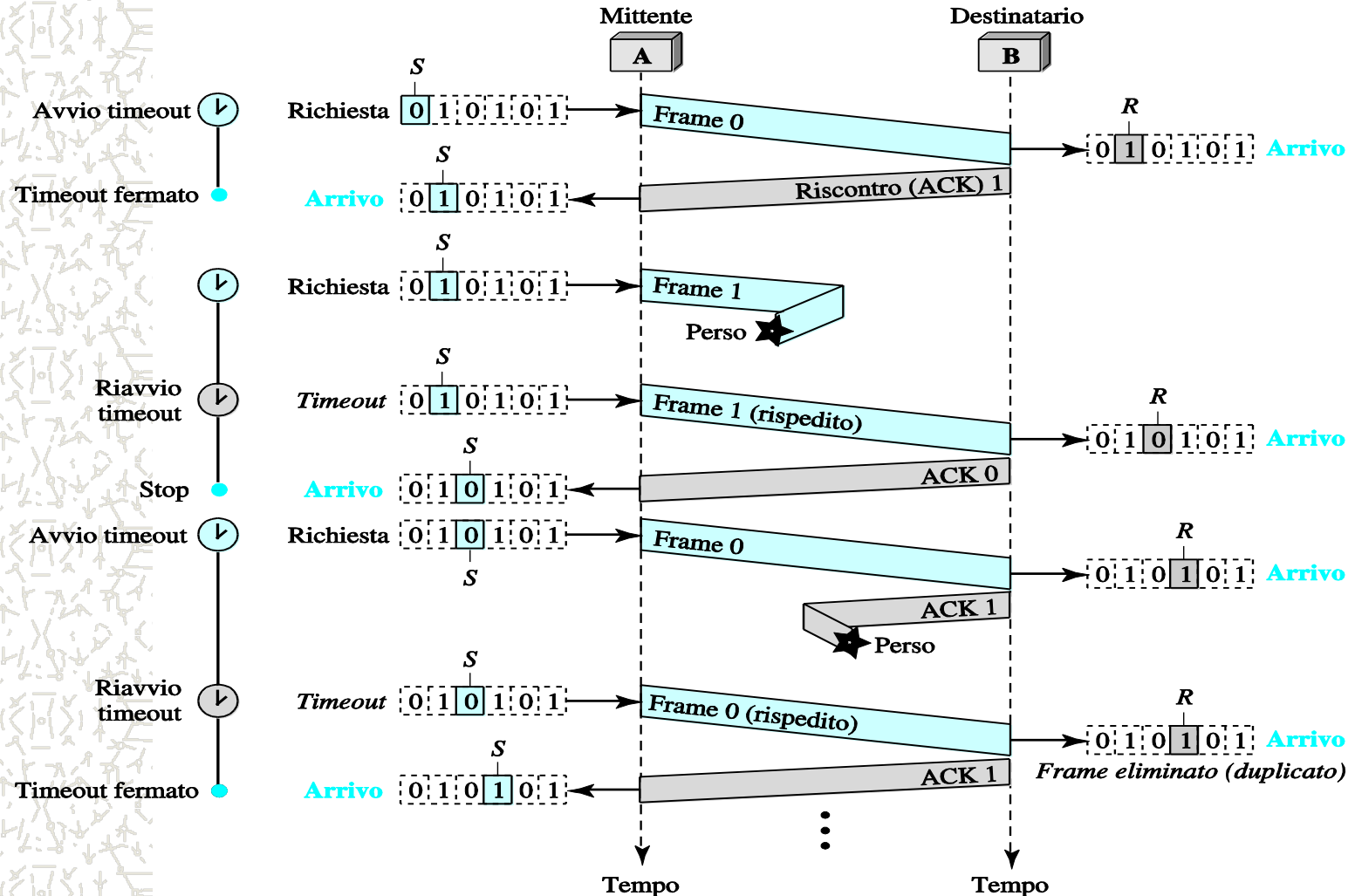
- Gestisce il caso di canali disturbati (perdita pacchetti)



Protocollo ARQ con timeout [continua]

- il mittente attende un ACK per un tempo “ragionevole” e ritrasmette se non riceve un ACK in questo periodo
 - se il pacchetto (o l'ACK) è soltanto in ritardo (non perso): la ritrasmissione sarà duplicata, ma l'uso dei numeri di sequenza gestisce già questo
 - occorre un contatore (countdown timer)
- Problema:
 - Quando il sender riceve un ACK, non sa se si riferisce al pacchetto spedito più di recente o si tratta di un ACK arrivato in ritardo.
- Soluzione:
 - il receiver deve specificare il numero di sequenza del pacchetto da riscontrare
 - Si inserisce nel pacchetto ACK un campo di riscontro (contenente il num di seq. del pacchetto dati ricevuto)
 - Il sender esaminando questo campo può individuare il pacchetto oggetto del riscontro

Protocollo ARQ con timeout [continua]



Prestazioni

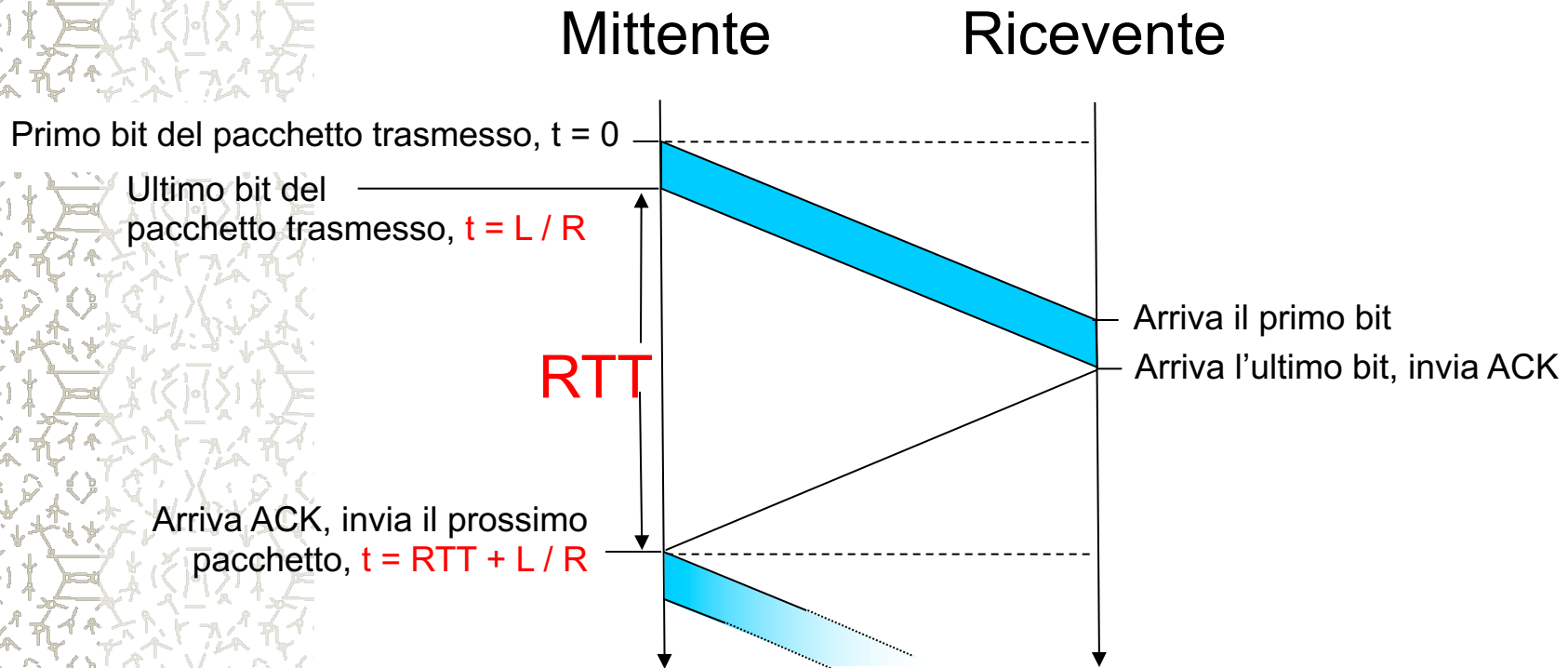
Esempio: collegamento da 1 Gbps, ritardo di propagazione 15 ms, pacchetti da 1 KB:

$$T_{\text{trasm}} = \frac{L \text{ (lunghezza del pacchetto in bit)}}{R \text{ (tasso trasmissivo, bps)}} = \frac{8 \text{ kb}}{10^9 \text{ b/sec}} = 8 \text{ microsec}$$

$$U_{\text{mitt}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

- U_{mitt} : **utilizzo** è la frazione di tempo in cui il mittente è occupato nell'invio di bit
- Un pacchetto da 1 KB ogni 30 msec -> throughput di 33 kB/sec in un collegamento da 1 Gbps
- Il protocollo di rete limita l'uso delle risorse fisiche!

Prestazioni: funzionamento con stop-and-wait



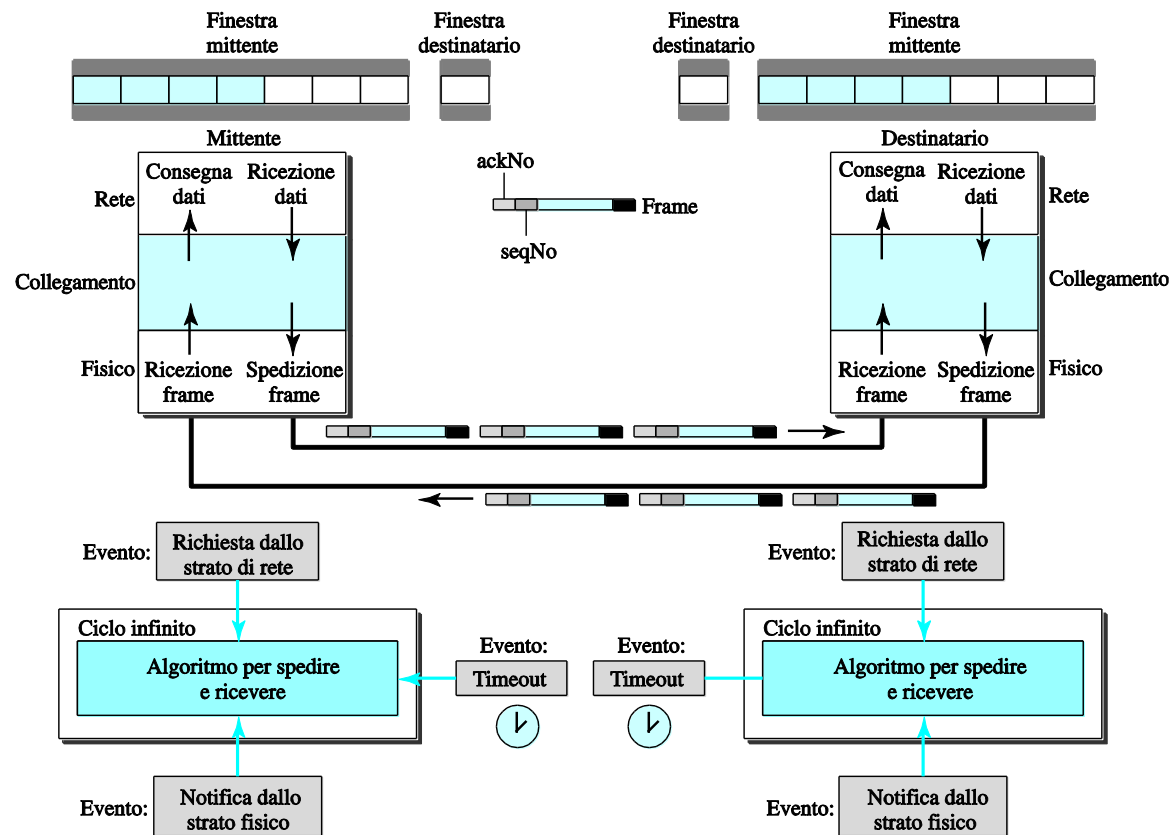
$$U_{\text{mitt}} = \frac{L / R}{RTT + L / R} = \frac{0,008}{30,008} = 0,00027$$

Trasmissioni full duplex

- Quando il canale di comunicazione permette l'invio di dati in **entrambe** le direzioni **contemporaneamente** e' possibile definire protocolli di comunicazione detti **full duplex**
- In caso di linea full duplex il canale trasmette **frame di dati** in un verso e **frame di ACK** relativi alla comunicazione nel verso opposto, **mischiati** tra loro
- I frame saranno **distinti** da una informazione contenuta nell'header del frame, che etichetta i frame come "**dati**" o come "**frame di controllo**"

Acknowledge in piggybacking

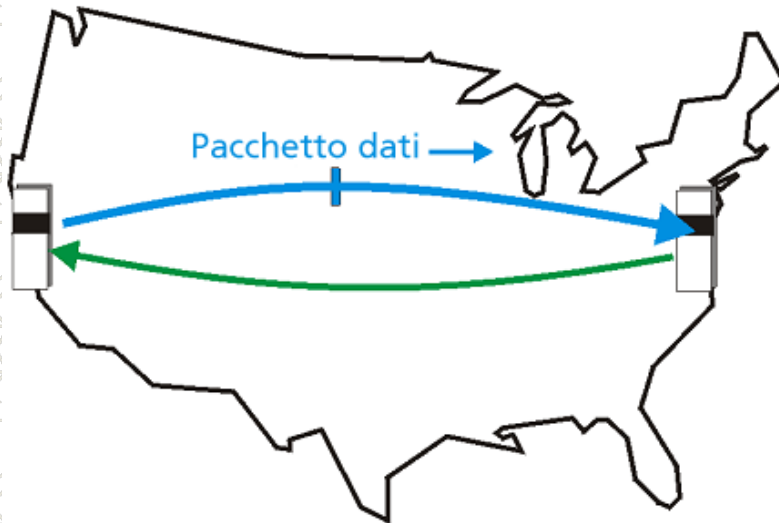
- Per motivi di **efficienza** spesso si utilizza una tecnica (detta “**piggybacking**”) per evitare di dover costruire e trasmettere un frame di ACK:
 - si dedica un campo **dell'header** di un frame di dati per trasportare l'ACK in senso **inverso**
- Quando si deve trasmettere un ACK, **si aspetta** di dover trasmettere un frame di dati che possa trasportare l'informazione di ACK
- Se non ci sono dati da inviare, si dovrà **comunque** inviare un frame di ACK **prima** che scada il timeout del trasmittente
 - questo implica il dover utilizzare **un altro timer** per decidere dopo quanto tempo inviare **comunque** l'ACK in caso di mancanza di dati da inviare in senso inverso



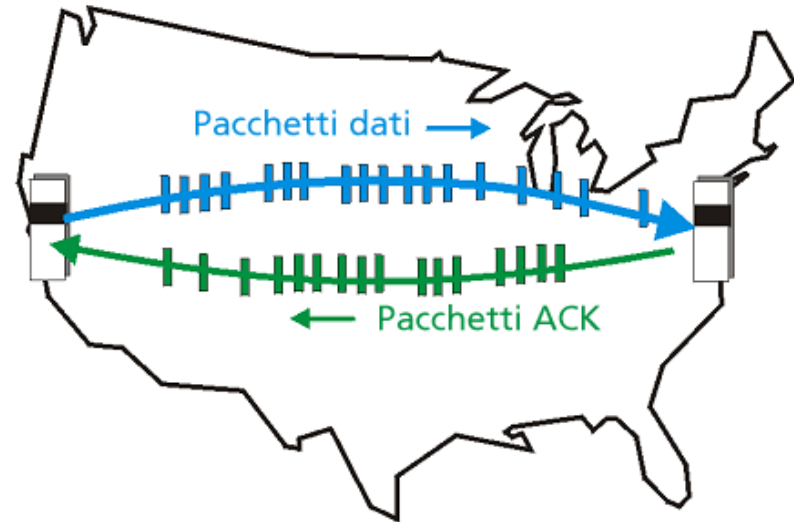
Protocolli con pipeline

Pipelining: il mittente ammette più pacchetti in transito, ancora da notificare

- l'intervallo dei numeri di sequenza deve essere incrementato
- buffering dei pacchetti presso il mittente e/o ricevente



a) Protocollo stop-and-wait all'opera



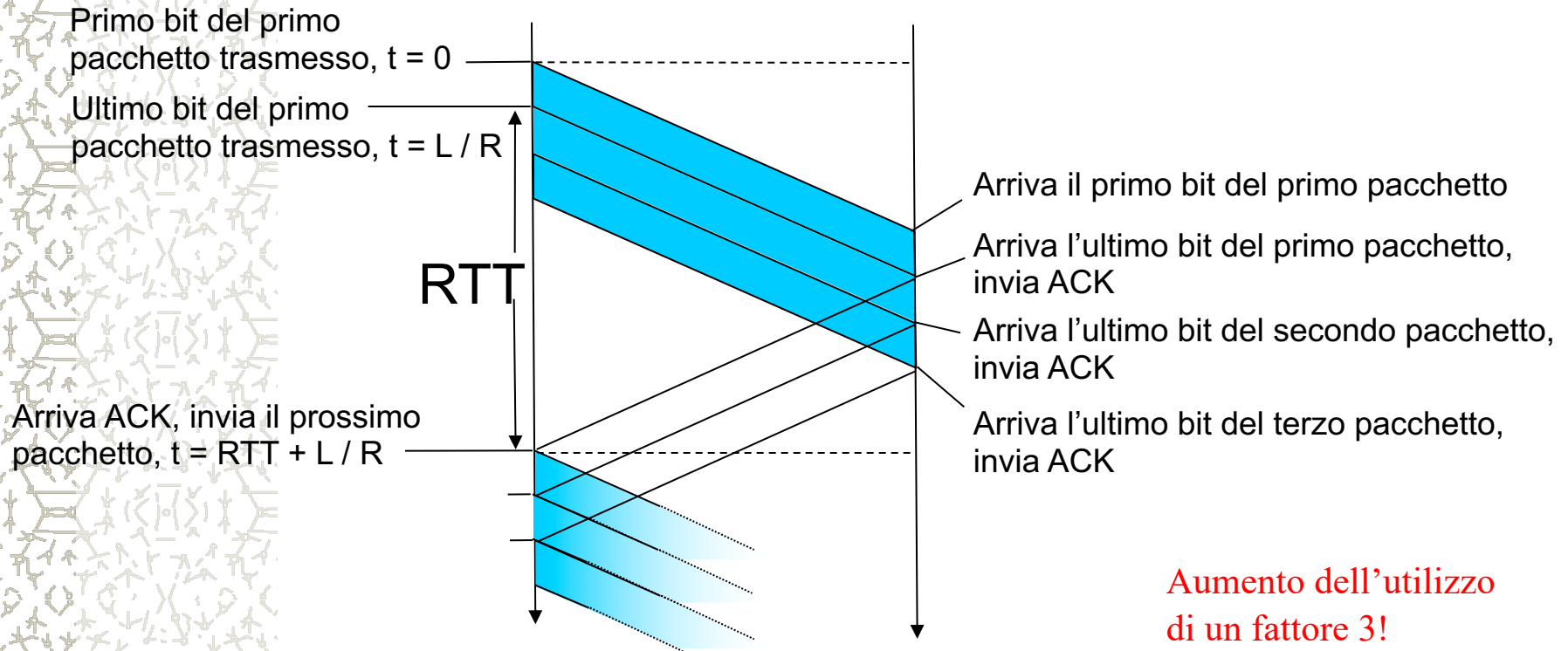
b) Protocollo con pipeline all'opera

- Due forme generiche di protocolli con pipeline:

Go-Back-N e *ripetizione selettiva*

Pipelining: aumento dell'utilizzo

Mittente Ricevente



Aumento dell'utilizzo
di un fattore 3!

$$U_{\text{mitt}} = \frac{3 * L / R}{RTT + L / R} = \frac{0,024}{30,008} = 0,0008 \text{ microsec}$$

Protocolli a finestra scorrevole

- I protocolli a finestra scorrevole (**sliding window**) permettono di inviare **piu' di un frame** prima di fermarsi per attendere il riscontro, fino ad un **valore massimo W** fissato a priori
- Poiche' in ricezione possono arrivare piu' frame consecutivi, i frame devono essere **numerati** per garantire in ricezione che non si siano persi frame: saranno dedicati **n bit** di controllo per la numerazione, ed i frame potranno avere numero **da 0 a 2^n-1**
- In ricezione non e' necessario riscontrare **tutti i frame**: il ricevente puo' attendere di ricevere un certo numero di frame (fino a W) prima di inviare un solo riscontro **cumulativo**
- La numerazione dei frame e' in **modulo 2^n** , cioe' il frame successivo a quello numerato 2^n-1 avra' come identificativo il numero **0**
- Per non avere **sovrapposizione** dei numeri identificativi tra i frame **in attesa di riscontro**, questi non dovranno essere in numero maggiore di **2^n** , quindi si avra' sempre **$W \leq 2^n$** ; in funzione del protocollo usato si potranno avere **restrizioni maggiori**

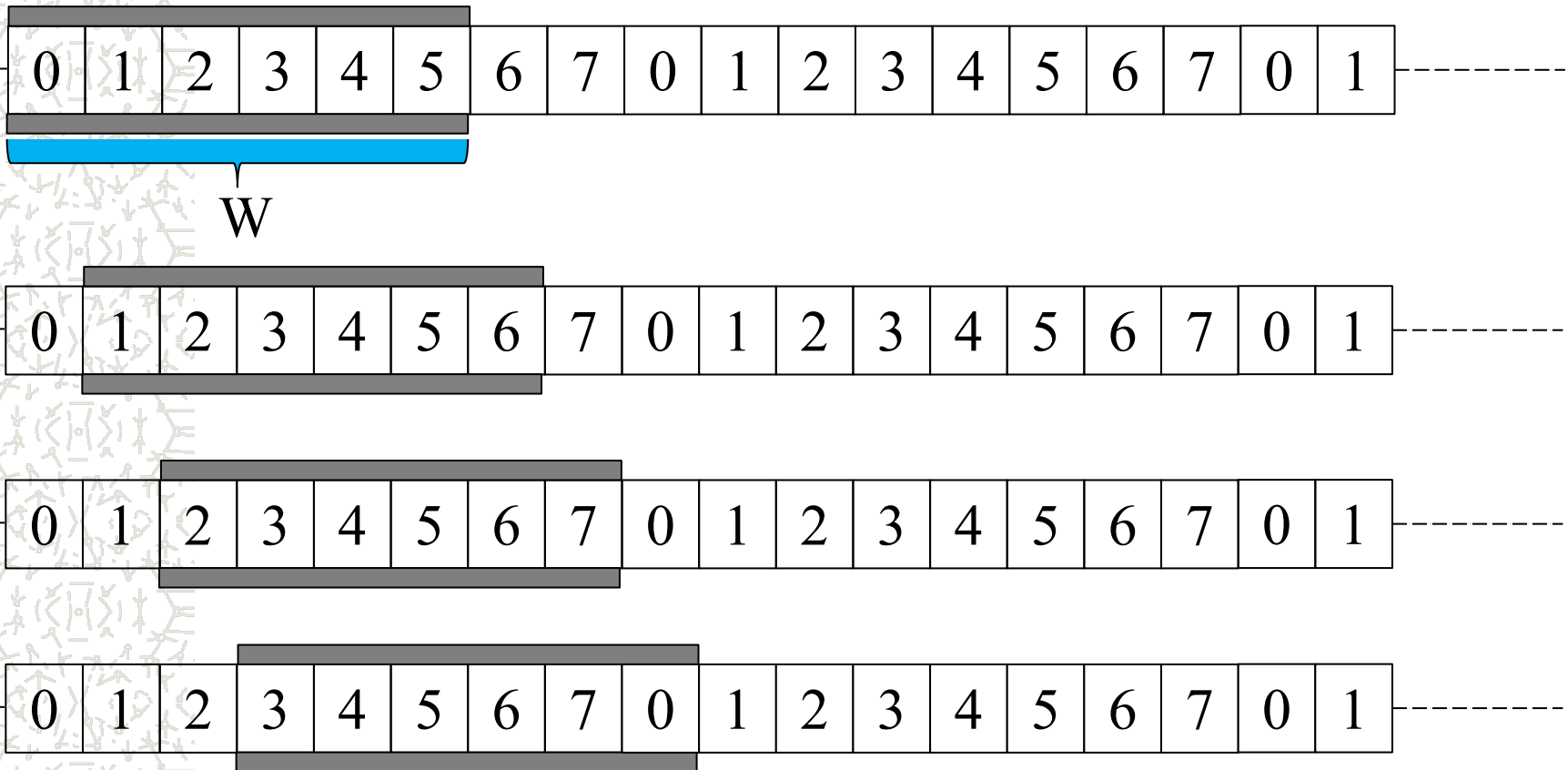
Protocolli a finestra scorrevole (cont.)

- Questo tipo di protocolli necessita' di **maggiori** risorse di buffer:
 - in trasmissione devono essere **memorizzati** i frame inviati in attesa di riscontro, per poterli **ritrasmettere** in caso di necessita'
 - ad ogni riscontro ricevuto, vengono **liberati** i buffer relativi ai frame riscontrati, per occuparli con i **nuovi frame** trasmessi
 - a seconda del protocollo anche in **ricezione** si deve disporre di buffer, ad esempio per memorizzare frame **fuori sequenza**;
 - ad ogni **riscontro inviato**, i frame riscontrati vengono **passati** allo strato di rete ed i relativi buffer vengono liberati per poter accogliere **nuovi frame in arrivo**
 - maggiore **complessita'** di calcolo
- La dimensione della finestra (W) puo' essere **fissata a priori** dal protocollo, ma esistono protocolli che permettono di modificarne il valore **dinamicamente** tramite **informazioni di controllo** del protocollo
- Abbiamo una finestra in trasmissione (finestra **mittente**) e una in ricezione (finestra **destinatario**)

Finestra scorrevole

$n=3 \Rightarrow 2^3=8$ Possibili numeri di sequenza: da 0 a 7 (da 0 a 2^n-1)

$W=6$ dimensione della finestra



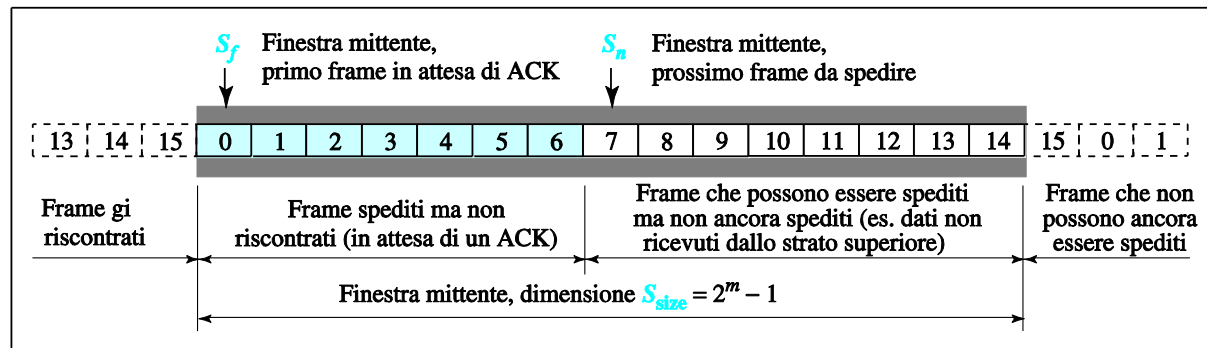
Protocolli a finestra scorrevole

- Quanto deve essere grande m ?
 - Vogliamo m piccolo per essere efficienti
 - Occupare poco spazio nel frame
 - Vogliamo m grande per non creare ambiguità fra i frame
- Per stop-and-wait
 - Basta $m=1$
 - Dobbiamo essere capaci solo di distinguere un frame dal successivo
 - Numerazione dei frame: 0,1,0,1,0,1,0,1,...

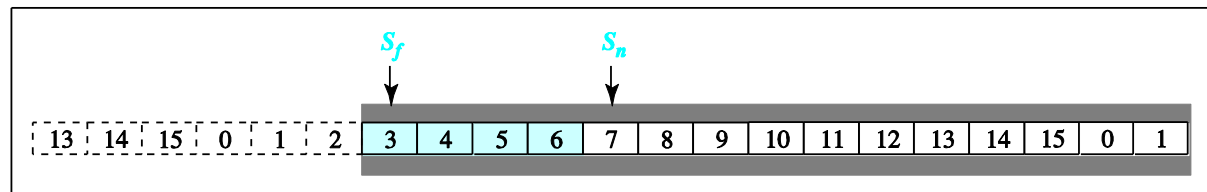
Per il protocollo stop-and-wait utilizziamo dei numeri di sequenza per identificare i frame. Bastano due numeri di sequenza, 0 e 1, usati in alternanza.

La finestra in trasmissione

- In trasmissione si deve tenere conto dei frame **inviati e non riscontrati**, e del numero massimo di frame che **possono** essere ancora inviati prima di doversi fermare
- Si utilizza una **sequenza** di numeri (finestra), di dimensioni $\leq W$ indicanti gli identificativi dei frame che il trasmittente e' **autorizzato ad inviare**
- Con il procedere della trasmissione la finestra **scorre in avanti**:
 - inizialmente la finestra ha limiti **0 e W-1**
 - ad ogni frame **inviato**, il limite inferiore della finestra **cresce di una unita'**;
 - quando la finestra si **chiude** (sono stati inviati W frame in attesa di riscontro) l'invio **deve fermarsi**
 - per ogni frame **riscontrato**, il limite superiore della finestra si **sposta in avanti** di una unita' (o piu' in caso di riscontri **cumulativi**), permettendo al trasmittente di inviare **nuovi frame**



a. Finestra mittente prima dello scorrimento

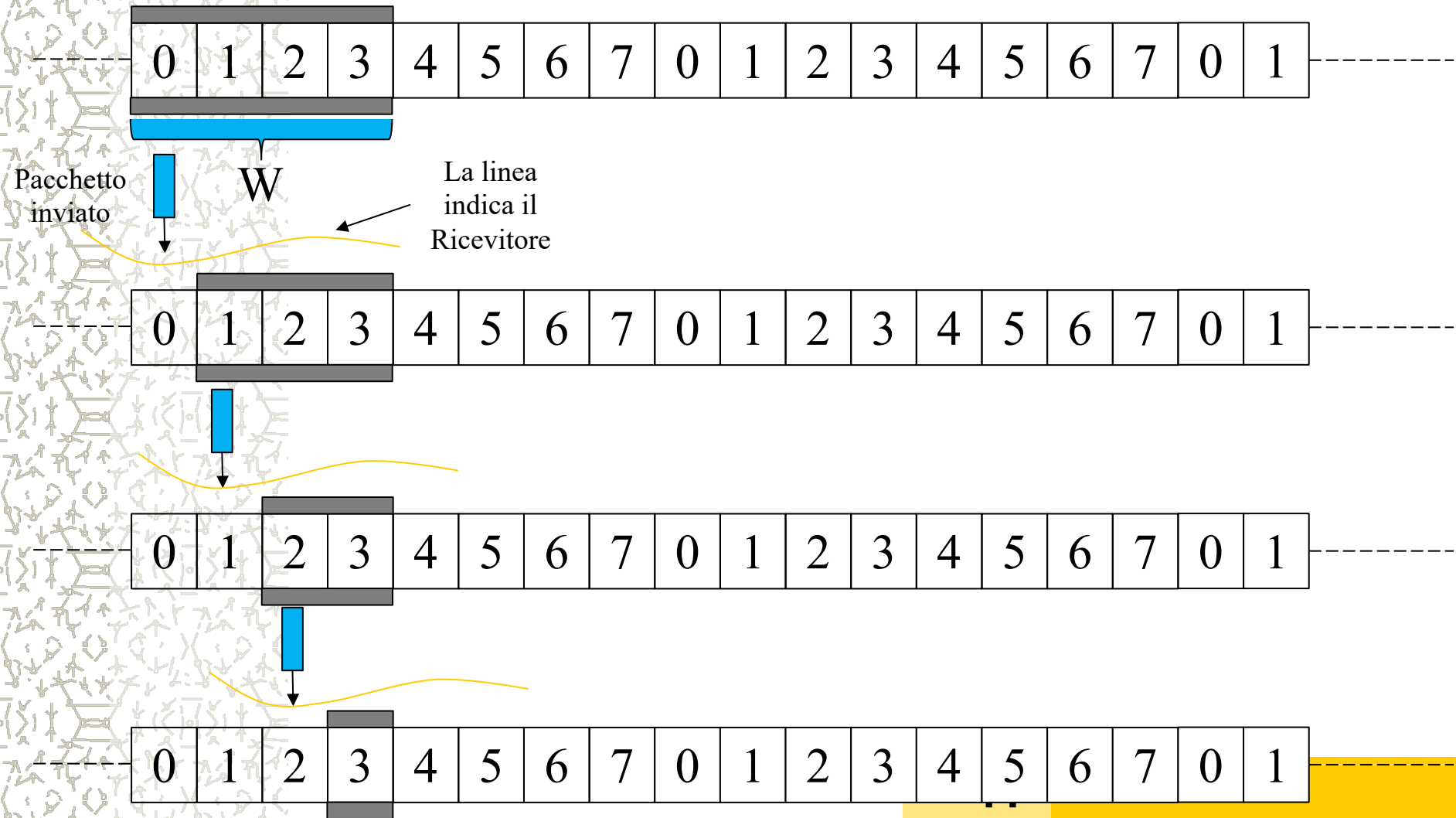


b. Finestra mittente dopo lo scorrimento

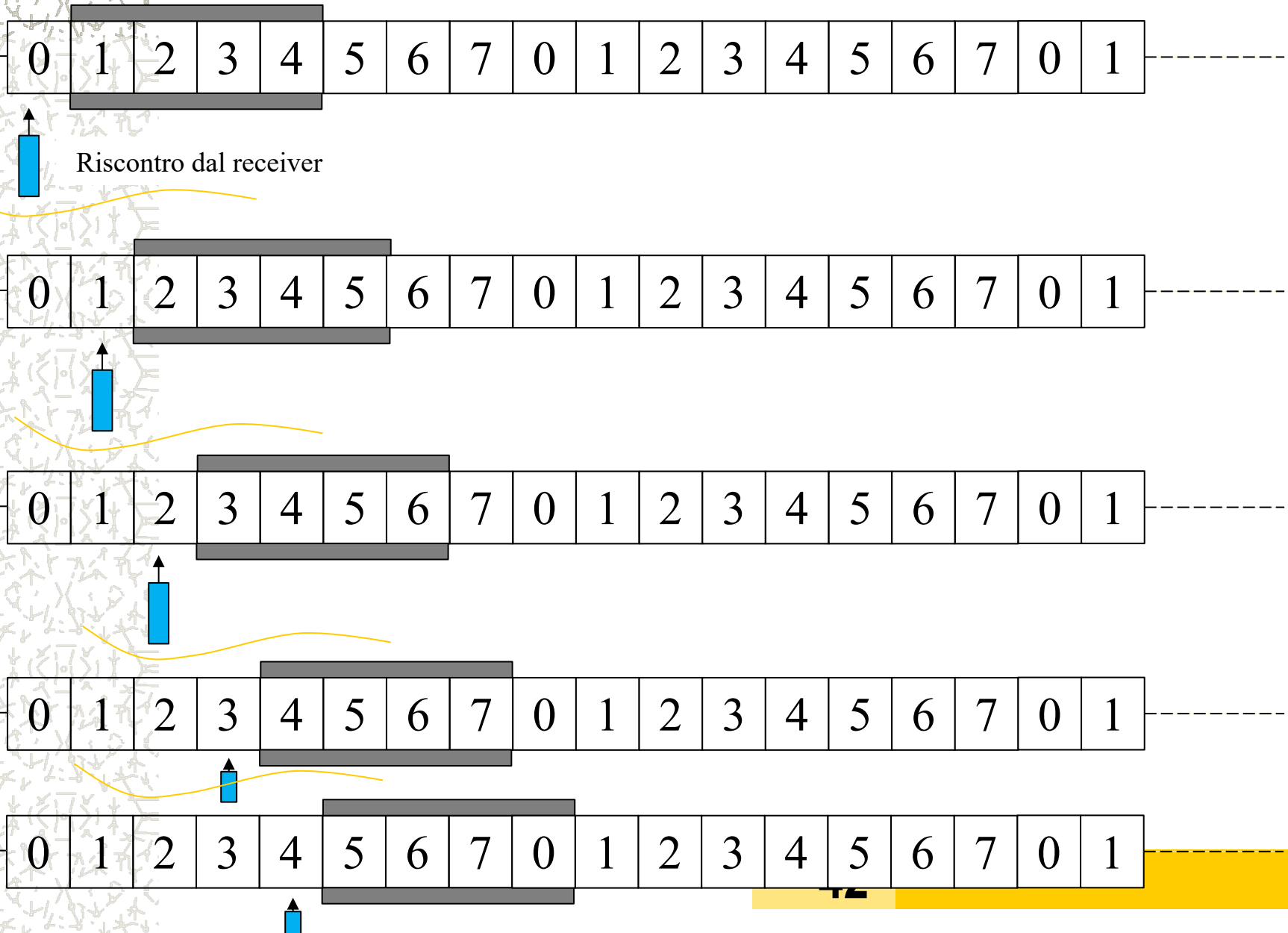
Protocolli a finestra scorrevole: trasmissione

$m=3 \Rightarrow 2^3=8$ Possibili numeri di sequenza: da 0 a 7 (da 0 a 2^m-1)

$W=4$ dimensione della finestra

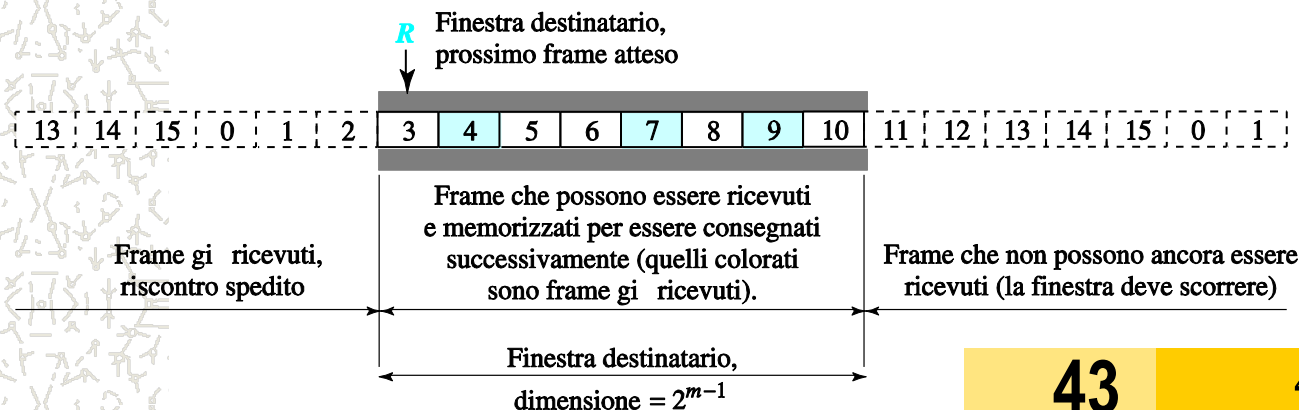


Protocolli a finestra scorrevole: trasmissione

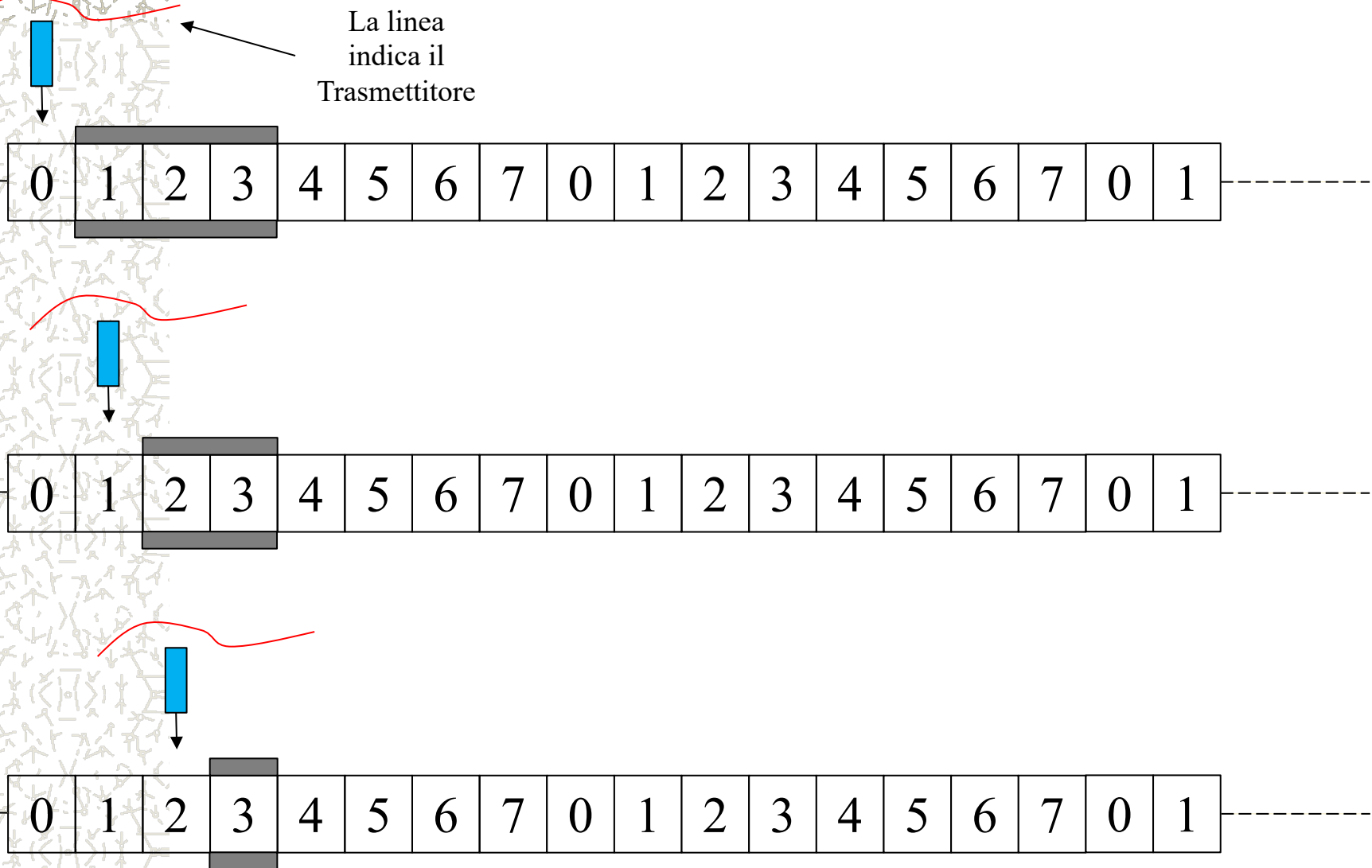


La finestra in ricezione

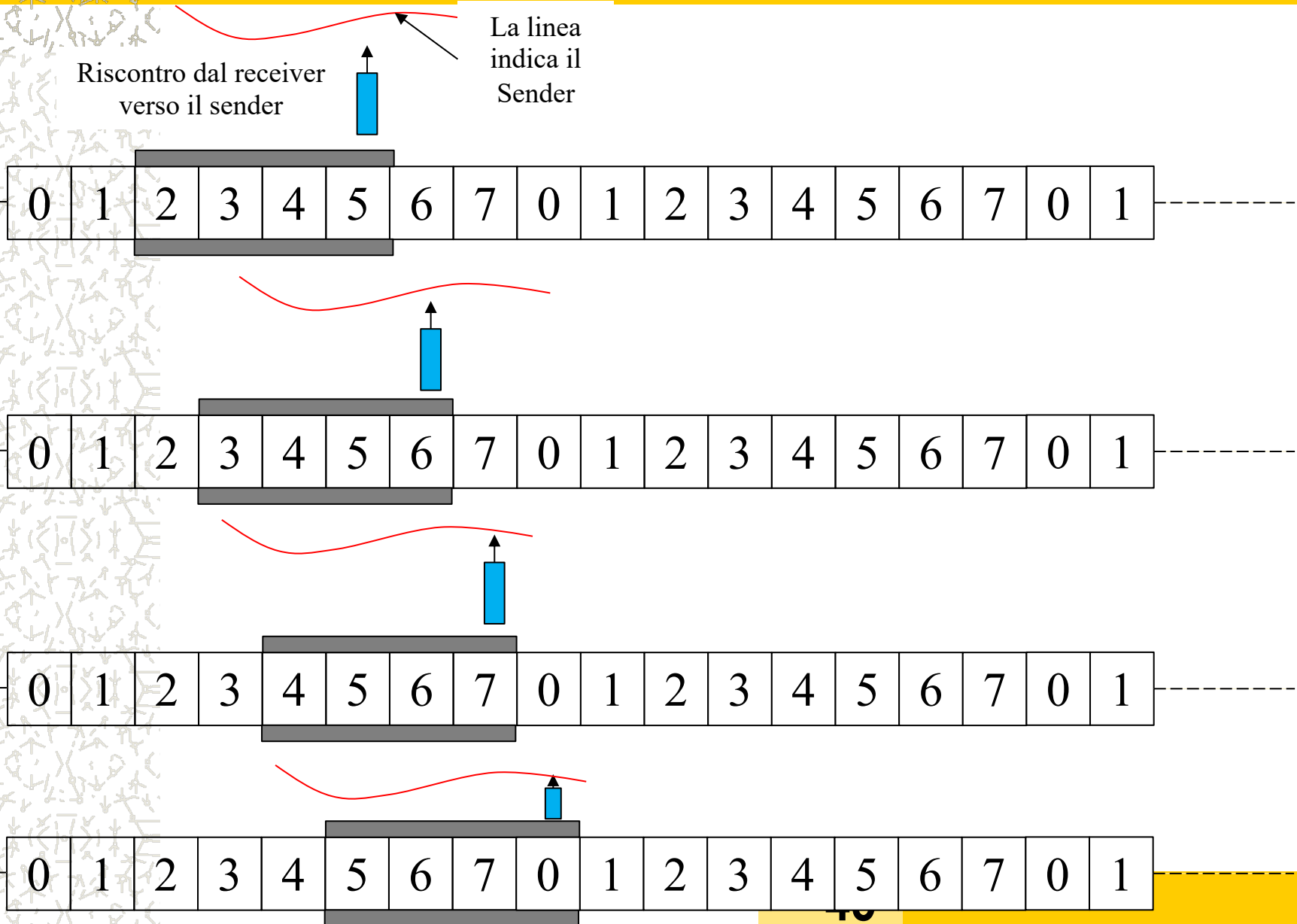
- In ricezione si deve tenere conto dei frame **ricevuti** di cui **non e'** stato ancora inviato l'**ACK**, e del numero di frame ancora **accettabili**
- Si utilizza una finestra analoga a quella in ricezione che contiene i numeri dei frame **accettabili**
- Il limite inferiore e' il numero del frame **successivo all'ultimo ricevuto**, mentre il limite superiore e' dato dal **primo non ancora riscontrato piu' W**
- Ad ogni nuovo frame **ricevuto** il limite inferiore della finestra **cresce** di una unita', mentre ad ogni **acknowledge inviato** il limite superiore **avanza** di una unita'
- La dimensione della finestra non puo' **eccedere** il valore di W (tutti i frame ricevuti sono riscontrati)
- Quando la finestra si azzerata si devono **per forza** inviare i riscontri, perche' la ricezione e' **bloccata**
- Qualsiasi frame ricevuto con numero **fuori dalla finestra di ricezione** sara' **scartato**
- La finestra in ricezione non deve **necessariamente** avere la **stessa dimensione** della finestra in trasmissione
- ad esempio una finestra in ricezione piu' piccola costringera' il ricevente ad inviare ACK **prima** che in trasmissione sia stata azzerata la finestra



Protocolli a finestra scorrevole: ricezione

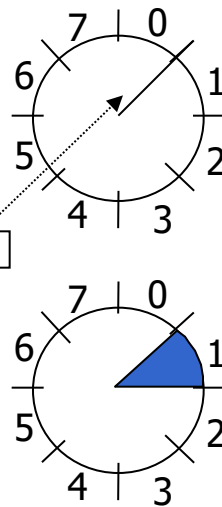
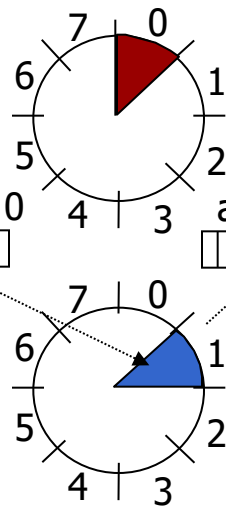
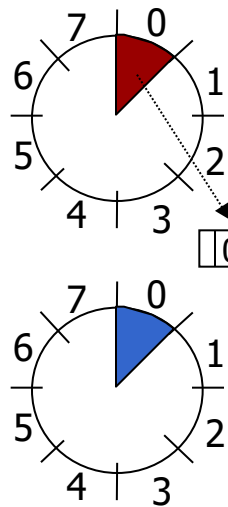
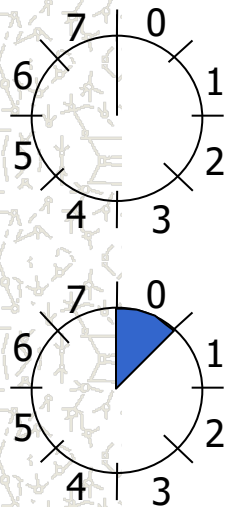


Protocolli a finestra scorrevole: trasmissione



Protocolli a finestra scorrevole

- Ogni frame spedito contiene un numero progressivo a n bit
- La **finestra di trasmissione** del mittente corrisponde ai frame che può ritrasmettere (frame ancora senza ack)
- La **finestra di ricezione** del ricevente indica i frame che può accettare (pacchetti attesi che non sono stati ancora ricevuti)

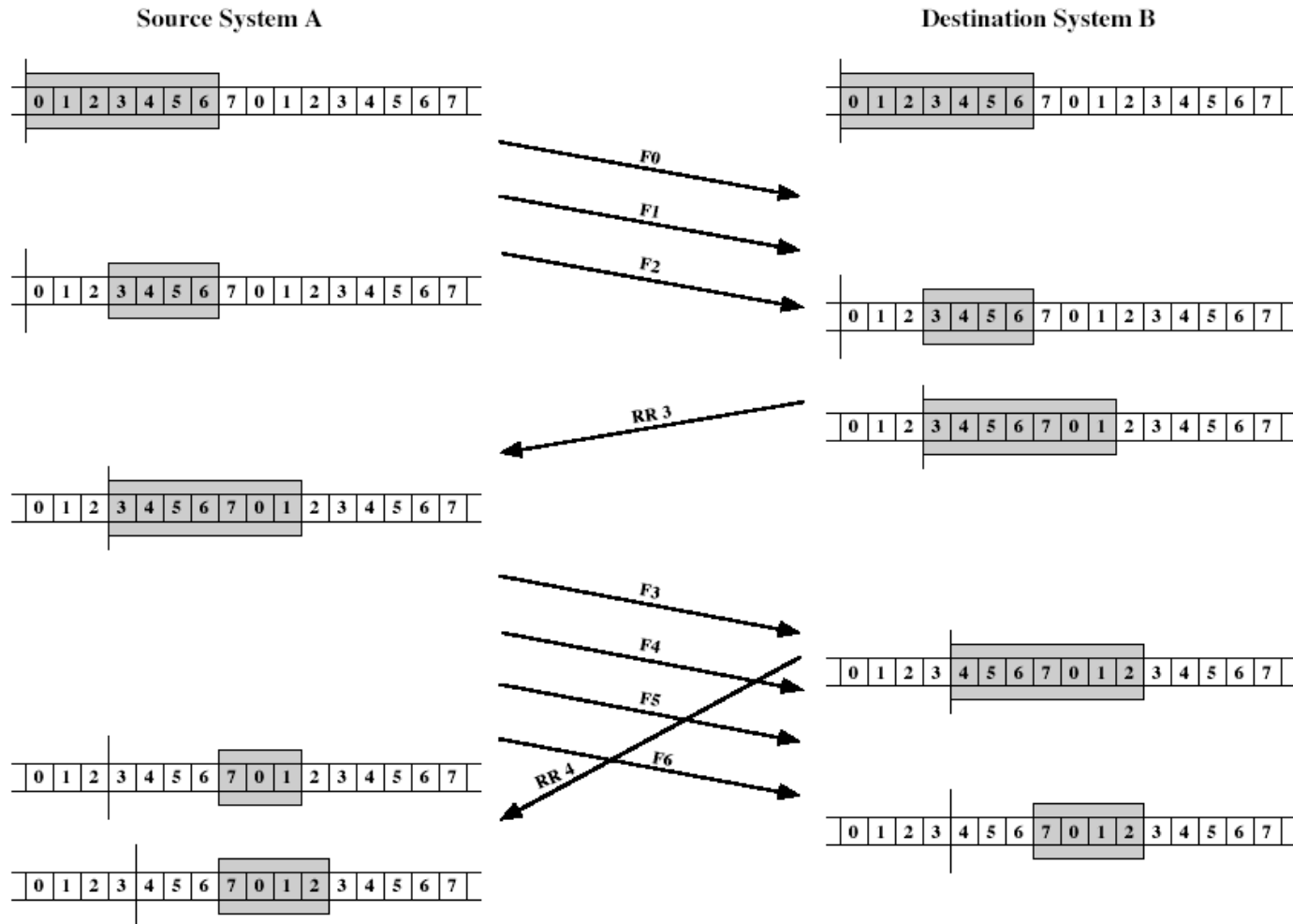


mittente
ack atteso

$W = 1$

ricevente
frame atteso

Protocolli a finestra scorrevole



Protocolli a finestra scorrevole



I numeri di sequenza sono limitati

È necessario riutilizzare i numeri di sequenza

La finestra risolve il problema

https://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198702.cw/index.html

<https://www.youtube.com/watch?v=9BuaeEjleQI>

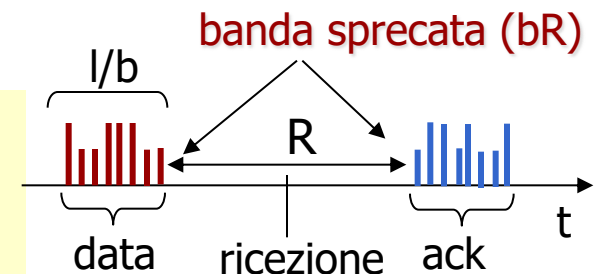
Protocolli a finestra scorrevole [continua]

- Il mittente mantiene in un buffer di dimensione w tutti i frame nella finestra nel caso debbano essere ritrasmessi
- Quando il buffer è pieno (non si sono ricevuti ack) il livello data link del mittente non accetta più pacchetti dal livello di rete
- Se $w=1$ si ha un protocollo stop-and-wait (si aspetta l'ack prima di spedire un nuovo frame)
- Il protocollo stop-and-wait spreca banda per le attese in caso di mezzo fisico con ritardo non trascurabile

velocità del canale = b bps

dimensione del frame = l bit

tempo del ciclo = R s (tempo di propagazione)



Protocolli sliding windows con errori

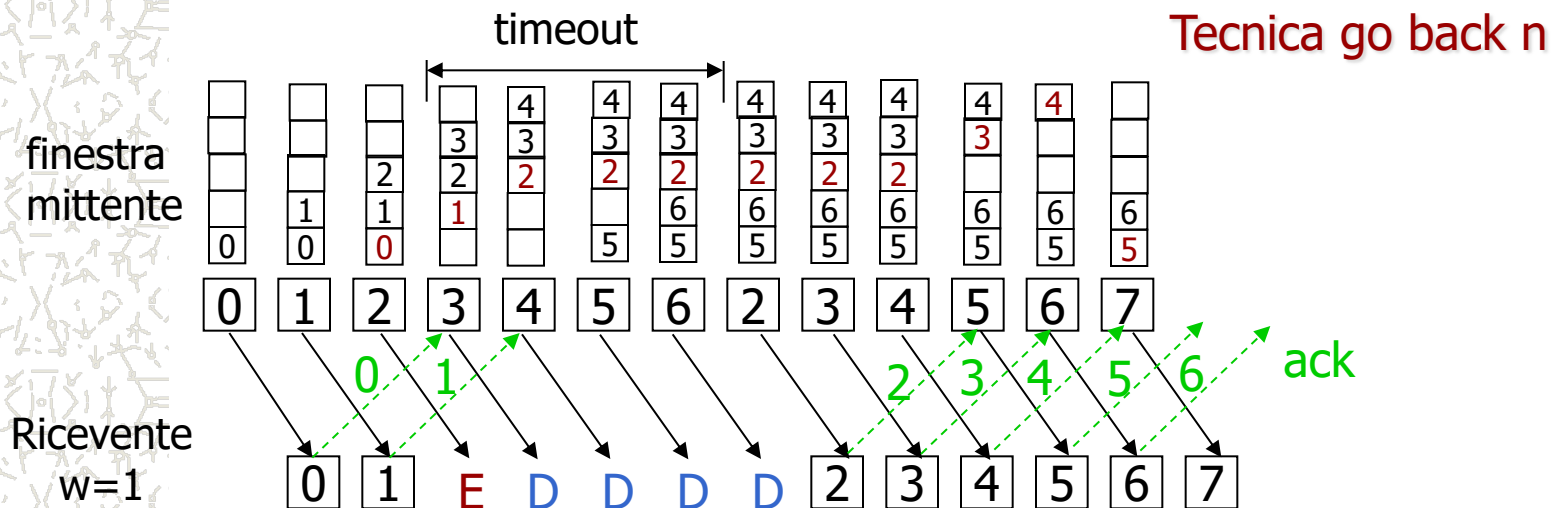
- L'utilizzo di un protocollo sliding window permette di utilizzare **meglio** la linea, ma **complica** il problema di gestire gli errori:
 - il trasmittente, **prima di accorgersi** che un frame e' stato ricevuto con errore, ha gia' inviato **altri frame**
 - in ricezione possono quindi arrivare frame corretti con numero di sequenza **successivo** ad un frame rigettato (non ricevuto)
- Esistono **due protocolli** che gestiscono in modo differente questa situazione:
 - protocollo **go-back-N**
 - protocollo **selective reject**

Protocolli sliding windows con errori

- Questi protocolli prevedono l'invio sia di frame **ACK** (per riscontrare un frame), indicati **anche** come **RR** (Receiver Ready), che **NAK** (**Not Acknowledged**), indicato anche come **REJ** (REject), utilizzato per informare il trasmittente che e' stato ricevuto un frame **fuori sequenza**
- Sia gli ACK (RR) che i REJ riportano l'indicazione del numero di sequenza del frame che **e' atteso** in ricezione (quello successivo all'ultimo riscontrato)
- Questi protocolli implementano anche frame di controllo **RNR** (**Receiver Not Ready**) che **impongono** al trasmittente di **fermarsi** fino alla ricezione di un nuovo **RR**; questi possono essere utilizzati come ulteriore controllo di flusso, per gestire situazioni non di errore ma di **congestione** o temporanea sospensione della attivita' in ricezione

Protocollo go-back-N

- Questo protocollo segue la logica che in ricezione vengano **rifiutati** tutti i frame **successivi** ad un frame danneggiato o mancante



Go-Back-N

Mittente:

- Numero di sequenza a k bit nell'intestazione del pacchetto
- "Finestra" contenente fino a N pacchetti consecutivi non riscontrati



"riscontri cumulativi": un riscontro con num. di seq. n è interpretato come riscontro cumulativo che indica che tutti i pacchetti con un numero di sequenza $\leq n$ sono stati correttamente ricevuti dal receiver.

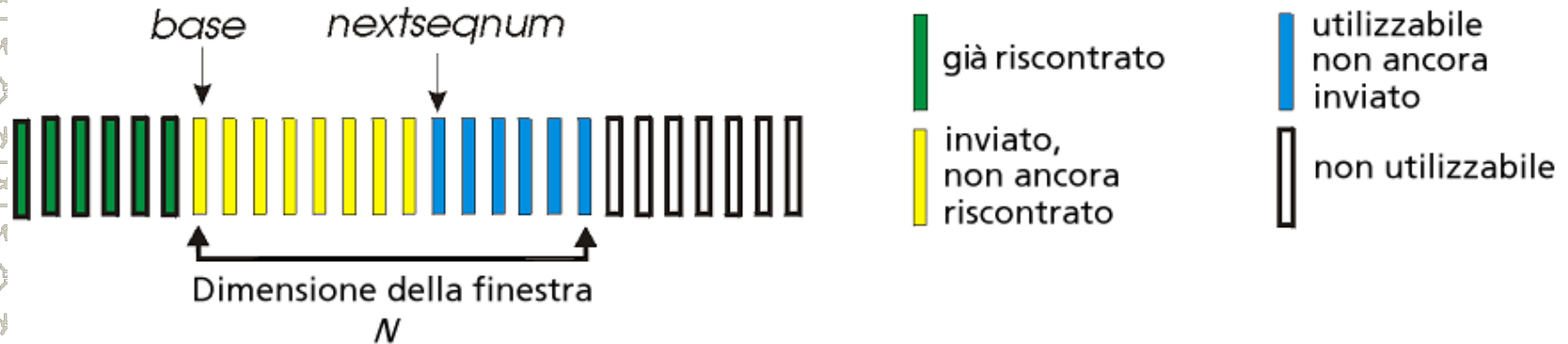
Richiede un timer per il primo pacchetto della finestra in transito

Se interviene un timeout, il sender rispedisce tutti i pacchetti già spediti ma senza riscontro.

Go-Back-N

Receiver:

- Numero di sequenza a k bit nell'intestazione del pacchetto
- "Finestra" contenente fino a N pacchetti consecutivi non riscontrati



Se un pacchetto con un num. di seq. n è ricevuto correttamente ed è in ordine, il receiver invia un ACK cumulativo per il pacchetto n ed invia i dati allo strato superiore.

In tutti gli altri casi, il receiver scarta il pacchetto e rispedisce un ACK relativo al pacchetto più di recente con l'ordine giusto.

Se arriva un pacchetto non in ordine, viene scartato. Infatti, si presume che il sender lo rimanda dopo il timeout e quindi sarebbe inutile conservarlo.

Go-Back-N



Vantaggio

Il receiver non ha bisogno di buffering, perché non ha bisogno di memorizzare alcun pacchetto fuori ordine.

L'unica cosa che il receiver deve conservare è il numero di sequenza del prossimo pacchetto in ordine.

Go-Back-N

Provare l'app al seguente link:

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html

Protocollo go-back-N

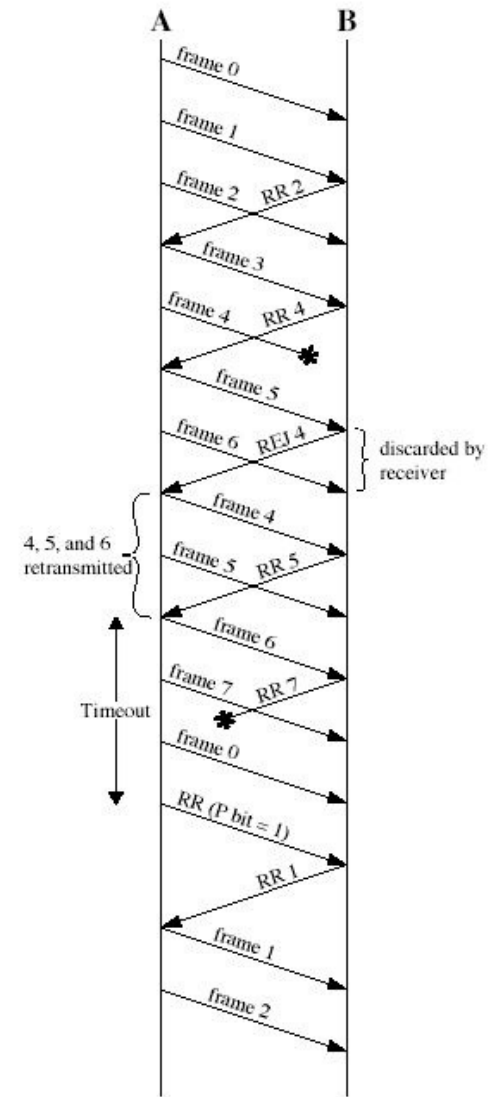
- Esistono due possibilità:
 - frame errato: in questo caso **B scarta il frame**:
 - se A **non invia** frame successivi, non accade nulla fino allo **scadere** del timer di A, quindi A **ricomincia** ad inviare frame a partire dal **primo** non riscontrato
 - se A **invia** frame successivi, B risponde con un **REJ** dei frame ricevuti, in modo da **notificare** ad A che il frame indicato nel REJ e' andato perso; al primo REJ ricevuto, A **ricomincia** dal primo frame non riscontrato

Protocollo go-back-N (cont.)

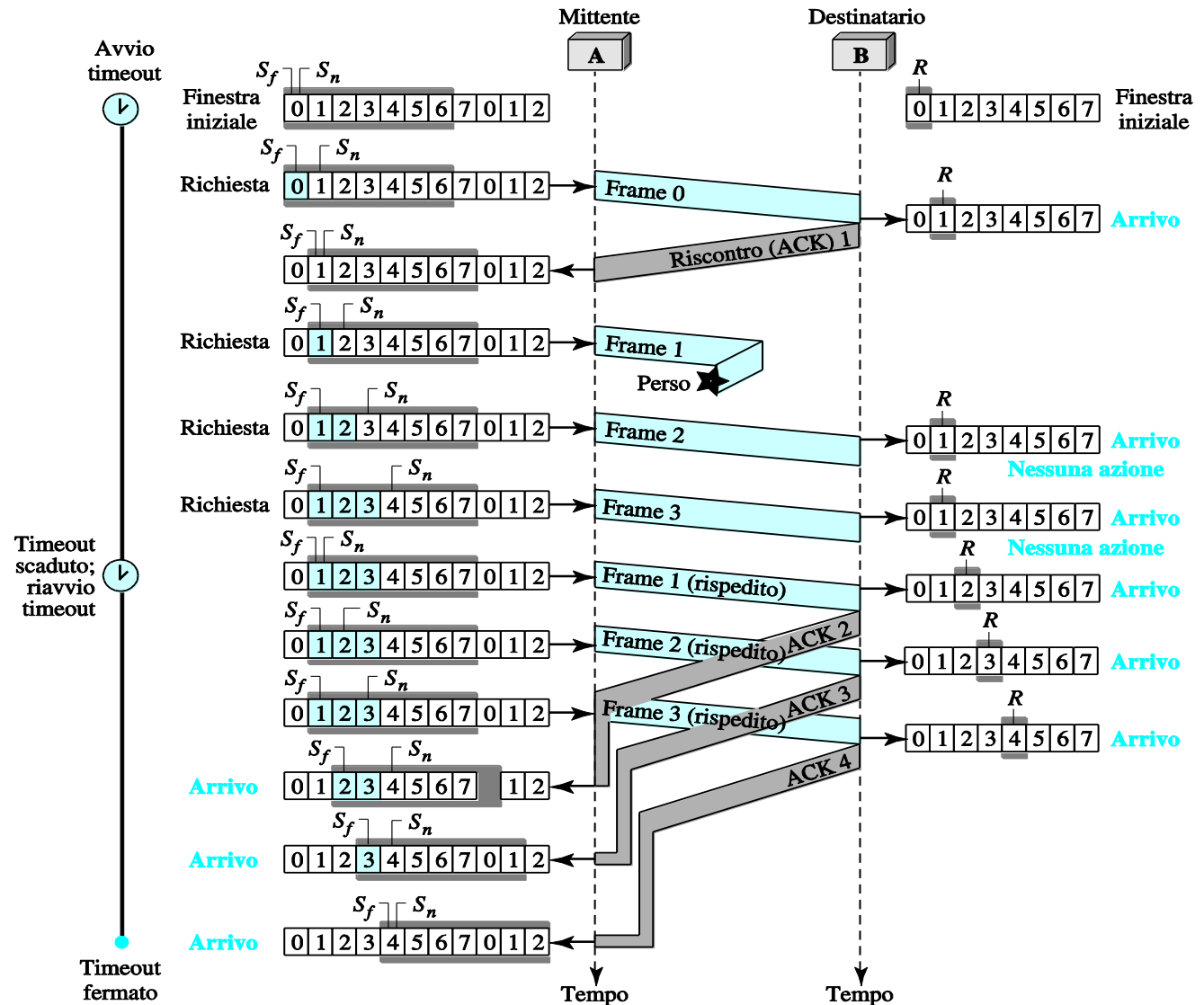
- ACK errato: in questo caso B ha accettato il frame:
 - se A non invia frame successivi, allo scadere del timer:
 - A invia nuovamente il frame; B lo rifiuta (duplicato) ma invia nuovamente l'ACK
 - alternatively, al timeout A puo' inviare un frame di controllo per chiedere conferma dell'ultimo frame ricevuto correttamente, a cui B risponde con l'ACK relativo
 - se A invia frame successivi, B risponde con l'ACK del frame successivo; poiche' gli ACK sono cumulativi, l'ACK del frame successivo riscontra anche quello di cui A non ha ricevuto l'ACK, quindi il trasferimento dati continua senza interruzioni

Esempio di go-back-N

- In questa immagine gli ACK sono indicati come RR (**Receiver Ready**)
- Alla ricezione del **frame 5** B identifica la perdita del 4, ed **invia un REJ** che indica il 4 come frame atteso; questo permette a B di **ripartire dal 4** prima del timeout
- la **perdita di RR7** comporta un timeout in quanto B **non ha riscontrato** i frame 7 e 0 in tempo; in questa situazione A **sollecita** un frame di RR, riceve il riscontro fino al frame 0 e ricomincia da 1



Esempio di go-back-N

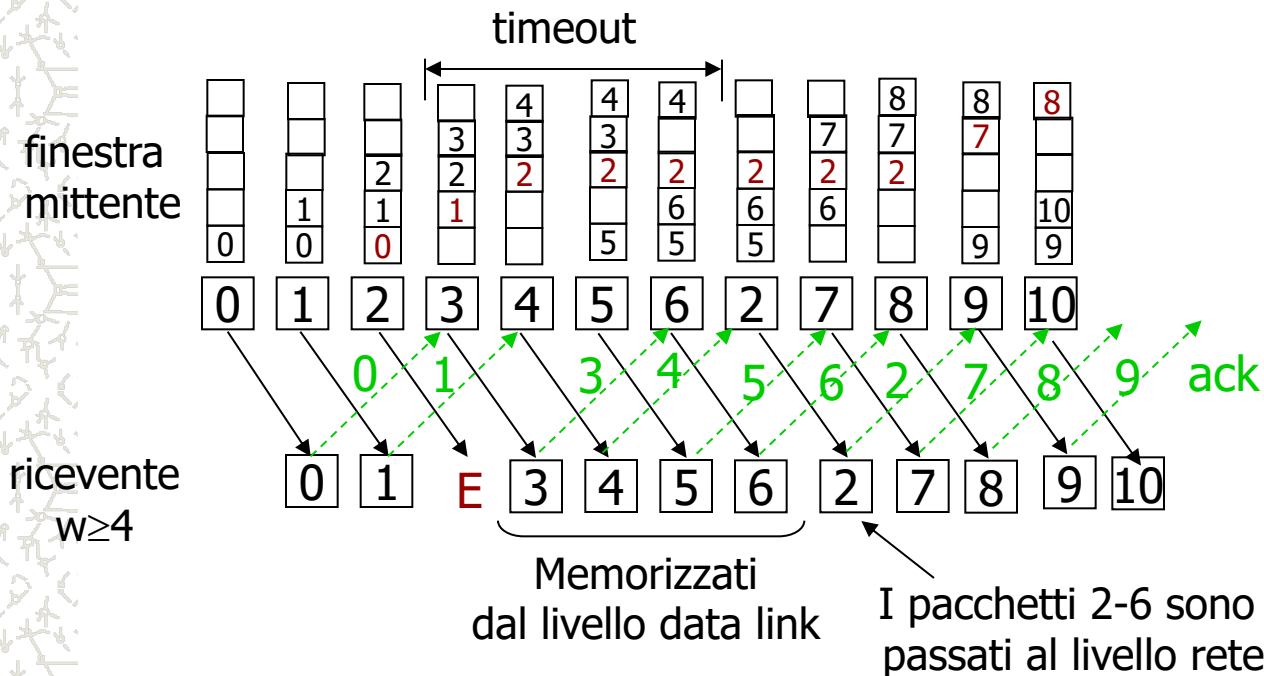


Dimensione della finestra per il go-back-N

- Poiche' i riscontri sono cumulativi, la dimensione della finestra deve essere $W \leq 2^n - 1$; infatti
 - supponiamo di avere $n=3$ (quindi numeri da 0 a 7) e scegliamo per W il valore 8
 - A invia il frame 7, e riceve ACK0 (riscontro del frame 7)
 - poi A invia i frame da 0 a 7, e riceve ACK0
 - A non puo' sapere se tutti i frame sono stati ricevuti (ACK0 e' il riscontro dell'ultimo frame inviato) o sono stati tutti perduti (ACK0 e' il riscontro ripetuto del primo frame inviato precedentemente)
 - Se nell'esempio la finestra e' $W = 7$, A puo' inviare frame da 0 a 6; a questo punto
 - se sono arrivati tutti, A riceverà ACK7
 - se sono andati tutti persi, A riceverà ACK0
- quindi con $W \leq 2^n - 1$ non c'e' ambiguita'

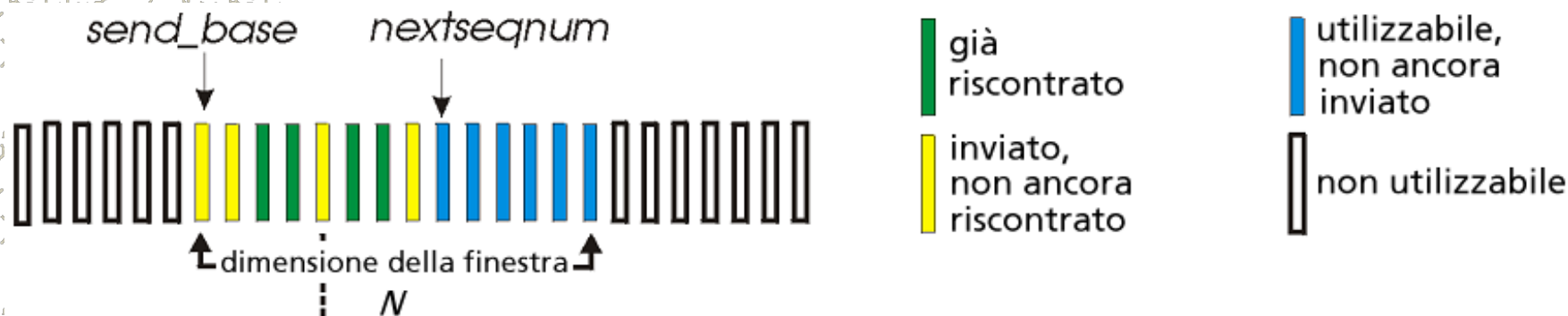
Protocollo selective reject

- Il protocollo **selective reject** prevede che in ricezione possano essere accettati frame **fuori sequenza**, utilizzando un meccanismo di ritrasmissione **selettiva** dei frame errati
- in questo modo si **riduce ulteriormente** il numero di frame ritrasmessi, mantenendo la caratteristica di recapitare allo strato di rete i dati **nell'ordine corretto**
- In ricezione i frame fuori ordine (ma **dentro** la finestra) vengono mantenuti nei **buffer** fino a che non siano stati ricevuti **tutti** i frame **intermedi**

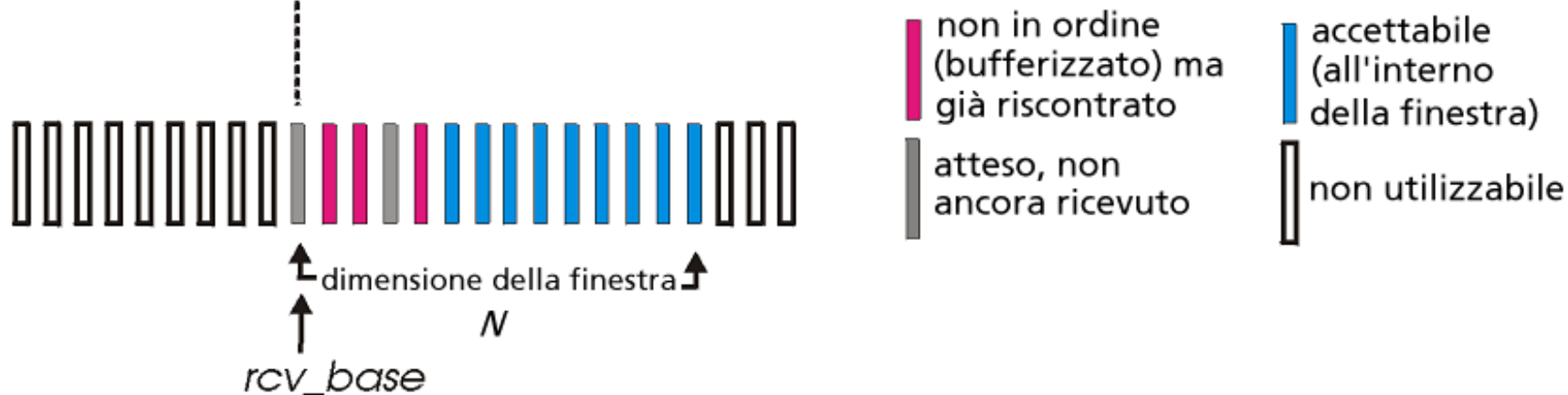


Ripetizione selettiva senza nack

Ripetizione selettiva: finestre del mittente e del ricevente



a) Visione del mittente sui numeri di sequenza



b) Visione del ricevente sui numeri di sequenza

Ripetizione Selettiva

Provare l'app al seguente link:

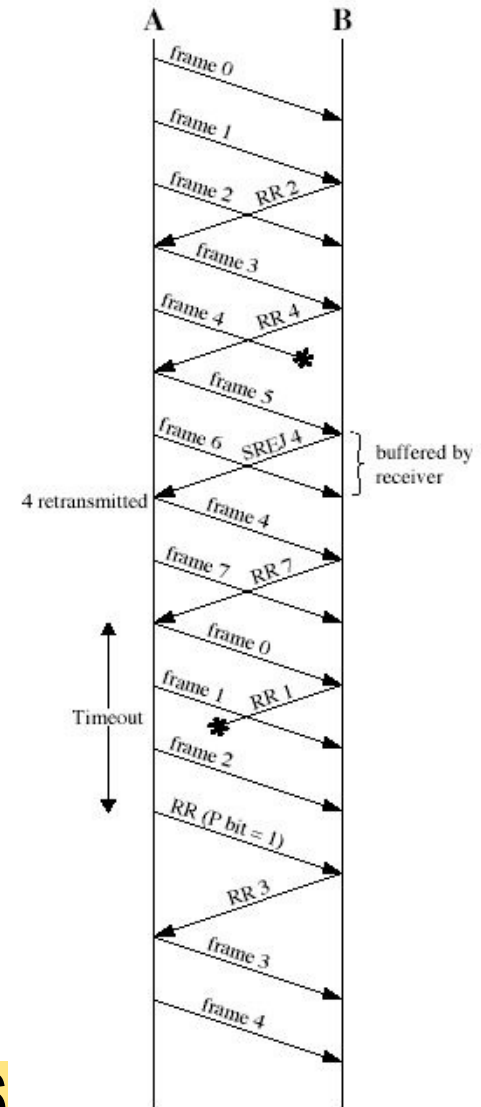
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwor_k_7/cw/content/interactiveanimations/selective-repeat-protocol/index.html

Protocollo selective reject (cont.)

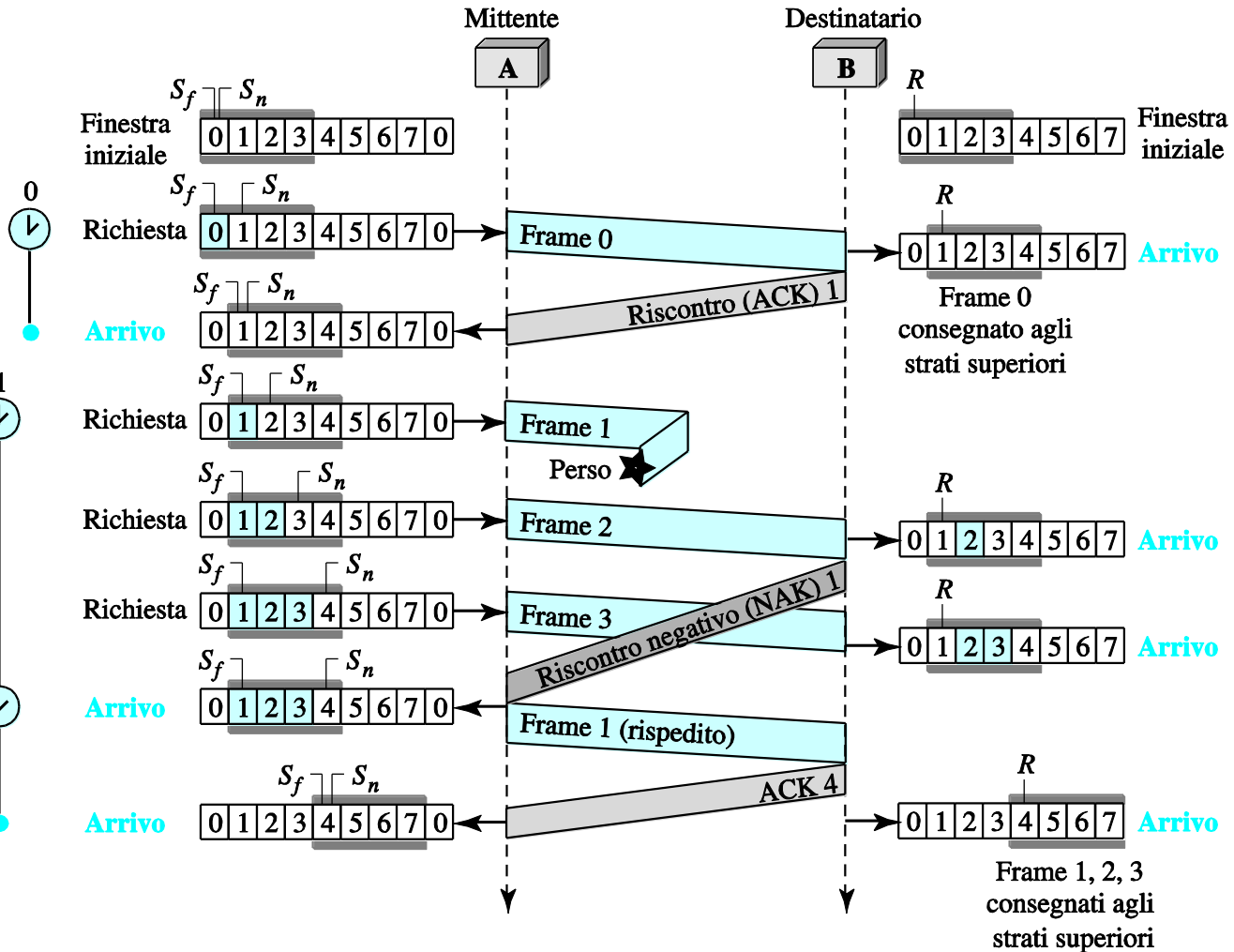
- Quando si ha un frame perduto, B riceverà il frame successivo **fuori sequenza**, al quale risponderà con un ACK **relativo al frame perduto**
- A non ritrasmette tutti i frame successivi a quello, ma **solo quello perduto**, quindi proseguirà con la normale sequenza
- B ha memorizzato i frame successivi, ed alla ricezione del frame **ritrasmesso** libererà tutti i buffer inviando un ACK relativo **all'ultimo frame** ricevuto **correttamente**
- In caso di **perdita dell'ACK**, sarà il timeout di A a generare un **frame di sollecito** di ACK per B, che risponderà di conseguenza

Esempio di selective reject

- Alla ricezione del **frame 5** B identifica la perdita del 4, ed invia un SREJ (o NACK) che indica il **4** come frame **atteso**; questo permette a A di **trasmettere il 4** dopo aver trasmesso il 6
- Nel frattempo B ha memorizzato il 5 ed il 6, ed alla ricezione del 4 invia l'RR **per il 6**
- la perdita di **RR1** comporta un timeout in quanto B non ha riscontrato i frame 1 e 2 in tempo; in questa situazione A sollecita un frame di RR, riceve il riscontro fino al frame 2 e **ricomincia da 3**



Esempio di selective reject



Dimensione della finestra per il selective reject

- La ricezione non sequenziale **limita ulteriormente** la massima dimensione della finestra in funzione del numero di bit per la numerazione del frame
- Come prima, supponiamo di avere 3 bit, ed una finestra a dimensione 7 (idonea per il protocollo go-back-N)
 - A **trasmette da 0 a 6**, B risponde con ACK7 e sposta la sua finestra in (7,0,1,2,3,4,5)
 - **l'ACK7 si perde**; dopo il timeout A **ritrasmette** il frame 0
 - B accetta 0 come un **nuovo frame** (ipotizza che il 7 sia andato perduto) e trasmette **NACK7**
 - A riceve NACK7, lo identifica come un **errore di protocollo** e chiede la ripetizione del riscontro, a cui B risponde con un **ACK7**
 - A ritiene a questo punto che i frame da **0 a 6** siano arrivati tutti e riparte con i nuovi: **7,0,1,...**
 - A riceve **7** (OK) ma lo 0 nuovo lo **interpreta** come **duplicato** di quello ricevuto precedentemente e lo butta; quindi si prosegue
- in questo esempio lo **strato di rete** riceve il **frame 0 vecchio** al posto del frame 0 nuovo
- Per eliminare l'ambiguità e' necessario che le **finestre** in trasmissione e ricezione **non si sovrappongano**; questo si ottiene imponendo che la finestra abbia dimensione $W \leq 2^{(n-1)}$, cioè la **meta** dello spazio di indirizzamento dei frame

Rilevazione dell'errore

- ❑ Il livello fisico offre un canale di trasmissione *non privo di errori*
 - errori sul singolo bit
 - replicazione di bit
 - perdita di bit
- ❑ Per la rilevazione di tali errori, nell'header di ogni trama il livello 2 inserisce un campo denominato **checksum**
 - il checksum è il risultato di un calcolo fatto utilizzando i bit della trama
 - la destinazione ripete il calcolo e confronta il risultato con il checksum: se coincide la trama è corretta



Controllo di parità

Il **bit di parità** è un codice di controllo utilizzato per prevenire errori nella trasmissione o nella memorizzazione dei dati.

Tale sistema prevede l'aggiunta di un bit ridondante ai dati, calcolato a seconda che il numero di bit che valgono 1 sia pari o dispari.

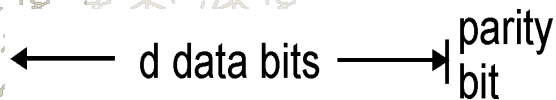
Ci sono due varianti del bit di parità:

- **bit di parità pari:** si pone tale bit uguale a 1 se il numero di "1" in un certo insieme di bit è dispari.
- **bit di parità dispari:** si pone tale bit uguale a 1 se il numero di "1" in un certo insieme di bit è pari

Controllo di parità

Esempio di bit di parità pari

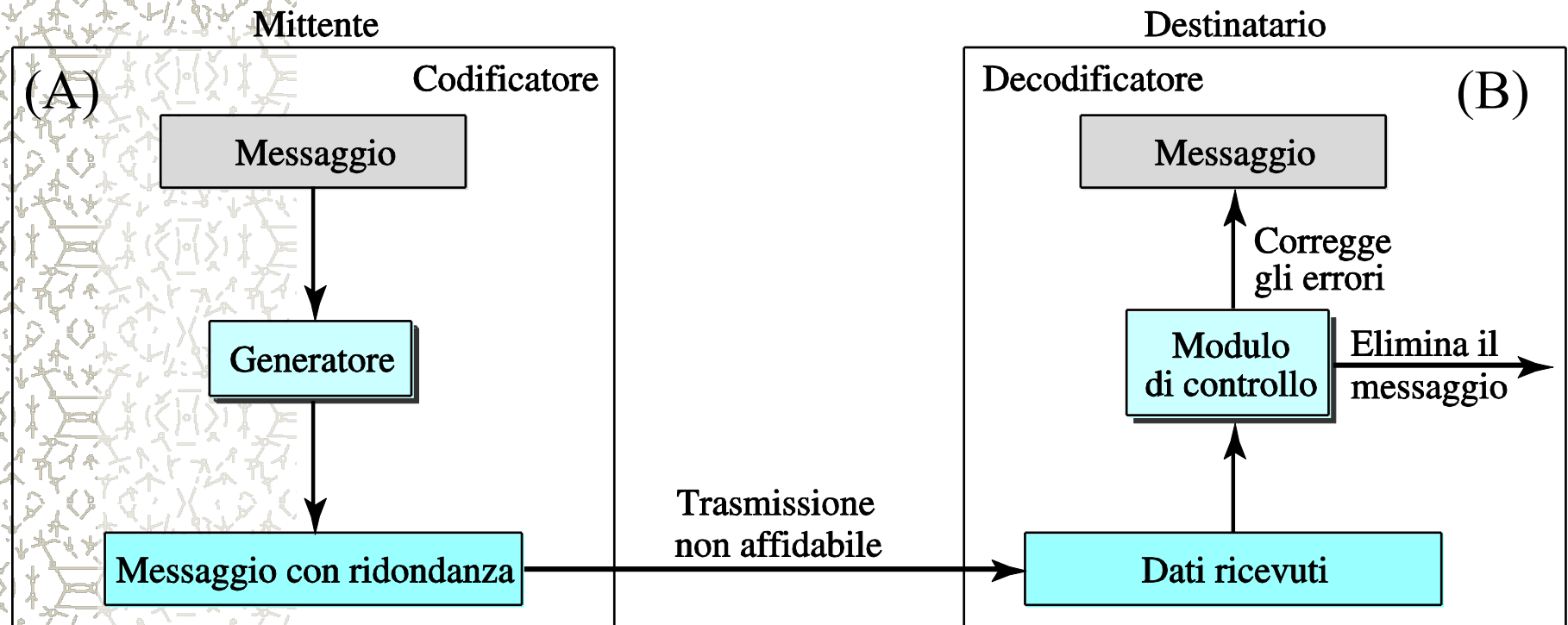
Parola sorgente	Parola codice	Parola sorgente	Parola codice
0000	0000 0	1000	1000 1
0001	0001 1	1001	1001 0
0010	0010 1	1010	1010 0
0011	0011 0	1011	1011 1
0100	0100 1	1100	1100 0
0101	0101 0	1101	1101 1
0110	0110 0	1110	1110 1
0111	0111 1	1111	1111 0



0111000110101011 0

Si è verificato almeno un errore in un bit

Controllo di parità



A vuole trasmettere: 1001

A calcola il bit di parità: $1^0 0^0 1 = 0$

A aggiunge il bit di parità e spedisce: 10010

B riceve: 10010

B calcola la parità totale: $1^0 0^0 1^0 = 0$

B può dire che la trasmissione è avvenuta correttamente.

Controllo di parità

A vuole trasmettere: 1001

A calcola il bit di parità: $1 \wedge 0 \wedge 0 \wedge 1 = 0$

A aggiunge il bit di parità e spedisce: 10010

*** ERRORE DI TRASMISSIONE ***

B riceve: 11010

B calcola la parità totale: $1 \wedge 1 \wedge 0 \wedge 1 \wedge 0 = 1$

B può dire che è avvenuto un errore durante la trasmissione.

- Il bit di parità garantisce di rilevare solo un numero dispari di errori.

A vuole trasmettere: 1001

A calcola il bit di parità: $1 \wedge 0 \wedge 0 \wedge 1 = 0$

A aggiunge il bit di parità e spedisce: 10010

*** ERRORE DI TRASMISSIONE ***

B riceve: 11011

B calcola la parità totale: $1 \wedge 1 \wedge 0 \wedge 1 \wedge 1 = 0$

B dice che la trasmissione è avvenuta correttamente anche se ci sono stati errori.

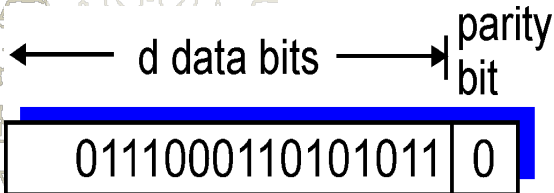
- Se avviene un numero pari di errori, non funziona.

Controllo di parità

Possibile soluzione:

Unico bit di parità:

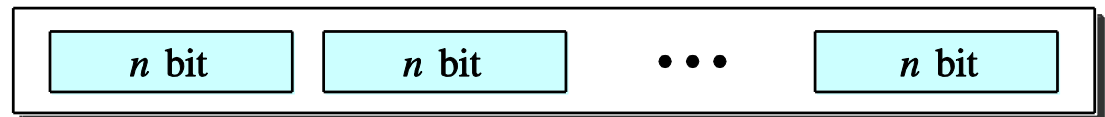
Si è verificato almeno un errore
in un bit



- Sequenza di bit sorgente
 - Divise in gruppi k bit, detti parole sorgente
- Codifica
 - Parole sorgente trasformate in parole codice di $n = k + r$, $r > 0$



2^k parole sorgenti, ognuna di k bit



2^n parole codice, ognuna di n bit (solo 2^k sono valide)

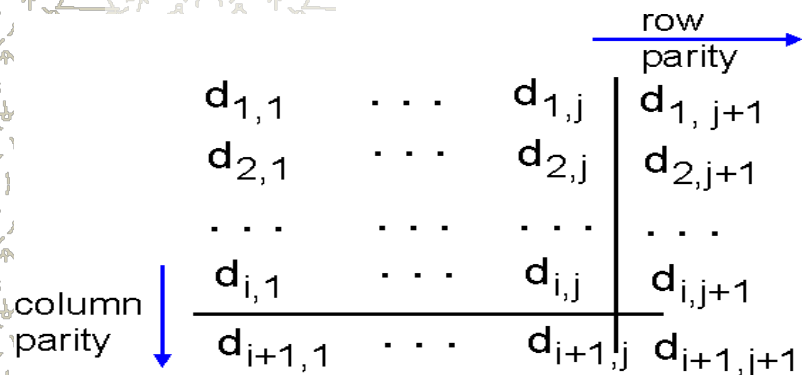
- Il bit di parità garantisce di rilevare solo un numero dispari di errori.

- Non è in grado di individuare il bit errato

Controllo di parità

Parità bidimensionale:

Individua e corregge il bit alterato



- Le parole di k bit sono disposte una sotto l'altra.
- Viene aggiunto il bit di parità sia considerando le righe che le colonne
- In definitiva si avranno $2K+1$ bit di parità

- È possibile individuare il bit errato

- Ma non sempre

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

no errors

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
0	0	1	0	1	0

parity
error

*correctable
single bit error*

Controllo di parità

Un errore influenza due bit di parità

1	1	0	0	1	1	1	1
1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1



Errore individuato:

Due errori influenzano due bit di parità

1	1	0	0	1	1	1	1
1	0	0	1	0	0	1	1
0	1	1	1	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1



Errori non individuati
ma rilevati

Tre errori influenzano quattro bit di parità

1	0	0	0	1	1	1	1
1	0	1	0	1	0	1	1
0	1	1	0	0	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1



Errori non individuati
ma rilevati

Quattro errori ...

1	1	0	1	0	1	1	1
1	0	1	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1

... non vengono rilevati

I campi di Galois

- Un campo finito con q elementi su cui sono definite due operazioni aritmetiche (addizione e moltiplicazione) che godono della proprietà commutativa ed associativa viene chiamato **Campo di Galois** ed indicato con **GF(q)**.
- **GF(q)** è chiuso rispetto all'addizione e moltiplicazione
- In generale q deve essere sempre primo o potenza di numeri primi
- Le operazioni di somma e moltiplicazione vengono calcolate utilizzando i concetti aritmetici tradizionali con l'applicazione di un'ulteriore operazione di **mod q** .

GF(5)

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

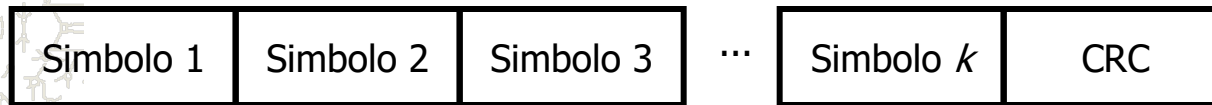
GF(2)

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

Semplici codici di controllo: CRC

Cyclic Redundance Check-sum su GF(5).



$$\text{CRC} = \sum_i m_i \quad \text{su GF}(q)$$

Data $m = 1023110223242234$ CRC = 2

Basato sul concetto di codice ciclico, in cui permutando ciclicamente gli elementi di una qualsiasi combinazione, si ottengono sempre combinazioni dello stesso codice.

Rappresentazione di sequenze di bit tramite polinomi

- Una sequenza di **N bit** puo' essere rappresentata tramite un **polinomio** a coefficienti **binari**, di grado pari a **N-1**, tale che i suoi coefficienti siano uguali ai valori dei bit della sequenza
- Il bit piu' a sinistra rappresenta il coefficiente del termine di grado N-1, mentre il bit piu' a destra rappresenta il termine noto (di grado 0)
- Ad esempio, la sequenza **1001011011** puo' essere rappresentata dal polinomio

$$x^9 + x^6 + x^4 + x^3 + x + 1$$

- Il **grado** del polinomio e' determinato dal **primo** bit a sinistra di **valore 1** presente nella sequenza

Codifica polinomiale (CRC)

- La tecnica consiste nel considerare i dati (m bit) da inviare come un **polinomio di grado $m-1$**
- Trasmettitore e ricevitore si **accordano** sull'utilizzo di un **polinomio generatore $G(x)$** di grado r
- Il trasmettitore **aggiunge** in coda al messaggio una sequenza di bit di controllo (**CRC**) in modo che il **polinomio associato** ai bit del frame **trasmesso**, costituito dall'insieme di dati e CRC, sia **divisibile** per $G(x)$
- In ricezione **si divide** il polinomio associato ai dati ricevuti per $G(X)$
 - se la divisione ha resto **nullo**, si assume che la trasmissione sia avvenuta **senza errori**
 - se la divisione ha resto **non nullo**, sono certamente avvenuti **errori**

Codici ciclici

Assegniamo un polinomio P di grado $p-1$ al messaggio che vogliamo trasmettere.

$$m = 10100011 \Rightarrow P(x) = 1 \cdot x^7 + 0 \cdot x^6 + 1 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x^1 + 1 \cdot x^0$$

$$\text{cioè } P(x) = x^7 + x^5 + x + 1$$

Scelto $G(x)$ di grado $r \leq p - 1$, detto **polinomio generatore** (noto a mittente e destinatario), per esempio, con $r = 3$ scegliamo $G(x) = x^3 + 1$

Si aggiungono r zeri ai p bit del blocco da trasmettere, che diviene quindi $x^r P(x)$:

Nel nostro esempio m diventa 10100011**000** cioè $x^3 P(x) = x^{10} + x^8 + x^4 + x^3$

Effettuando la divisione per $G(x)$:
$$\frac{x^r P(x)}{G(x)} \rightarrow Q(x)G(x) + R(x) = x^r P(x)$$

cioè $x^r P(x) - R(x) = Q(x)G(x)$ Poiché operiamo nel caso dei codici binari, il campo di Galois utilizzato è GF(2). Quindi $-R(x) = +R(x)$

La formula precedente diventa: $x^r P(x) + R(x) = Q(x)G(x)$ che è una quantità divisibile per $G(x)$ con resto 0 (multiplo intero)

Codici ciclici [continua]

Quindi quello che trasmettiamo è esattamente la parola di codice corrispondente al polinomio ottenuto aggiungendo r zeri al messaggio sorgente cui viene aggiunto il resto della sua divisione per il polinomio generatore $G(x)$, cioè: $x^r P(x) + R(x)$

Nel nostro esempio: $x^3 P(x) = x^{10} + x^8 + x^4 + x^3$ e $G(x) = x^3 + 1$

$$Q(x) = 10110101 \rightarrow x^7 + x^5 + x^4 + x^2 + 1$$

$$R(x) = 101 \rightarrow x^2 + 1$$

$$x^r P(x) + R(x) \quad 10100011\mathbf{000} + \mathbf{101}$$

Che nel nostro caso diventa: 10100011101 $T(x) = x^{10} + x^8 + x^4 + x^3 + x^2 + 1$

Come conseguenza delle definizioni precedenti $T(x)$ definisce una parola di un codice ciclico che è sempre multiplo del polinomio generatore $G(x)$.

Quindi per verificare la corretta trasmissione basta dividere $T(x)$ per $G(x)$. Se **il resto della divisione è zero**, allora **non si è verificato nessun errore**.

CRC standard

- Sono stati definiti dei polinomi di fatto usati come standard

$$G(x) = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1 \quad \text{CRC-12}$$

$$G(x) = x^{16} + x^{15} + x^2 + 1 \quad \text{CRC-16}$$

$$G(x) = x^{16} + x^{12} + x^5 + 1 \quad \text{CRC-CCITT}$$

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad \text{CRC-32}$$

- Tutti contengono $x+1$ come fattore
- CRC-16 e CRC-CCITT riconoscono errori singoli e doppi, errori con un numero dispari di bit, i burst di errori di lunghezza massima 16, il 99.997% dei burst di lunghezza 17 bit
- Il circuito per il calcolo del checksum può essere realizzato semplicemente in hardware