Appunti del corso di
Reti di elaboratori

PROF. G. BONGIOVANNI

3) IL LIVELLO DUE (DATA LINK) 2

3.1) Framing 4
3.1.1) Conteggio 4
3.1.2) Caratteri di inizio e fine 5
3.1.3) Bit pattern di inizio e fine 5
3.1.4) Violazioni della codifica 6

3.2) Rilevamento e correzione errori 6
3.2.1) Codici per la correzione degli errori 7
3.2.2) Codici per il rilevamento degli errori 9

3.3) Gestione sequenza di trasmissione e flusso 11
3.3.1) Protocollo 1: Heaven 13
3.3.2) Protocollo 2: Simplex Stop and Wait 14
3.3.3) Protocollo 3: simplex per canale rumoroso 15
3.3.4) Protocolli a finestra scorrevole 21

3.4) Esempi di protocolli data link 28
3.4.1) HDLC (High Level Data Link Control) 28
3.4.2) SLIP (Serial Line IP) 29
3.4.3) PPP (Point to Point Protocol) 29

3) Il livello due (Data Link)

Questo livello ha il compito di offrire una comunicazione affidabile ed efficiente a due macchine ***adiacenti***, cioé connesse fisicamente da un canale di comunicazione (ad es.: cavo coassiale, doppino, linea telefonica).

Esso si comporta come un "tubo digitale", cioé i bit partono e arrivano nello stesso ordine. La cosa non è banale come sembra, perché:
- ci sono errori e disturbi occasionali;
- il canale ha un data rate finito;
- c'è un ritardo nella propagazione.

I protocolli usati per la comunicazione devono tenere in conto tutto questo.

Competenze del livello data link
Questo livello ha le seguenti incombenze principali:
- offrire servizi al livello network con un'interfaccia ben definita;
- determinare come i bit del livello fisico sono raggruppati in ***frame*** (framing);
- gestire gli errori di trasmissione;
- regolare il flusso della trasmissione fra sorgente e destinatario.

Servizi offerti al livello Network
Il servizio principale è trasferire dati dal livello network della macchina di origine al livello network della macchina di destinazione.

Come sappiamo, la trasmissione reale passa attraverso il livello fisico, ma noi ragioniamo in termini di dialogo fra due processi a livello data link che usano un protocollo di livello data link.

I servizi offerti possono essere diversi. I più comuni sono:
- ***connectionless non confermato***
    - si mandano frame indipendenti;
    - i frame non vengono confermati;
    - non si stabilisce una connessione;
    - i frame persi non si recuperano (in questo livello);
    - è appropriato per:
        - canali con tasso d'errore molto basso;
        - traffico real-time (es. voce);
        - le LAN, nelle quali, in effetti, è molto comune.

- ***connectionless confermato***
    - come sopra, però i frame vengono confermati;
    - se la conferma non arriva, il mittente può rispedire il frame;
    - è utile su canali non affidabili (ad es. in sistemi wireless);
    - nota 1: la perdita di un ack può causare la trasmissione di più copie dello stesso frame;
    - nota 2: avere la conferma a questo livello è un'ottimizzazione, mai una necessità. Infatti la conferma può sempre essere fatta a livello superiore, ma con linee disturbate ciò può essere gravoso.
- ***connection oriented confermato***
    - è il servizio più sofisticato;
    - prevede tre fasi (apertura conn./invio dati/chiusura conn.);
    - garantisce che ogni frame sia ricevuto esattamente una volta e nell'ordine giusto;
    - fornisce al livello network un flusso di bit affidabile.

Vediamo ora un tipico esempio di funzionamento. Consideriamo un router con alcune linee in ingresso ed alcune in uscita. Ricordiamo che il routing avviene a livello tre (network), quindi il router gestisce i livelli uno, due e tre.

1. Quando al router arrivano dei bit da una linea fisica, l'hardware apposito se ne accorge (siamo a livello 1) e li passa al corrispondente SW/HW di livello due.
2. Il SW/HW di livello due (data link), che in genere è contenuto in un chip sulla ***scheda (o adattatore) di rete*** (tipico esempio è una scheda Ethernet) fa i controlli opportuni:
    - framing;
    - controllo errori di trasmissione;
    - controllo numero di frame (se necessario).
3. Se tutto è OK, il SW/HW di livello due genera un interrupt alla cpu, che chiama in causa il SW di livello tre (network):
    - questo è tipicamente un processo di sistema, al quale viene passato il pacchetto contenuto nel frame di livello due per l'ulteriore elaborazione;
    - l'elaborazione consiste nel determinare, sulla base delle caratteristiche del pacchetto (in particolare dell'indirizzo di destinazione), su quale linea in uscita instradarlo.
4. Presa questa decisione, il SW di livello tre consegna il pacchetto al corrispondente SW/HW di livello due, che lo imbusta in un nuovo frame e lo consegna al sottostante livello fisico (ossia quello relativo alla linea in uscita prescelta).

Il livello uno accetta un flusso di bit grezzi e cerca di farli arrivare a destinazione. Però:
- il flusso non è esente da errori;
- possono arrivare più o meno bit di quanti sono stati inviati.

E' compito del livello due rilevare, e se possibile correggere, tali errori. L'approccio usuale del livello due è il seguente.

- In trasmissione:
  - spezza il flusso di bit che arriva dal livello tre in una serie di frame;
  - calcola un'apposita funzione (***checksum***) per ciascun frame;
  - inserisce il checksum nel frame;
  - consegna il frame al livello uno, il quale lo spedisce come sequenza di bit.
- In ricezione:
  - riceve una sequenza di bit dal livello uno;
  - ricostruisce da essa un frame dopo l'altro;
  - per ciascun frame ricalcola il checksum;
  - se esso è uguale a quello contenuto nel frame questo viene accettato, altrimenti viene considerato errato e quindi scartato.

Dunque, la prima cosa (più difficile di quanto sembri) di cui occuparsi, è come delimitare un singolo frame.

3.1) Framing

E' rischioso usare lo spazio temporale che intercorre tra i frame per delimitarli, perché nelle reti in genere non ci sono garanzie di temporizzazione. Quindi, si devono usare degli appositi marcatori per designare l'inizio e la fine dei frame.

Ci sono vari metodi:

- ***conteggio dei caratteri***;
- ***caratteri di inizio e fine***, con ***character stuffing***;
- ***bit pattern di inizio e fine***, con ***bit stuffing***;
- ***violazioni della codifica*** dei bit usata nel livello fisico.

3.1.1) Conteggio

Si utilizza un campo nell'header, per indicare quanti caratteri ci sono del frame

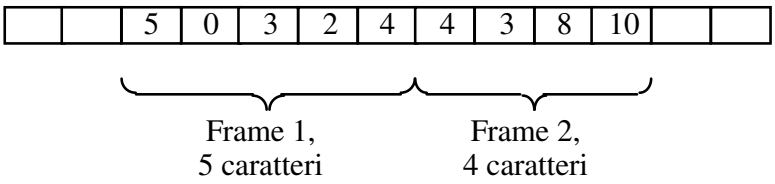


Figura 3-1: Il metodo del conteggio

Se durante la trasmissione si rovina il campo del frame che contiene il conteggio, diventa praticamente impossibile ritrovare l'inizio del prossimo frame e di conseguenza anche quello dei successivi. A causa della sua scarsa affidabilità questo metodo è usato ormai pochissimo.

3.1.2) Caratteri di inizio e fine

Ogni frame inizia e finisce con una particolare la sequenza di caratteri ASCII.

Una scelta diffusa è la seguente:

- inizio frame:
  - ***DLE*** (Data Link Escape), ***STX*** (Start of TeXt)
- fine frame:
  - ***DLE, ETX*** (End of TeXt)

In questo modo, se la destinazione perde traccia dei confini di un frame la riacquista all'arrivo della prossima coppia DLE, STX e DLE, ETX.

Esiste però un problema: nella trasmissione di dati binari, il byte corrispondente alla codifica di DLE può apparire dentro il frame, imbrogliando le cose.

Per evitare questa evenienza, il livello due sorgente aggiunge davanti a tale byte un altro DLE, per cui in arrivo solo i singoli DLE segnano i confini dei frame. Naturalmente, il livello due di destinazione rimuove i DLE aggiunti dentro ai dati prima di consegnarli al livello tre. Questa tecnica si chiama ***character stuffing***.

3.1.3) Bit pattern di inizio e fine

La tecnica precedente è adatta alle situazioni nelle quali la dimensione del frame è misurata in byte, ossia il numero totale di bit di ogni frame è sempre un multiplo di 8. Una tecnica analoga permette di delimitare frame costituiti da un numero arbitrario di bit.

Ogni frame inizia e finisce con una specifica sequenza di bit (bit pattern), ad es.:

```
01111110
```

chiamata un ***flag byte***.

Ovviamente esiste un problema analogo al caso precedente: tale sequenza di bit può infatti apparire all'interno dei dati che devono essere trasmessi.

In questo caso:
- in trasmissione: ogni volta che il livello due incontra nei dati da trasmettere 5 bit consecutivi uguali a 1 inserisce uno zero aggiuntivo;
- in ricezione: quando nei dati ricevuti compaiono 5 bit uguali a uno, si rimuove lo zero che li segue.

Dunque, il flag byte può apparire solo all'inizio ed alla fine dei frame. Questa tecnica va sotto il nome di ***bit stuffing***.

3.1.4) Violazioni della codifica

In molte reti (sopratutto LAN) si codificano i bit al livello fisico con una certa ridondanza. Ad esempio:
- il valore 1 di un bit di dati è codificato con la coppia ***high/low*** di bit fisici;
- il valore 0 di un bit di dati è codificato con la coppia ***low/high*** di bit fisici.

Le coppie low/low ed high/high non sono utilizzate. Codifiche come questa (***Manchester encoding***, usata in IEEE 802.3) hanno lo scopo di consentire una facile determinazione dei confini di un bit dati, poiché esso ha sempre una trasmissione nel mezzo (però ci vuole una bandwidth doppia per trasmetterlo a parità di velocità).

Le coppie high/high e low/low possono quindi essere usate per delimitare i frame.

3.2) Rilevamento e correzione errori

Ci sono molti fattori che possono provocare errori, sopratutto sul local loop e nelle trasmissioni wireless. Viceversa, essi sono piuttosto rari nei mezzi più moderni quali le fibre ottiche.

Gli errori sono dovuti in generale a:
- rumore di fondo;
- disturbi (ad es. fulmini) improvvisi;
- interferenze (ad es. motori elettrici).

Ci sono due approcci al trattamento degli errori:
- includere abbastanza informazione aggiuntiva in modo da poter ricostruire il messaggio originario (***correzione dell'errore***);
- includere meno informazione aggiuntiva, in modo da accorgersi che c'è stato un errore, senza per questo essere in grado di correggerlo (***rilevazione dell'errore***).

3.2.1) Codici per la correzione degli errori

Normalmente, un frame (a parte i delimitatori) consiste di

$$n = m + r$$

bit, dove:
- m bit costituiscono il messaggio vero e proprio;
- r bit sono ridondanti, e sono detti ***reduntant bit*** (o ***check bit***).

Una sequenza di n bit fatta in tal modo si dice ***codeword***, o ***parola di codice***.

Date due qualunque parole di codice, ad es.:

```
1000 1001
1011 0001
```

è possibile determinare il numero di bit che in esse differiscono (tre nell'esempio) tramite un semplice XOR fatto bit a bit.

Tale numero si dice la ***distanza di Hamming*** delle due codeword (Hamming, 1956). Se due codeword hanno una distanza di Hamming uguale a d, ci vogliono esattamente d errori su singoli bit per trasformare l'una nell'altra.

Un insieme prefissato di codeword costituisce un ***codice*** (***code***). La ***distanza di Hamming di un codice*** è il minimo delle distanze di Hamming fra tutte le possibili coppie di codeword del codice.

Ora, le capacità di rilevare o correggere gli errori dipendono strettamente dalla distanza di Hamming del codice scelto.

In particolare:
- per rilevare d errori serve un codice con distanza di Hamming (d+1). Infatti, in questo caso qualunque combinazione di d errori non riesce a trasformare un codeword valido in un altro codeword valido, per cui si ha per forza un codeword non valido, che quindi rivela il fatto che ci sono stati degli errori;
- per correggere d errori, serve un codice di Hamming con distanza (2d+1). Infatti in questo caso un codeword contenente fino a d errori è più vicino a quello originario che a qualunque altro codeword valido.

Dunque, a seconda degli scopi che si vogliono raggiungere, si progetta un algoritmo per il calcolo degli r check bit (in funzione degli m bit del messaggio) in modo che i codeword di n = m + r bit risultanti costituiscano un codice con la desiderata distanza di Hamming.

Vediamo ora, come esempio, un codice ottenuto mediante l'aggiunta di un bit di parità, calcolato in modo tale che il numero totale di bit uguali ad 1 sia pari (in questo caso si parla di ***even parity***)

```
<-- m -->  r

1011 0101  1
1000 0111  0
```

Questo codice, detto ***codice di parità*** (***parity code***) ha distanza di Hamming uguale a due, e dunque è in grado di rilevare singoli errori. Infatti, un singolo errore produce un numero dispari di 1 e quindi un codeword non valido.

Come secondo esempio, consideriamo il codice costituito soloemante dalle seguenti quattro codeword:

```
00000 00000
00000 11111
11111 00000
11111 11111
```

Esso ha distanza 5, per cui è in grado di correggere due errori.
Infatti, se viene inviata la sequenza:

```
00000 11111
```

ed arriva (le cifre sottolineate indicano i bit che hanno subito gli errori) la sequenza:

```
00000 00111
```

si può risalire correttamente al codeword più vicino, che è ovviamente, fra i quattro che costituiscono il codice, la sequenza:

```
00000 11111
```

Però, se si verificassero invece tre errori ed arrivasse ad esempio la sequenza:

```
00000 00011
```

saremmo nei guai, perché essa verrebbe interpretata erroneamente come il codeword

```
00000 00000
```

da cui la sequenza ricevuta dista 2, anziché come il codeword

```
00000 11111
```

da cui la sequenza ricevuta dista 3.

Per correggere un singolo errore su m bit, si devono impiegare almeno r check bit, con

$$2^r >= m + r + 1$$

ossia sono necessari circa lg_2 m bit. Esiste un codice (***codice di Hamming***) che raggiunge questo limite teorico.

Ora, c'è una semplice tecnica con la quale tale codice può essere impiegato per correggere gruppi contigui di errori (detti ***burst di errori***) di lunghezza massima prefissata.

Supponiamo di voler correggere burst di lunghezza k:
- si accumulano k codeword, riga per riga;
- i codeword si trasmettono per colonne;
- quando arrivano si riassemblano per righe. Un burst di k errori comporta un singolo errore in ciascun codeword, che quindi può essere completamente ricostruito.

3.2.2) Codici per il rilevamento degli errori

I codici correttori di errore sono usati raramente (ad esempio in presenza di trasmissioni simplex, nelle quali non è possibile inviare al mittente una richiesta di ritrasmissione), perché in generale è più efficiente limitarsi a rilevare gli errori e ritrasmettere saltuariamente i dati piuttosto che impiegare un codice (più dispendioso in termini di ridondanza) per la correzione degli errori.

Infatti, ad esempio, supponiamo di avere:
- canale con errori isolati e probabilità di errore uguale a 10^{-6} per bit;
- blocchi dati di 1.000 bit.

Per correggere errori singoli su un blocco di 1.000 bit, ci vogliono 10 bit, per cui un Megabit richiede 10.000 check bit.

Viceversa, per rilevare l'errore in un blocco, basta un bit (con parity code). Ora, con 10^{-6} di tasso d'errore, solo un blocco su 1.000 è sbagliato e quindi deve essere ritrasmesso. Di conseguenza, per ogni Megabit si devono rispedire 1.001 bit (un blocco più il parity bit).

Dunque, l'overhead totale su un Megabit è:

- 1.000 bit per parity bit su 1.000 blocchi,
- 1.001 bit per il blocco ritrasmesso,
- per un totale di 2.000 bit contro i 10.000 del caso precedente.

L'uso del parity bit può servire (con un meccanismo analogo a quello visto per la correzione di burst di k errori) per rilevare burst di errori di lunghezza <= k. La differenza è che non si usano r check bit per ogni codeword, ma uno solo.

Esiste però un altro metodo che nella pratica viene usato quasi sempre, il ***Cyclic Redundancy Code*** (***CRC***), noto anche come ***polynomial code***. I polynomial code sono basati sull'idea di considerare le stringhe di bit come rappresentazioni di polinomi a coefficenti 0 e 1 (un numero ad m bit corrisponde ad un polinomio di grado m-1).
Ad sempio, la stringa di bit 1101 corrisponde al polinomio $x^3 + x^2 + x^0$.

L'aritmetica polinomiale è fatta modulo 2, secondo le regole della teoria algebrica dei campi. In particolare:
- addizione e sottrazione sono equivalenti all'or esclusivo (non c'è riporto o prestito);
- la divisione è come in binario, calcolata attraverso la sottrazione modulo 2.

Il mittente ed il destinatario si mettono d'accordo su un polinomio generatore G(x), che deve avere il bit più significativo e quello meno significativo entrambi uguali ad 1. Supponiamo che G(x) sia di grado r, e quindi abbia (r + 1) bit.

Il frame M(x), del quale si vuole calcolare il checksum, dev'essere più lungo di G(x). Supponiamo che abbia m bit, con m > r +1.

L'idea è di appendere in coda al frame un checksum costituito di r bit tale che il polinomio corrispondente (che ha grado m - 1 + r) sia divisibile per G(x).

Quando il ricevitore riceve il frame più il checksum, divide il tutto per G(x). Se il resto della divisione è zero è tutto OK, altrimenti c'è stato un errore.

Il calcolo del checksum si effettua come segue:
1. Appendere r bit a destra del frame, che quindi ha m + r bit, e corrisponde ad $x^rM(x)$;
2. Dividere $x^rM(x)$ per G(x);
3. Sottrarre ad $x^rM(x)$ il resto della divisione effettuata al passo precedente. Ciò che si ottiene è il frame più il checksum da trasmettere, che è ovviamente divisibile per G(x). Si noti che di fatto questa è un'operazione di XOR fatta sugli r bit meno significativi, dato che il resto della divisione ha al più r bit, e quindi non modifica il frame.

Questo metodo è molto potente, infatti un codice polinomiale con r bit:
- rileva tutti gli errori singoli e doppi;

- rileva tutti gli errori di x bit, x dispari;
- rileva tutti i burst di errori di lunghezza <= r.

Tra i polinomi sono diventati standard internazionali:

- ***CRC-12***: $x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$;
- ***CRC-16***: $x^{16} + x^{15} + x^2 + 1$;
- ***CRC-CCITT***: $x^{16} + x^{12} + x^5 + 1$;

Un checksum a 16 bit rileva:
- errori singoli e doppi;
- errori di numero dispari di bit;
- errori burst di lunghezza <= 16;
- 99.997% di burst lunghi 17;
- 99.998% di burst lunghi 18.

Questi risultati valgono sotto l'ipotesi che gli m bit del messagio siano distribuiti casualmente, il che però non è vero nella realtà, per cui i burst di 17 e 18 possono sfuggire più spesso di quanto si creda.

3.3) Gestione sequenza di trasmissione e flusso

Dopo aver trovato il modo di delineare l'inizio e la fine dei frame e di gestire gli errori di trasmissione, bisogna trovare la maniera di informare opportunamente il mittente se i frame spediti sono anche arrivati, e senza errori, dall'altra parte:
- servizi connectionless non confermati: non c'è bisogno di alcuna conferma;
- servizi connectionless confermati: sono arrivati tutti e senza errori?
- servizi connection oriented confermati: sono arrivati tutti, senza errori e nell'ordine giusto?

Per saperlo si introduce il concetto di ***acknowledgement***, che è un messaggio inviato dal destinatario al mittente per informarlo che:
- il frame è arrivato correttamente (***positive ack***);
- il frame è errato (***negative ack***).

Nel seguito, ove necessario per evitare ambiguità, col termine ***frame dati*** indicheremo un frame che trasporta informazioni generate nel colloquio fra le peer entity; col termine ***frame di ack*** indicheremo un frame il cui solo scopo è trasportare un acknowledgement.

Introducendo l'uso degli ack sorgono però alcuni problemi, fra cui:

Problema 1	un frame può anche sparire del tutto, per cui il mittente rimane bloccato in attesa di un ack che non arriverà mai;
Soluzione 1	il mittente stabilisce un time-out per la ricezione dell'ack. Se questo non arriva in tempo, il frame si ritrasmette.

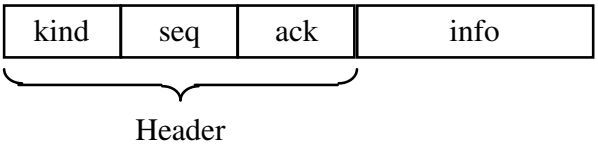
Problema 2	se sparisce l'ack, il destinatario può trovarsi due (o più) copie dello stesso frame;
Soluzione 2	il mittente inserisce un numero di sequenza all'interno di ogni frame dati.

Un altro aspetto importante, che richiede anch'esso un feedback dal destinatario al mittente, è il controllo del flusso, onde impedire che il mittente spedisca dati più velocemente di quanto il destinatario sia in grado di gestirli. Spesso si usano meccanismi basati sull'esplicita autorizzazione, data da parte del destinatario al mittente, di inviare un ben preciso numero di frame.

Vedremo ora le caratteristiche di diversi protocolli, di complessità crescente, per il livello data link.

Prima però dobbiamo fare alcune assunzioni:
- nei livelli fisico, data link e network, ci sono processi (HW o SW) indipendenti, che comunicano fra loro, ad esempio scambiandosi messaggi;
- quando il SW di livello data link riceve un pacchetto dal livello network, lo incapsula in un header ed un trailer contenenti informazioni di controllo; quindi vengono calcolati il checksum e i delimitatori (di norma a cura di un HW apposito di livello data link);
- il frame viene passato al livello sottostante, che trasmette il tutto;
- in ricezione, l' HW di livello data link identifica i delimitatori, estrae il frame, ricalcola il checksum:
    - se è sbagliato, il SW di livello data link viene informato dell'errore;
    - altrimenti il SW di livello data link riceve il frame (senza più checksum).
- il SW di livello data link, quando riceve un frame, esamina le informazioni di controllo (ossia lo header e il trailer):
    - se è tutto OK consegna il pacchetto, e solo quello, al livello network;
    - altrimenti intrapende quanto è necessario fare per recuperare la situazione e non consegna il pacchetto al livello network.

Nei nostri esempi, la struttura di un frame è questa:



I campi del frame hanno le seguenti funzioni:

kind	servirà a distinguere il tipo di frame (contiene dati, è solo di controllo, ecc.)
seq	contiene il numero progressivo del frame
ack	contiene informazioni legate all'acknowledgement
info	contiene un pacchetto di livello network completo (comprendente quindi le informazioni di controllo di tale livello).

Naturalmente, i vari algoritmi useranno via via un numero maggiore di tali informazioni.

3.3.1) Protocollo 1: Heaven

Questo protocollo, per canale simplex, è molto semplice ed è basato sulle ipotesi (non realistiche) che:

- i frame dati vengono trasmessi in una sola direzione;
- le peer entity di livello network sono sempre pronte (non devono mai attendere per inviare o ricevere al/dal livello data link);
- si ignora il tempo di elaborazione del livello data link;
- c'è spazio infinito per il buffering nel ricevitore;
- il canale fisico non fa mai errori.

Il protocollo consiste di due procedure, relative rispettivamente al mittente e al destinatario.

Mittente (loop infinito):

```
1) attende un pacchetto dal livello network;
2) costruisce un frame dati;
3) passa il frame al livello fisico;
4) torna ad 1).
```

Destinatario (loop infinito):

```
1) attende evento:
     * arriva frame da livello fisico:
         2) estrae pacchetto;
         3) lo passa al livello network;
         4) torna ad 1).
```

3.3.2) Protocollo 2: Simplex Stop and Wait

Rilasciamo l'ipotesi (poco realistica) che esista un buffer infinito nel ricevitore. Tutte le altre rimangono, incluso il fatto che il traffico dati viaggia in una sola direzione. La novità è che il mittente deve essere opportunamente rallentato. Ciò però non si può fare con ritardi prefissati: sarebbe troppo gravoso, perché questi dovrebbero essere calibrati sul caso pessimo, che non sempre si verificherà.

La soluzione consiste nell'invio, da parte del destinatario, di una esplicita autorizzazione all'invio del prossimo frame. Questo tipo di protocolli, nei quali il mittente attende un OK dal destinatario, si chiamano ***stop and wait***.

Mittente (loop infinito):

```
1) attende un pacchetto dal livello network;
2) costruisce un frame dati;
3) passa il frame al livello fisico;
4) attende evento:
     * arriva frame di ack (vuoto):
         5) torna ad 1).
```

Destinatario (loop infinito):

```
1) attende evento:
     * arriva frame dati da livello fisico:
         2) estrae il pacchetto;
         3) consegna il pacchetto al livello network;
         4) invia un frame di ack (vuoto) al mittente;
         5) torna ad 1).
```

Si noti che, sebbene il traffico dati viaggi in una sola direzione, i frame viaggiano in entrambe, dunque ci vuole un canale almeno half-duplex (c'è alternanza stretta nelle due direzioni).

3.3.3) Protocollo 3: simplex per canale rumoroso

Assumiamo ora che il canale possa fare errori. I frame (dati o di ack) possono essere danneggiati o persi completamente. Se un frame arriva danneggiato, l'HW di controllo del checksum se ne accorge e informa il SW di livello data link; se il frame si perde del tutto, ovviamente, la cosa non si rileva.

L'aggiunta di un timer al protocollo 2) può bastare? Cioé, è adeguato uno schema quale il seguente?

- quando il mittente invia un frame dati, fa anche partire un timer; se non arriva l'ack entro la scadenza del timer, invia nuovamente il frame;
- il destinatario invia un ack quando un frame dati arriva senza errori;

Questo semplice schema in realtà non basta, infatti può verificarsi la seguente sequenza di eventi:
1. il frame dati x arriva bene;
2. l'ack del frame x si perde completamente (gli errori non discriminano tra frame dati e frame di ack);
3. il frame dati x viene inviato di nuovo e arriva bene;
4. il livello network di destinazione riceve due copie di x, errore!

E' necessario che il destinatario possa riconoscere gli eventuali doppioni. Ciò si ottiene sfruttando il campo ***seq*** dell'header, dove il mittente mette il numero di sequenza del frame dati inviato.

Notiamo che basta un bit per il numero di sequenza, poiché l'unica ambiguità in ricezione è tra un frame ed il suo immediato successore (siano ancora in stop and wait): infatti, fino a che un frame non viene confermato, è sempre lui ad essere ritrasmesso, altrimenti è il suo successore.

Dunque, sia mittente che destinatario useranno, come valori per i numeri di sequenza, la successione

```
...01010101...
```

Il mittente trasmette i frame dati alternando zero ed uno nel campo seq; passa a trasmettere il prossimo frame solo quando riceve l'ack di quello precedente.

Il destinatario invia un frame di ack per tutti quelli ricevuti senza errori, ma passa al livello network solo quelli con il numero di sequenza atteso.

Per quanto riguarda le possibilità che i frame dati arrivino rovinati o non arrivino affatto, un meccanismo basato su timer va bene, bisogna però che esso sia regolato in modo da permettere sicuramente l'arrivo dell'ack, pena la ritrasmissione errata di un duplicato del frame, che può creare grossi guai (vedremo in seguito).

Protocolli come questo, in cui il mittente aspetta un ack di conferma prima di trasmettere il prossimo frame, si chiamano ***PAR (Positive Ack with Retransmission)*** o ***ARQ (Automatic Repeat Request)***.

Mittente (loop infinito; [seq] rappresenta il campo seq di un frame):

```
0) n_seq = 0;
1) n_seq = 1 - n_seq;
2) attende un pacchetto dal livello network;
3) costruisce frame dati e copia n_seq in [seq];
4) passa il frame dati al livello fisico;
5) resetta il timer;
6) attende un evento:
     * timer scaduto: torna a 4)
     * arriva frame di ack (vuoto) non valido: torna a 4)
     * arriva frame di ack (vuoto) valido: torna ad 1)
```

Destinatario (loop infinito; [seq] rappresenta il campo seq di un frame):

```
0) n_exp = 1;
1) attende evento;
     * arriva frame dati valido da livello fisico:
         2) se ([seq] == n_exp)
            2.1) estrae pacchetto
            2.2) lo consegna al livello network
            2.3) n_exp = 1 - n_exp
         3) invia frame di ack (vuoto)
         4) torna ad 1)
     * arriva frame non valido: torna ad 1)
```

In sintesi:

- Il mittente etichetta i frame dati con la sequenza `...0,1,0,1...`, ma passa all'etichetta e frame successivi solo quando arriva un ack; finché ciò non succede, continua a ritrasmettere lo stesso frame.
- Il ricevente invia un ack di conferma per tutti i frame dati privi di errori, ma consegna al livello network solo quelli giusti, e cioé etichettati secondo la sequenza `...0,1,0,1...`.

I disegni seguenti illustrano varie eventualità che possono verificarsi durante il dialogo secondo il protocollo 3. I numeri ai lati delle figure indicano i numeri di sequenza che, rispettivamente:

- il mittente usa per etichettare il frame dati da trasmettere;
- il destinatario usa per decidere se il prossimo frame che arriva va passato al livello network o no.

In assenza di errori il funzionamento è il seguente:

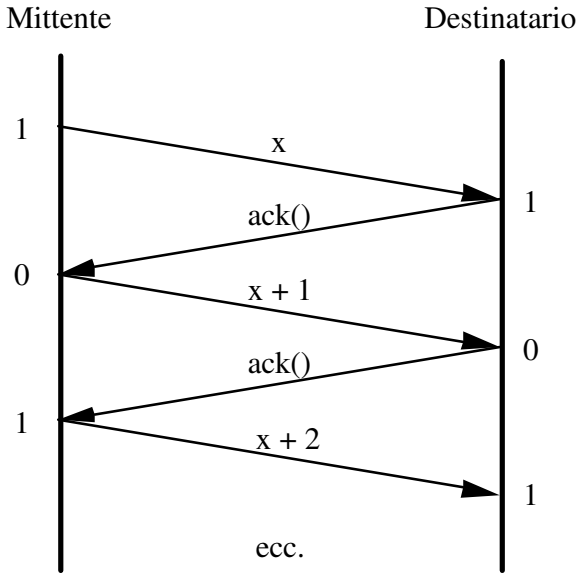


Figura 3-2: Normale funzionamento

Nel caso in cui un frame dati si perda o si danneggi, la situazione è la seguente:

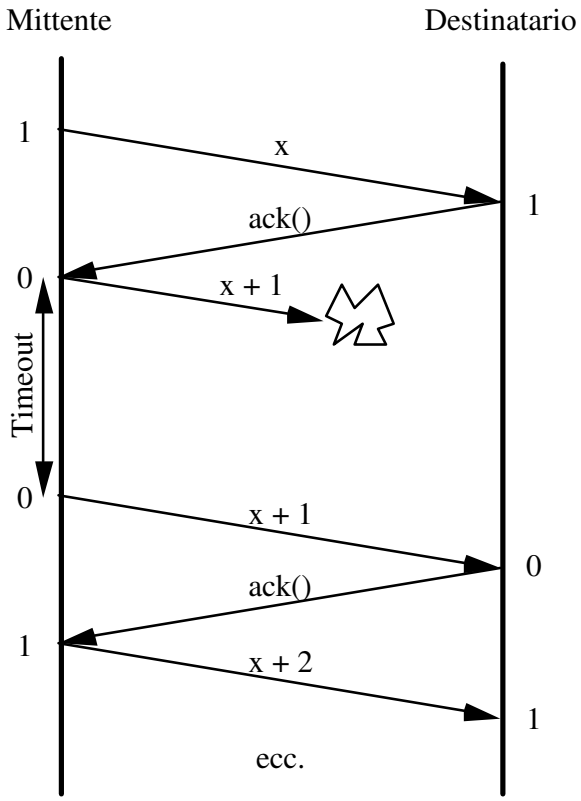


Figura 3-3: Perdita (o danneggiamento) di un frame

Nel caso in cui invece si perda o si danneggi un frame di ack, la situazione è la seguente:

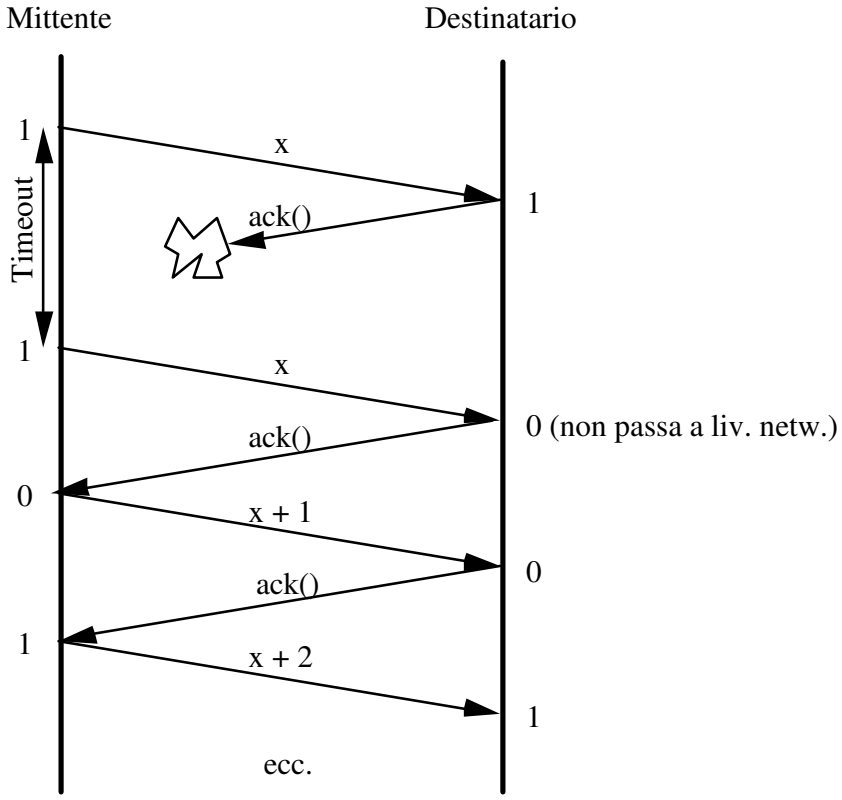


Figura 3-4: Perdita (o danneggiamento) di un ack

Un problema importante è legato, come abbiamo già accennato, alla lunghezza del timeout. Se esso è troppo breve, può succedere questo:

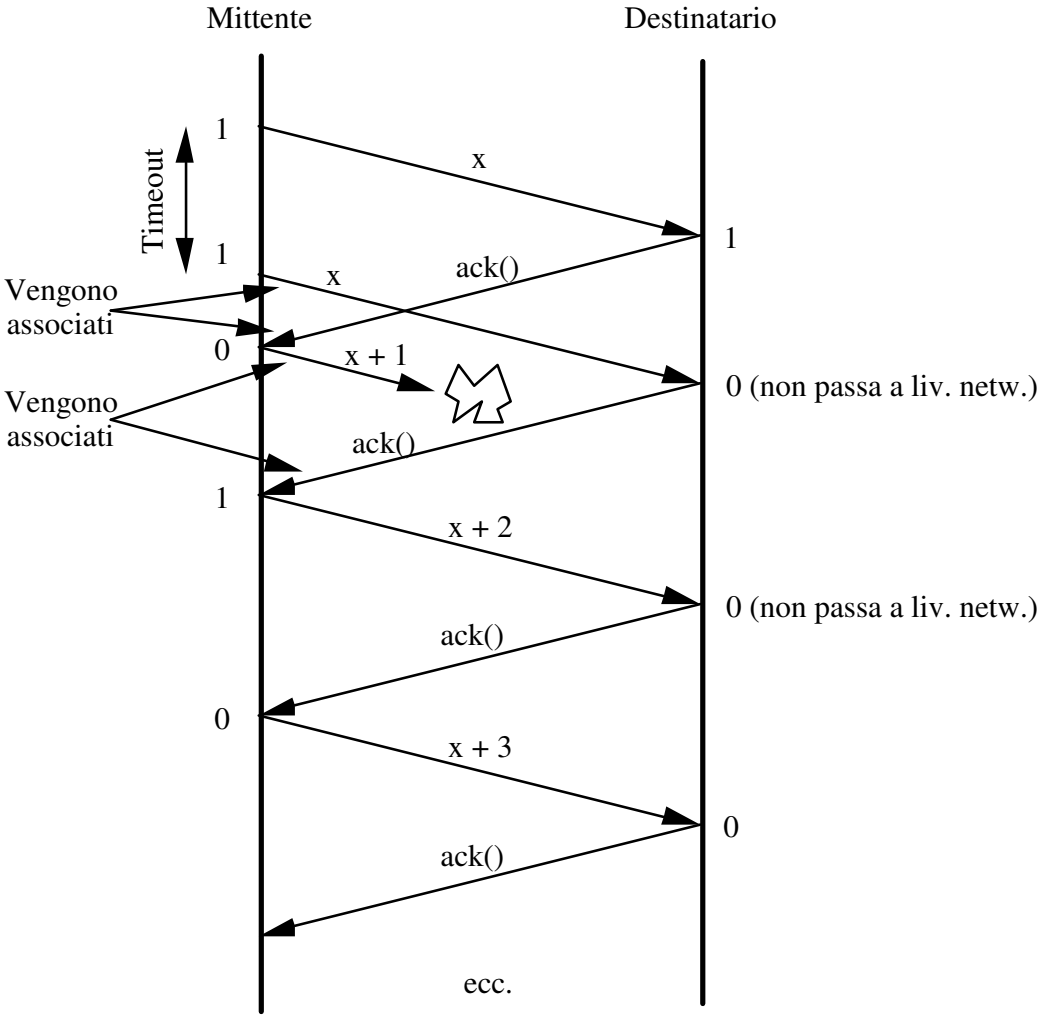


Figura 3-5: Timeout troppo ridotto

Nell'esempio, i frame dati (x+1) e (x+2) si perdono (nel senso che non vengono consegnati al livello network) e nessuno se ne accorge.

D'altronde, se cambiamo il protocollo mandando un frame di ack solo per i frame dati che non sono duplicati, ci si può bloccare:

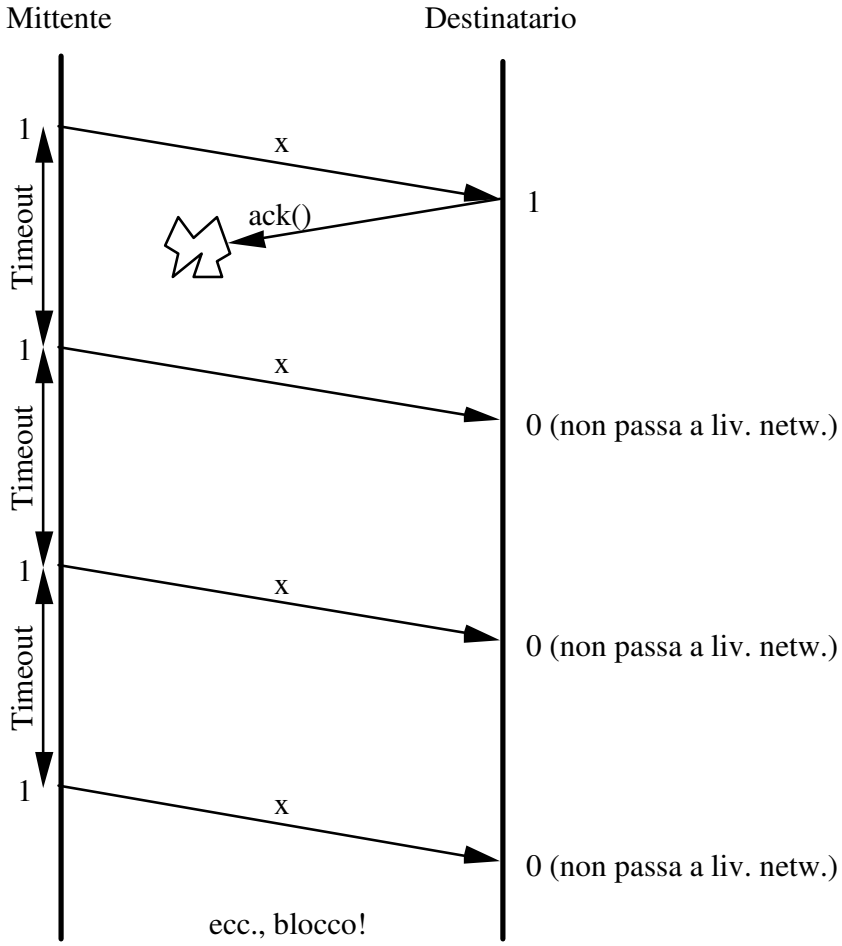


Figura 3-6: Ack inviato solo per frame non duplicati

3.3.4) Protocolli a finestra scorrevole

Nei casi precedenti i frame dati viaggiano in una sola direzione e i frame di ack nella direzione contraria, quindi esistono dei frame che viaggiano in entrambe le direzioni.

Dunque, volendo stabilire una comunicazione dati bidirezionale è necessario disporre di due circuiti, il che ovviamente è uno spreco di risorse. Un'idea migliore è usare un solo circuito, nel quale far convivere tutte le esigenze.

Supponendo che la comunicazione sia fra A e B, si avrà che:

- nella direzione da A a B viaggiano i frame dati inviati da A a B e i frame di ack inviati da A a B (in risposta ai frame dati inviati da B ad A);
- nella direzione da B a A viaggiano i frame dati inviati da B a A e i frame di ack inviati da B a A (in risposta ai frame dati inviati da A ad B);
- il campo ***kind*** serve a distinguere fra i due tipi di frame, dati e di ack, che viaggiano nella stessa direzione.

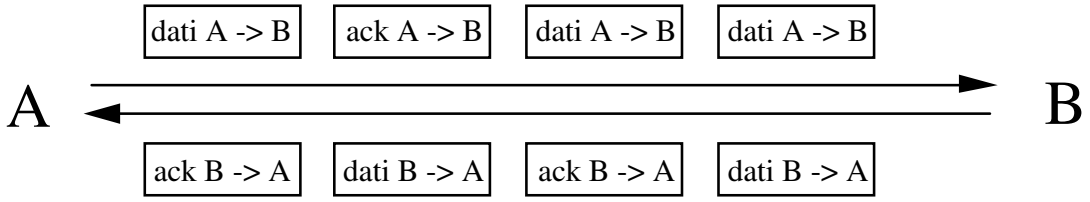


Figura 3-7: Comunicazione dati bidirezionale su unico canale

Però c'è un'idea ancora migliore: se, quando si deve inviare un ack da B ad A, si aspetta un pò fin che è pronto un frame dati che B deve inviare ad A, si può "fare autostop" e mettere dentro tale frame dati anche le informazioni relative all'ack in questione. Questa tecnica si chiama ***piggybacking*** (letteralmente, portare a spalle).

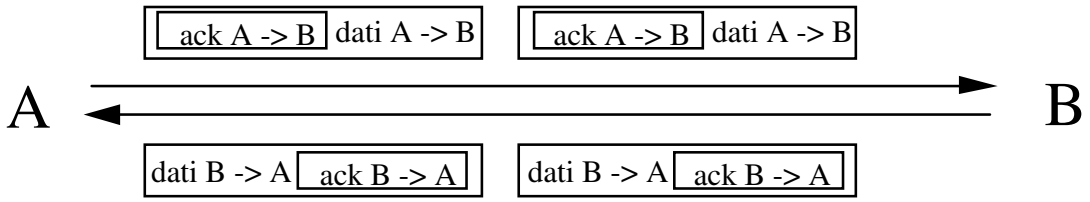


Figura 3-8: Piggybacking

Il campo ***ack*** serve proprio a questo scopo, infatti è il campo in cui viene trasportato, se c'è, un ack.

Questa tecnica consente un notevole risparmio di:
- banda utilizzata;
- uso di cpu.

Infatti, con questa tecnica le informazioni di ack non richiedono la costruzione di un apposito frame (e quindi il tempo necessario alla creazione ed al riempimento della struttura, al calcolo del checksum, ecc.) né la sua trasmissione (e quindi l'uso di banda).

Però c'è un aspetto da non trascurare: per quanto si può aspettare un frame su cui trasportare un ack che è pronto e deve essere inviato? Non troppo, perché se l'ack non arriva in tempo il mittente ritrasmetterà il frame anche se ciò non è necessario. Dunque si stabilisce un limite al tempo di attesa di un frame sul quale trasportare l'ack; trascorso tale tempo si crea un frame apposito nel quale si mette l'ack.

I protocolli che vedremo ora appartengono alla classe dei ***protocolli sliding window*** (finestra scorrevole), sono full-duplex (per i dati), sfruttano il piggybacking e sono più robusti di quelli precedenti.

Differiscono fra loro per efficienza, complessità e capacità dei buffer. Alcuni aspetti però sono comuni a tutti gli algoritmi:

- ogni frame inviato ha un numero di sequenza, da 0 a $2^n - 1$ (il campo ***seq*** è costituito da n bit);
- ad ogni istante il mittente mantiene una finestra scorrevole sugli indici dei frame, e solo quelli entro la finestra possono essere trasmessi. I numeri di sequenza entro la finestra rappresentano frame da spedire o spediti, ma non ancora confermati:
    - quando arriva dal livello network un pacchetto, un nuovo indice entra nella finestra;
    - quando arriva un ack, il corrispondente indice esce dalla finestra;
    - i frame dentro la finestra devono essere mantenuti in memoria per la possibile ritrasmissione; se il buffer è pieno, il livello data link deve costringere il livello network a sospendere la consegna di pacchetti;
- analogamente, il destinatario mantiene una finestra corrispondente agli indici dei frame che possono essere accettati:
    - se arriva un frame il cui indice è fuori dalla finestra:
        - il frame viene scartato (e non si invia il relativo ack);
    - se arriva un frame il cui indice è entro la finestra:
        - il frame viene accettato;
        - viene spedito il relativo ack;
        - la finestra viene spostata in avanti;
    - si noti che la finestra del destinatario rimane sempre della stessa dimensione, e se essa è pari a 1 il livello accetta i frame solo nell'ordine giusto (ma per dimensioni maggiori di 1 questo non è più detto).
- le finestre di mittente e destinatario non devono necessariamente avere uguali dimensioni, né uguali limiti inferiori o superiori.

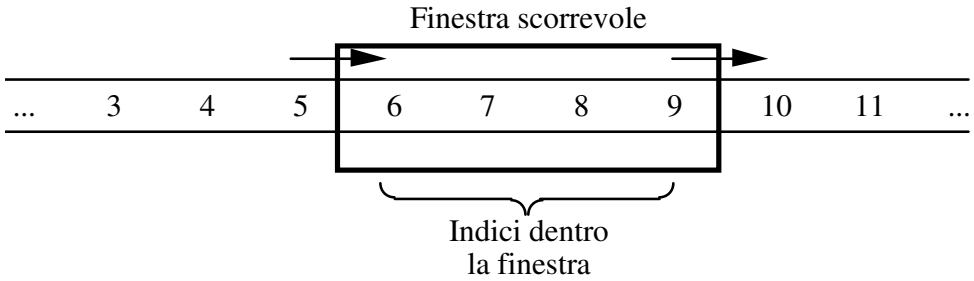


Figura 3-9: Finestra scorrevole sugli indici dei frame

In questi protocolli il livello data link ha più libertà nell'ordine di trasmissione, fermo restando che:

- i pacchetti devono essere riconsegnati al livello network nello stesso ordine di partenza;
- il canale fisico è ***wire-like***, cioé consegna i frame nell'ordine di partenza.

Protocollo a finestra scorrevole di un bit

In questo protocollo sia mittente che destinatario usano una finestra scorrevole di dimensione uno. Di fatto questo è un protocollo stop-and-wait.

Non c'è differenza di comportamento fra mittente e destinatario, tutt'e due usano lo stesso codice (questo vale anche per i protocolli successivi).

Il funzionamento, molto semplice, è il seguente:

- il mittente, quando invia un frame, fa partire un timer:
    - se prima che scada il timer arriva un ack con lo stesso numero di sequenza del frame che si sta cercando di trasmettere, si avanza la finestra e si passa a trasmettere il frame successivo;
    - se arriva un ack diverso o scade il timer, si ritrasmette il frame;
- il destinatario invece:
    - quando arriva un frame corretto, senza errori, invia un ack col corrispondente numero di sequenza;
    - se il frame non è un duplicato lo passa al livello network e avanza la finestra.

Qui sta la principale novità rispetto al protocollo 3: l'ack è etichettato col numero di sequenza del frame a cui si riferisce. I valori dell'etichetta possono solo essere 0 e 1, come nel protocollo 3.

Il peggio che può succedere è la ritrasmissione inutile di qualche frame, ma questo protocollo è sicuro.

Protocolli go-back-n e selective repeat
Se il tempo di andata e ritorno del segnale (***round-trip time***) è alto, come ad esempio nel caso dei canali satellitari nei quali è tipicamente pari a 500 + 500 msec, c'è un enorme inefficienza coi protocolli stop-and-wait, perché si sta quasi sempre fermi aspettando l'ack.

Per migliorare le cose, si può consentire l'invio di un certo numero di frame anche senza aver ricevuto l'ack del primo. Questa tecnica va sotto il nome di ***pipelining***.

Ciò però pone un serio problema, perché se un frame nel mezzo della sequenza si rovina molti altri frame vengono spediti prima che il mittente sappia che qualcosa è andato storto.

Il primo approccio al problema è quello del protocollo ***go-back-n***:

- se arriva un frame privo di errori e col numero di sequenza uguale a quello che il destinatario è pronto ad accettare, il destinatario accetta il frame, lo passa al livello superiore ed invia al mittente un ack con il numero di sequenza del frame testé accettato. Inoltre si predispone ad accettare un frame con il numero di sequenza successivo (ossia sposta avanti di uno la sua finestra);
- se arriva un frame danneggiato, il destinatario ignora tale frame, non inviando alcun ack;
- se arriva un frame privo di errori ma con un numero di sequenza diverso da quello che il destinatario è pronto ad accettare, esso non accetta tale frame ed invia un ack di conferma contenente l'indice dell'ultimo (in ordine di tempo) frame accettato. Ciò corrisponde ad una finestra di dimensione uno nel ricevitore, che quindi accetta i frame solo nell'ordine giusto;
- il mittente ad un certo punto va in time-out sul frame danneggiato dagli errori, e poi su tutti quelli successivi (scartati dal destinatario), e quindi provvede a ritrasmettere la sequenza di frame che inizia con quello per il quale si è verificato il primo time-out.

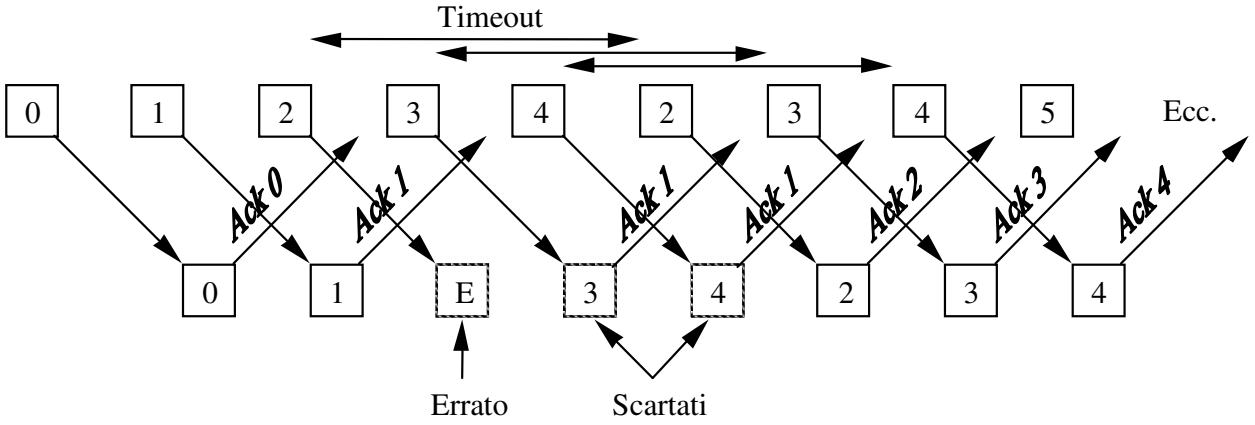


Figura 3-10: Funzionamento del protocollo go-back-n

L'arrivo di un ack con un certo valore conferma implicitamente anche l'arrivo di tutti i frame precedenti; questa tecnica si indica col termine ***cumulative ack***, ed è necessaria per evitare un problema che si potrebbe verificare se il destinatario non inviasse alcun ack di conferma per i

frame che non accetta. Supponiamo che il frame X arrivi, sia accettato e confermato. Il ricevente si aspetterà successivamente il frame (X + 1), e solo quello accetterà e confermerà. Ora, supponiamo che disgraziatamente l'ack di X si perda e quindi non raggiunga il mittente. Esso andrà in timeout e ritrasmetterà X, che però non sarà né accettato né confermato dal destinatario, per cui il mittente continuerà all'infinito a ritrasmettere il frame X.

Il mittente deve mantenere in un apposito buffer tutti i frame non confermati per poterli eventualmente ritrasmettere. Se il buffer si riempie, il mittente deve bloccare il livello network fino a che non si ricrea dello spazio. Inoltre, vi è spreco di banda se il tasso d'errore è alto e/o il time-out è lungo.

Si noti che col protocollo go-back-n il mittente è obbligato a non superare il limite di (k-1) frame trasmessi ma non ancora confermati, dove k è il numero dei valori possibili degli indici dei frame. Ad esempio, con 3 bit per rappresentare gli indici dei frame i valori di tali indici sono otto (da 0 a 7). Il mittente non è autorizzato ad avere più di 7 frame spediti ma non confermati. Per comprenderne la ragione si consideri questa situazione.

Ammettiamo che, con indici a tre bit, il mittente possa spedire otto frame consecutivi. Supponiamo che esso invii la sequenza di frame 0, 1, 2, … 7 e che tutti questi vengano confermati. Dunque, l'ultimo ack di conferma ricevuto avrà il valore 7. Dopo aver ricevuto tale ack=7, il mittente passa a trasmettere i successivi otto frame, di nuovo con indici di sequenza 0, 1, 2, …, 7. Ora si supponga che a fronte della trasmissione di questo secondo gruppo di otto frame il mittente riceva un solo ack, col valore 7. Che cosa è successo:

- tutti i frame del secondo gruppo sono arrivati bene, ma solo l'ultimo ack è arrivato al mittente (gli altri si sono persi), oppure:
- un solo frame del secondo gruppo (e purtroppo non quello con indice 0, ossia il primo del gruppo) è arrivato al destinatario, che usando il cumulative ack ha confermato di nuovo l'ultimo frame del gruppo precedente?

Il mittente non ha modo di sapere quale delle due situazioni si è verificata, e quindi il protocollo fallisce. Se invece il mittente trasmette a gruppi di sette frame ciascuno:

primo gruppo: 0,1,2,3,4,5,6
secondo gruppo: 7,0,1,2,3,4,5
terzo gruppo: 6,7,0,1,2,3,4
ecc.

E' facile vedere che tale problema non si può verificare, perché se dopo l'inizio della trasmissione del secondo gruppo torna un ack=6 esso è il cumulative ack del gruppo precedente (l'indice 6 non è presente nel secondo gruppo), mentre un qualunque altro valore di ack si riferisce al gruppo di frame in fase di trasmissione.

Il secondo approccio è più efficiente, ed è chiamato ***selective repeat***:

- il destinatario mantiene nel suo buffer tutti i frame ricevuti successivamente ad un eventuale frame rovinato; non appena questo arriva nuovamente (senza errori), esso e tutti i successivi frame contigui che il destinatario ha mantenuto nel buffer vengono consegnati al livello network;
- per ogni frame arrivato senza errori, il destinatario invia un ack col numero più alto della sequenza completa arrivata fino a quel momento (cumulative ack);
- quando si verifica un timeout, il mittente rispedisce il frame corrispondente.

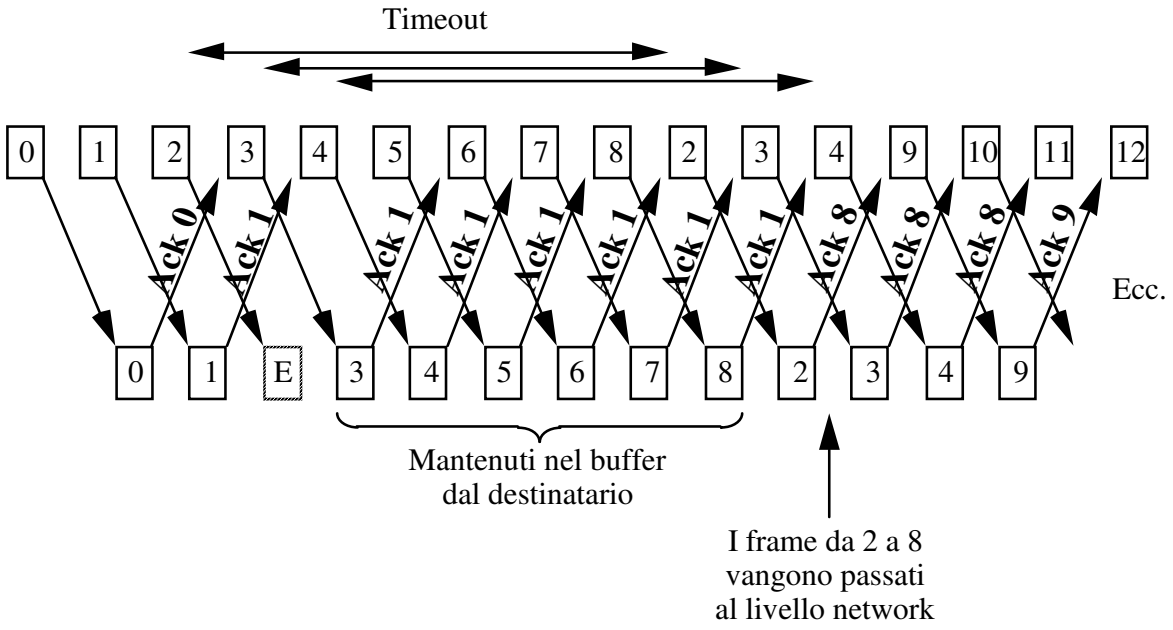


Figura 3-11: Funzionamento del protocollo selective repeat

Alcune considerazioni a proposito del protocollo selective repeat:

- mittente e destinatario devono entrambi gestire un buffer per mantenervi i frame:
  - non confermati (mittente);
  - successivi ad un errore (destinatario);
- vi è un basso spreco di banda, che si si può ulteriormente diminuire mandando un ***NACK*** (***Negative ACKnowledgement***) quando:
  - arriva un frame danneggiato;
  - arriva un frame diverso da quello atteso (ciò può indicare l'avvenuta perdita del frame precedente).

Si noti che col protocollo selective repeat, per ragioni simili a quelle viste nel caso del go-back-n, il mittente è obbligato a non superare il limite di (k/2) frame trasmessi ma non ancora confermati, dove k è il numero dei valori possibili degli indici dei frame.

Infine, si noti che per entrambi i precedenti protocolli:

- è necessaria la gestione di timer multipli (uno per ogni frame inviato e non confermato);
- il ricevente, per inviare gli ack, usa il piggybacking se possibile, altrimenti invia un apposito frame.

3.4) Esempi di protocolli data link

I protocolli data link più diffusi oggi sono discendenti del protocollo ***SDLC*** (***Synchronous Data Link Control***), nato nell'ambito dell'architettura SNA. Nel seguito verranno illustrate brevemente le caratteristiche di tre diffusi protocolli: ***HDLC*** (standard ISO), ***SLIP*** (architettura TCP/IP) e ***PPP*** (suo successore).

3.4.1) HDLC (High Level Data Link Control)

E' un protocollo bit oriented, e quindi usa la tecnica del bit stuffing. Il formato del frame HDLC è illustrato nella figura seguente.

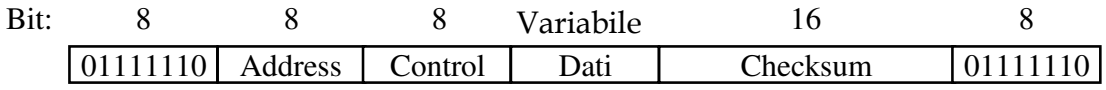


Figura 3-12: Frame HDLC

I campi del frame hanno le seguenti funzioni:

Address	utilizzato nelle linee multipunto, dove identifica i diversi terminali (il protocollo offre funzioni per il dialogo fra un concentratore e diversi terminali)
Control	contiene numeri di sequenza, ack, ecc.
Dati	contiene i dati da trasportare
Checksum	è calcolata con CRC-CCITT

Le caratteristiche salienti sono le seguenti:

- usa una finestra scorrevole con numeri di sequenza a 3 bit, contenuti dentro un campo ***Seq*** situato all'interno del campo Control;
- utilizza il campo ***Next***, anch'esso contenuto in Control, per il piggybacking degli ack;
- ha tre tipi di frame (identificati dai primi due bit di Control):
    - ***Information***, per la trasmissione dati;
    - ***Supervisory***, per comandare diverse modalità di ritrasmissione;
    - ***Unnumbered*** (non ha il numero di sequenza), per finalità di controllo o per trasportare il traffico di connessioni non affidabili.

3.4.2) SLIP (Serial Line IP)

E' nato nel 1984 ed è il più vecchio protocollo di livello data link dell'Internet Protocol Suite.

Molto semplice, nacque per collegare via modem macchine Sun ad Internet. Spedisce sulla linea pacchetti IP terminati col byte `0xC0`. Usa character stuffing.
Ha diverse limitazioni:

- non c'è controllo degli errori;
- supporta solo IP, e per di più solo indirizzi statici;
- non è uno standard ufficiale di Internet.

3.4.3) PPP (Point to Point Protocol)

Per migliorare le cose, IETF ha prodotto uno standard ufficiale, il ***Point to Point Protocol*** (RFC 1661, 1662 e 1663). Esso è adatto sia a connessioni telefoniche che a linee router-router.

Esso fornisce le seguenti funzionalità:

- framing;
- rilevamento degli errori;
- un protocollo di controllo per attivare, testare e diasattivare la linea (***LCP, Link Control Protocol***);
- supporto di molteplici protocolli di livello network;
- un protocollo per negoziare opzioni di livello network (***NCP, Network Control Protocol***):
    - per ogni livello network supportato c'è un differente NCP;
    - ad esempio, nel caso di IP, NCP viene usato per negoziare un indirizzo IP dinamico;

Il traffico derivante (nelle fasi iniziali e finali della connessione) dall'uso dei protocolli LCP e NCP viene trasportato dentro i frame PPP.

Il protocollo è modellato su HDLC, ma con alcune differenze:

- è character-oriented anziché bit-oriented, e utilizza il character stuffing (quindi i frame sono costituiti da un numero intero di byte);
- c'è un campo apposito per il supporto multiprotocollo offerto al livello network.

Il formato del frame PPP è il seguente.

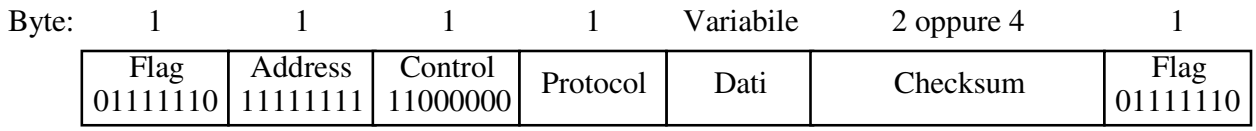


Figura 3-13: Frame PPP

I campi del frame hanno le seguenti funzioni:

Flag	come in HDLC
Address	sempre 11111111: di fatto non ci sono indirizzi, in quanto non c'è più l'idea di gestire linee multipunto
Control	il default (11000000) indica un unnumbered frame, quindi relativo ad un servizio non affidabile
Protocol	indica il protocollo relativo al pacchetto che si trova nel payload (LCP, NCP, IP, IPX, Appletalk, ecc.)
Payload	è di lunghezza variabile e negoziabile, il default è 1500 byte
Checksum	normalmente è di due byte (quattro sono negoziabili)