

## Livello due (Livello data link)

### 3.1 Generalità:

Questo livello ha il compito di offrire una comunicazione affidabile ed efficiente a due macchine adiacenti, cioè connesse fisicamente da un canale di comunicazione. I bit partono e arrivano nello stesso ordine.

Le competenze di questo livello sono :

- offrire servizi al livello network
- il framing
- gestire gli errori di trasmissione
- regolare il flusso tra sorgente e destinazione

Il servizio principale offerto è di trasferire i dati dal livello network della sorgente al livello network della destinazione gli altri servizi sono:

- connectionless non confermato:
  - si mandano frame indipendenti
  - i frame non vengono confermati
  - non si stabilisce una connessione
  - i frame persi non si recuperano
  - ottimo per canali con tasso d'errore minimo, traffico real-time, LAN
- connectionless confermato:
  - i frame vengono confermati
  - se la conferma non arriva vengono rispediti i frame
  - utile nei canali non affidabili
- connection oriented confermato
  - tre fasi per la connessione (apertura/invio dati/chiusura)
  - garantisce che ogni frame sia ricevuto una sola volta e nell'ordine giusto
  - flusso di bit affidabile per il livello network

### 3.2 Framing:

Per poter individuare l'inizio e la fine di un frame occorre in qualche modo delimitarli, perché sarebbe troppo rischioso utilizzare lo spazio che intercorre tra un frame ed un altro per tale delimitazione.

Vari metodi:

- conteggio caratteri
- character stuffing
- bit stuffing
- violazioni della codifica

#### 3.2.1 Conteggio dei caratteri:

Viene utilizzato un campo nell'header che indica di quanti caratteri è composto il frame, questa tecnica comporta però che se si rovina il campo in questione diventa impossibile ricostruire il frame e quello successivo.

#### 3.2.2 Character stuffing:

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

una scelta diffusa è:

inizio frame : DLE (Data Link Escape), STX(Start of Text)

fine frame: DLE , ETX (End of Text)

Nasce un problema se all'interno dei dati da trasmettere i bit con cui sono codificati questi tipi di marcatori sono presenti perché parte integrante dei dati, in questo caso il livello due sorgente ha il compito di inserire un nuovo marcatore DLE prima di quello presente nel campo dati.

### 3.2.3 Bit stuffing:

Simile al character stuffing ma effettuato attraverso una sequenza di bit chiamata flag byte, anche in questo caso si potrebbe presentare il problema di prima, in questa situazione quando si incontra una sequenza di bit uguale a quella del flag byte viene introdotto uno 0, mentre in ricezione quando c'è una sequenza uguale a quella del flag byte viene rimosso il primo 0 che segue.

### 3.2.4 Violazione della codifica:

In molte reti LAN i bit vengono codificati con delle transazioni del tipo alto/basso per 1 e basso/alto per 0, non vengono mai utilizzate transazioni alto/alto e basso/basso, che possono quindi essere utilizzate per la delimitazione dei frame.

## 3.3 Rilevamento e correzione errori:

Molte possono essere le cause d'errore nella trasmissione dati, tra le quali c'è il rumore di fondo, disturbi improvvisi come i fulmini e le interferenze ad esempio motori elettrici.

In tutti questi casi sono possibili due approcci per la risoluzione del problema, o l'errore viene corretto oppure solamente rilevato.

### Correzione:

Normalmente un frame consiste di  $n=m+r$  bit dove  $m$  sono i dati veri e propri e  $r$  sono dei bit di controllo detti redundant bit o check bit.

Una sequenza di  $n$  bit composti in questo modo si chiama *codeword*, prese due qualunque codeword è possibile determinare il numero di bit in cui differiscono attraverso un semplice xor, il risultato viene detto *distanza di Hamming*.

Questo valore è utile per determinare quanti errori ci vogliono per trasformare la codeword di partenza nell'altra.

Un insieme prefissato di codeword si dice codice, la distanza di Hamming del codice corrisponde al valore minimo di tutte le distanze di Hamming fra le possibili coppie di codeword.

Per il rilevamento di  $d$  errori occorre una distanza di Hamming pari a  $d+1$ , in modo tale che non sia possibile ottenere nessun'altra codeword valida del codice.

Per la correzione di  $d$  errori occorre una distanza di Hamming pari a  $2d+1$ , poichè in questo caso qualunque codeword con  $d$  errori è più vicino a quello originario che a qualunque altro.

Il bit di parità: E' un codice con distanza di Hamming pari a 2 quindi in grado di rilevare solamente 1 errore.

### Rilevamento:

Per poter rilevare burst di errori di lunghezza  $k$  occorre accumulare  $k$  codeword riga per riga, i codeword si trasmettono per colonne, quando arrivano a destinazione si riassemblano per righe, un Burst di  $k$  errori comporta così un errore in ciascun codeword che può quindi essere ricostruito.

Per quanto riguarda il rilevamento dell'errore ci sono alcuni codice largamente diffusi, e sono il CRC standard e nelle sue varianti a 12 e 16 bit ed il CRC-CCITT.

Il CRC è un codice polinomiale, in questo codice le stringhe di bit sono viste come polinomi a coefficienti 0 ed 1. Il funzionamento di questo sistema prevede inizialmente l'accordo tra mittente e destinatario su un codice generatore  $G(x)$  di  $r$  bit e che termini con 1, a questo punto al frame  $M(x)$ , necessariamente più lungo di  $G(x)$ , viene appeso in coda un checksum tale che il polinomio risultante sia di grado  $m+r-1$  e che tale polinomio sia divisibile per  $G(x)$ .

Quando il destinatario riceve il frame divide il tutto per  $G(x)$  e se il risultato è diverso da 0 c'è stato un errore.

Per calcolare il checksum si eseguono queste operazioni:

- si appendono  $r$  bit a destra del frame, che quindi ha  $m+r$  bit. e corrisponde ad  $x^r M(x)$
- bisogna dividere  $x^r M(x)$  per  $G(x)$
- sottrarre ad  $x^r M(x)$  il resto della divisione quello che si ottiene è proprio il checksum.

Con questo metodo è possibile individuare errori singoli e doppi, errori di  $x$  bit con  $x$  dispari, burst di errori di lunghezza  $\leq r$ .

I più comuni polinomi sono:

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

### 3.4 Gestione sequenza trasmissione e controllo di flusso:

Dopo aver trovato il modo di delimitare i frame occorre trovare il modo di confermare la corretta ricezione di questi nei servizi di tipo connectionless confermato e connection oriented.

A questo scopo si introduce il concetto di *acknowledgement*, ossia di conferma dell'avvenuta ricezione o della mancata ricezione; per poter dare queste conferme si usano due frame particolari l'*ack* e il *nack*.

L'introduzione di questi due frame comporta però la presenza di alcuni problemi, che sono:

- se un pacchetto non viene ricevuto non potrà essere mandato né un *ack* né un *nack*; la soluzione sta nello stabilire un timeout per i frame inviati e mai confermati, dopo il timeout il frame viene rispedito.
- se sparisce l'*ack* il destinatario potrebbe ricevere più copie dello stesso frame; la soluzione consiste nell'introduzione di un numero di sequenza all'interno dei frame dati.

Un altro aspetto importante consiste nel controllo di flusso, cioè nell'impedire che il mittente spedisca frame ad una velocità più elevata di quella a cui il destinatario li smaltisce, anche questo problema richiede un feedback dal destinatario.

#### 3.4.1 Protocollo Heaven:

Basato su ipotesi poco realistiche, presuppone difatti che il canale sia completamente privo di rumore e di errori, che i frame siano trasmessi in una sola direzione, ci sia spazio infinito nei buffer, e che le peer entity di livello network siano sempre pronte.

Consiste di due procedure una per il mittente e una per il destinatario:

- Mittente: attende un pacchetto dal livello network, crea il frame, passa il frame a livello fisico, torna in attesa.
- Destinatario: riceve il frame dal livello fisico, estrae il pacchetto, lo passa al livello network e si rimette in attesa.

### 3.4.2 Protocollo Stop and Wait:

Questo protocollo rilascia l'ipotesi che il buffer del ricevitore sia infinito, mentre tutte le altre ipotesi rimangono.

Il protocollo stop and wait deriva il suo nome dal fatto che tra una trasmissione e l'altra il mittente attende la conferma del destinatario.

- Mittente: attende un pacchetto dal livello network; costruisce il frame dati; passa il frame al livello fisico; attende l'ack, se arriva torna in attesa dal livello network.
- Destinatario: arriva il frame dal livello fisico; estrae il pacchetto; lo invia al livello network; invia un frame ack al mittente; torna in attesa.

### 3.4.3 Protocollo Stop and Wait su canale rumoroso:

Viene rilasciata anche l'ipotesi che il canale non introduca errori, quindi i frame possono essere danneggiati o persi completamente. Quando un frame arriva danneggiato l'HW di controllo del checksum informa il SW di livello data link; se il frame si perde, la semplice introduzione del timeout non basta a risolvere il problema.

Difatti si potrebbe presentare questa situazione:

1. il frame dati arriva ;
2. l'ack si perde;
3. il frame viene rispedito e arriva bene;
4. il livello network riceve due copie dello stesso frame, ERRORE!

È quindi necessario introdurre un numero di sequenza che possa identificare ciascun frame, in modo tale da potersi accorgere di frame duplicati; si usa quindi il campo *seq* nell'header. In questo protocollo basta un solo bit per il seq, perché l'unica ambiguità in ricezione è tra un frame e il successivo, quindi il mittente marca alternativamente i frame da spedire con 0 e 1. Il destinatario invia un ack per tutti i frame ricevuti correttamente ma passa al livello network solo quelli con il numero di sequenza atteso.

- Mittente:
  1.  $n\_seq=0$ ;
  2.  $n\_seq=1-n\_seq$ ;
  3. attende un pacchetto dal livello network;
  4. costruisce frame dati e copia  $n\_seq$  in [seq];
  5. passa il frame al livello fisico;
  6. resetta il timer;
  7. attende un evento:
    1. timer scaduto passa a 4;
    2. arriva ack(vuoto) non valido torna a 4;
    3. arriva ack(vuoto) valido: torna a 2;
- Destinatario:
  1.  $n\_exp=1$ ;
  2. attende evento:
    1. arriva frame dati valido :
      1. se ( $[seq]== n\_exp$ )
        1. estrae pacchetto
        2. lo consegna al livello network
        3.  $n\_exp=1-n\_exp$ ;
      2. invia ack
      3. torna a 1
    2. arriva frame non valido torn a 1

Questo tipo di protocolli sono chiamati anche ARQ (Automatic Repeat Request)

#### 3.4.4 Protocolli a finestra scorrevole:

Questi protocolli sono di tipo full duplex, quindi c'è comunicazione fra mittente e destinatario e viceversa contemporaneamente. Viene introdotto un nuovo campo *kind* che serve ad identificare se i frame che sta viaggiando in una direzione sono di ack o dati.

Una tecnica abbastanza sofisticata che serve a ridurre lo spreco di risorse e di tempo nelle comunicazioni è il *piggybacking* ossia il destinatario di un frame quando deve inviare un ack attende che sia pronto un frame dati da inviare al mittente e ci inserisce all'interno anche l'ack. Nasce un problema relativo al tempo di attesa per il frame dati, se dopo un certo tempo questo frame non arriva allora si crea il frame contenente solamente l'ack.

In questa famiglia di protocolli, ogni frame inviato ha un numero di seq da 0 a  $2^{(n-1)}$  ad ogni istante il mittente mantiene una finestra scorrevole sugli indici dei frame e solamente quelli all'interno di questa finestra possono essere trasmessi. Tutti gli indici interni alla finestra rappresentano frame spediti ma non confermati o da spedire, quando un frame viene confermato esce dalla finestra.

Analogamente il destinatario mantiene una finestra corrispondente agli indici dei frame che possono essere accettati, quindi se arriva un frame con indice fuori dalla finestra viene scartato, altrimenti viene accettato, viene spedito un ack e la finestra spostata in avanti.

##### 3.4.4.1. Finestra scorrevole di 1 bit:

Sia mittente che destinatario usano una finestra scorrevole di dimensione 1 quindi questo protocollo è praticamente identico al protocollo stop and wait, l'unica differenza è che il destinatario quando deve inviare l'ack inserisce anche il numero di sequenza del frame a cui si riferisce.

##### 3.4.4.2. GoBack -N:

In caso di canali con grande ritardo (es. quelli satellitari), i protocolli di tipo stop and wait non sono affatto efficienti. Per migliorare queste prestazioni esiste la tecnica del *pipelining* che consiste nell'inviare più frame anche non confermati contemporaneamente.

Questa tecnica pone però un nuovo problema difatti se ad esempio si dovesse perdere un frame nel mezzo di quelli spediti, trascorrerà molto tempo prima che il mittente si renda conto di quello che è successo.

Il protocollo GoBack-N utilizza proprio questa tecnica, ed in particolare il suo comportamento è questo:

- se arriva un frame danneggiato o con numero di seq non progressivo, il destinatario ignora il frame e tutti i successivi, non inviando gli ack. Questo corrisponde ad una finestra di dimensione 1 nel ricevitore che accetta soltanto pacchetti nel giusto ordine.
- il mittente va in timeout sul frame sbagliato e poi su tutti quelli successivi, e provvede a rispedire la sequenza di frame dal primo per il quale si è verificato il timeout.

In questo protocollo c'è necessità da parte del mittente di mantenere un buffer con i frame ancora non confermati, se questo buffer si riempie si è costretti a bloccare il livello network, quindi i danni principali di questo approccio sono lo spreco di banda, e perdita di tempo.

#### 3.4.4.3. *Selective repeat:*

Un secondo approccio è quello del selective repeat:

- il destinatario mantiene nel suo buffer tutti i frame ricevuti successivi ad uno errato, non appena questo arriva senza errori vengono tutti passati al livello network.
  - per ogni frame arrivato bene il destinatario invia un ack col numero più alto della sequenza completa.
  - quando si verifica un timeout il mittente rispedisce solamente il frame scaduto.
- Questo approccio consente un minore spreco di banda, ulteriormente migliorabile con l'uso dei nack quando arriva un frame danneggiato o un frame diverso da quello atteso.

#### 3.4.5 *Protocolli data link:*

##### 3.4.5.1. *HDLC:*

È un protocollo di tipo bit oriented, i campi del frame sono:

<i>Address</i>	utilizzato nelle linee multipunto, identifica i diversi terminali.
<i>Control</i>	contiene numeri di sequenza, ack, ecc.
<i>Dati</i>	contiene i dati da trasportare
<i>Checksum</i>	usa il CRC-CCITT

Questo protocollo usa una finestra scorrevole di 3 bit contenuti dentro al campo seq, situato all'interno del campo control.

Utilizza un campo next anch'esso contenuto in control per il piggybacking.

Ha tre tipi di frame:

1. Information, per la trasmissione dati.
2. Supervisory, per comandare la ritrasmissione
3. Unnumbered, per finalità di controllo o per trasportare traffico di connessioni non affidabili.

##### 3.4.5.2. *SLIP (serial line IP):*

Nato per collegare macchine Sun via modem ad internet, spedisce sulla linea d'uscita pacchetti Ip terminanti col byte 0xC0, e usa il character stuffing.

Questo protocollo non prevede controllo d'errore, supporta solo IP, non è standard ufficiale per internet.

##### 3.4.5.3. *PPP:*

Standard ufficiale dell'IETF utilizzato per le linee telefoniche e per i collegamenti router-router.

Le funzionalità principali sono:

- framing;
- rilevamento errori;
- un protocollo per testare, attivare e disattivare la linea LCP;
- supporto per molteplici protocolli di livello network;
- un protocollo per negoziare opzioni di livello network NCP
  - per ogni livello network supportato c'è un NCP;
  - in IP, NCP serve a negoziare un indirizzo IP dinamico;
- è byte-oriented ;
- modellato su HDLC ;
- utilizza il character stuffing;
- c'è un campo apposito per il supporto multiprotocollo;

Il frame è composto in questo modo:

<i>Flag</i>	come in HDLC
<i>Address</i>	sempre 11111111
<i>Control</i>	di default 00000011 indica un unnumbered frame quindi relativo ad un servizio non affidabile
<i>Protocol</i>	LCP,NCP,IP,IPX,AppleTalk
<i>Dati</i>	di lunghezza variabile di default 1500 byte
<i>Checksum</i>	2 byte