

Nella fibra ottica c'è la multiplazione a divisione di frequenza, l'intera banda viene divisa in canali all'interno di ognuno viene modulato un segnale.

Che cosa limita la velocità di comunicazione in questo tipo di sistema?

- L'attenuazione, la potenza che si riceva a distanza d:

$$P_{rx}(d) = P_{tx} 10^{-\frac{dd}{10}}$$

$$\log_{10} P_{rx}(d) = \log_{10} P_{tx} - dd \text{ dB/km}$$

L'attenuazione si contrasta con gli amplificatori.

- DISPERSIONE**, il segnale in uscita non è proporzionale all'ingresso. I ritardi sono sempre diversi. Questa è legata alla polarizzazione, al materiale ecc. la dispersione si misura con un coefficiente :

coefficiente di dispersione  $D$   $\text{ps/mm/km}$  → ci dice di quanto si allunga (in ps) l'impulso in base al km di propagazione.

la dispersione potrebbe andare ad occupare il canale successivo creando interferenza intermodulistica.

- EFFETTI NON LINEARI**, in particolare il Four Wave Mixing



Se si mandano in ingresso 3 sinusoidi a frequenze diverse in un canale lineare e tempo invariante :

$$x(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t) + \sin(2\pi f_3 t)$$

$$y(t) = y_1(t) + y_2(t) + y_3(t)$$

Se il mezzo invece non è lineare ottengo in uscita la parte lineare più altri sinusoidi che sono la combinazioni di diverse frequenze tra i tre segnali mandati. Queste frequenze "spurie" sono l'effetto della non linearità. Parte di queste frequenze cadono lontane dalle frequenze del segnale quindi non danno fastidio, parte ricadono vicino o sopra.

Se contestualizziamo nel caso della fibra ottica:

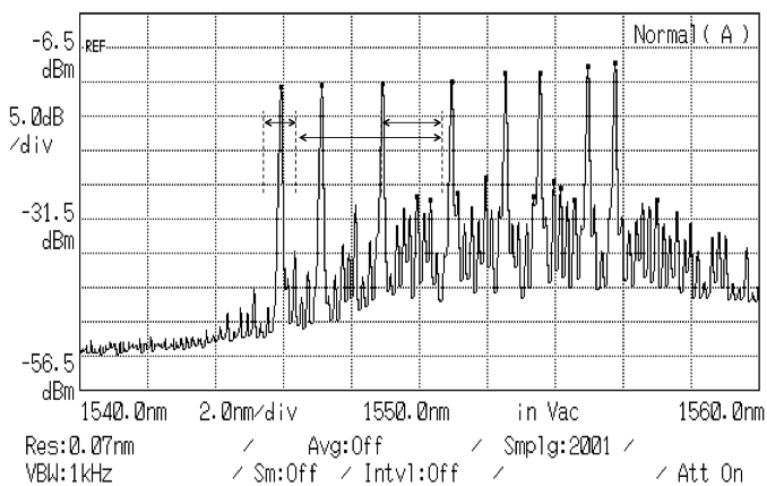
$$f_1 = 193,1 \text{ THz}$$

$$f_2 = 193,2 \text{ THz} \quad f_1 + f_2 + f_3 \rightarrow \text{e' molto lontano}$$

$$f_3 = 193,3 \text{ THz} \quad f_1 - f_2 + f_3 \rightarrow \text{in sovrapposizione} (\approx 193,2 \text{ THz})$$

Quelli che ricadono sopra al segnale vengono chiamati rumori di intermodulazione (o four wave mixing).

Questo rumore ha un effetto di deviazione del segnale, ad esempio con 8 portanti :

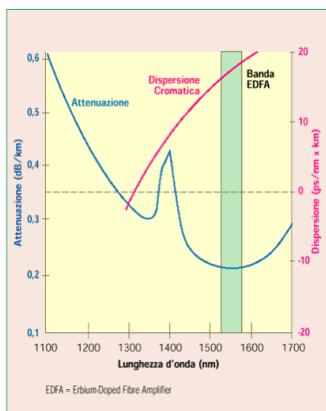


Altri difetti:

- DIAFONIA

Studiando le proprietà della fibra si può ridurre il numero di rigenerazioni sulle distanze.

### Standard fiber (G.652)

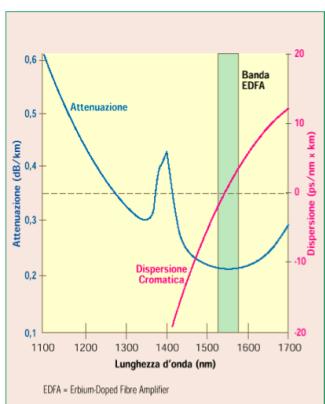


La fibra standard è caratterizzata da attenuazione minima ma dispersione elevata.

Si osserva che la dispersione varia con la frequenza, va a 0 per certe lunghezze d'onda.

Allora viene "spostata" la curva di dispersione.

### Dispersion shifted fiber (G.653)



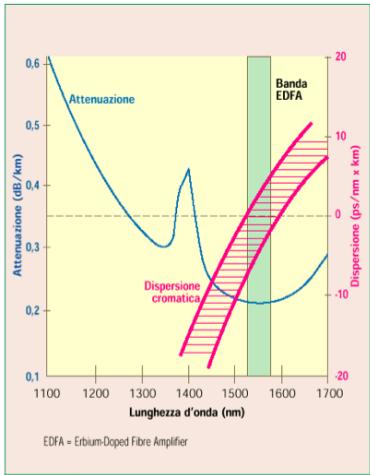
Si ottiene questo tipo di fibra dove la dispersione diventa molto piccola. Il problema che sorge con questa fibra è che gli effetti non lineari sono tanto più grandi tanto è più piccola la dispersione.

Quindi in questo caso gli effetti lineari sono massimi.

Aumentando la distanza si comincia a sovrapporre rumore (i picchi sono l'effetto lineare).

Dopo un po' di km bisogna fare rigenerazione.

## Non-zero dispersion fiber (G.655)

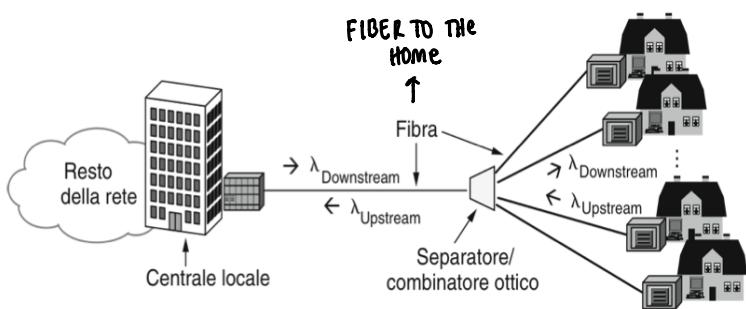


È stata creata quindi una fibra in cui si raggiunge un compromesso. Si mantiene un po' di dispersione cromatica che limitano gli effetti non lineari.

## FTTH con fibra ottica passiva

Le fibre ottiche sono talmente un buon mezzo di comunicazione che a partire dagli anni 80 sono state utilizzate per la rete a lunga distanza e successivamente hanno cominciato a sostituire il rame praticamente ovunque.

Sta prendendo piede anche in ambito locale, verso le residenze / uffici.



L'ultima parte di rete, quella che va dal nodo finale agli utenti, si chiama **rete di distribuzione**.

Mano mano questa si sta sostituendo con la fibra ottica.

## ERROR DETECTION AND CORRECTION (gestione/controllo di errori)

Abbiamo visto che un canale di comunicazione ha probabilità di errore non nulla.

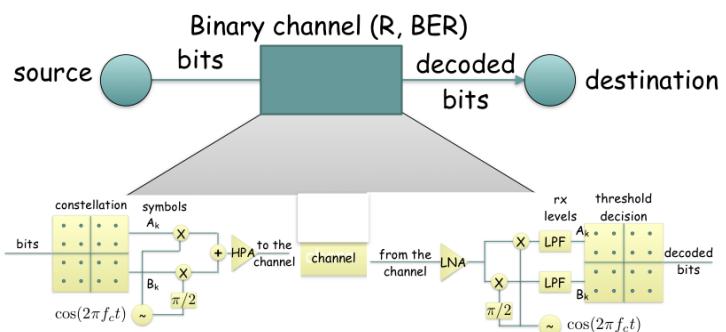


$$y(t) = G \cdot x(t - \tau) + z(t) \quad \text{AWGN} \quad y_n = h_n x_n + z_n$$

Quindi è possibile avere errori binari, il controllo di errori serve ad accorgersi di essi.

ESEMPIO : 11011001 → 11001001

Formalizzando:



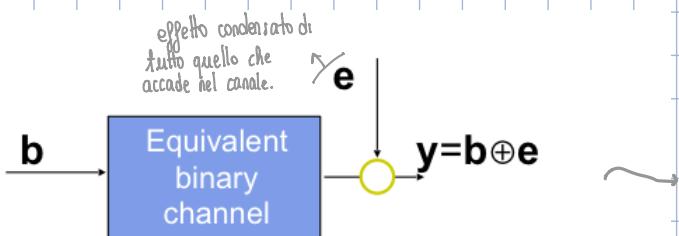
Macroscopicamente abbiamo una sorgente che emette bit, il destinatario li vuole ricevere, in mezzo c'è un canale che li trasferisce (in giallo tutto quello che succede, che viene poi riassunto con e). Tuttavia i bit ch escono non è detto che coincidano con i bit mandati, ossia errori binari.

Per risolvere gli errori si fa:

- **Error detection (rivelazione degli errori).** Consiste nel rispondere si o no alla domanda "ho ricevuto gli stessi bit in ingresso?". Poter dire che ci sono errori può servire a farsi rimandare l'informazione (ma si perde molto tempo).

Se invece non intendo richiedere trasmissione, perché magari non si possono ritrasmettere, la rivelazione degli errori serve a capire cosa serve e cosa no, quindi per eliminare errori (non li passo nel caso al livello superiore). Altro motivo per fare error detection è monitorare (solo a livello fisico) quanto frequentemente ci sono gli errori, se ce ne sono troppi dichiaro il fuori servizio.

- **Error correction (correzione degli errori).** Codifico i dati tale che correggo subito gli errori. Tuttavia non si fa spesso perché sono molti invasivi, di media si aggiungono 32 bit per correggere quindi i bit in uscita potrebbero essere il doppio di bit rispetto a quelli in entrata. Nel caso di audio e video si fa solo rivelazione di errori solo per buttare dati se ci sono errori perché non c'è tempo per fare ritrasmissione. Oppure si fa error correction.



### Equivalent binary channel model

Riassunto dell'intero canale tx-rx.

il vettore  $b$  che entra in uscita è  $y = b \oplus e$

	0	1
0	0	1
1	1	0

$x \oplus y$

EXOR, somma modulo 2

i bit che ricevo  $y = b \oplus e$  → ogni volta che c'è un 1 in  $e$  indica un errore binario

$$y_i = b_i \oplus e_i$$

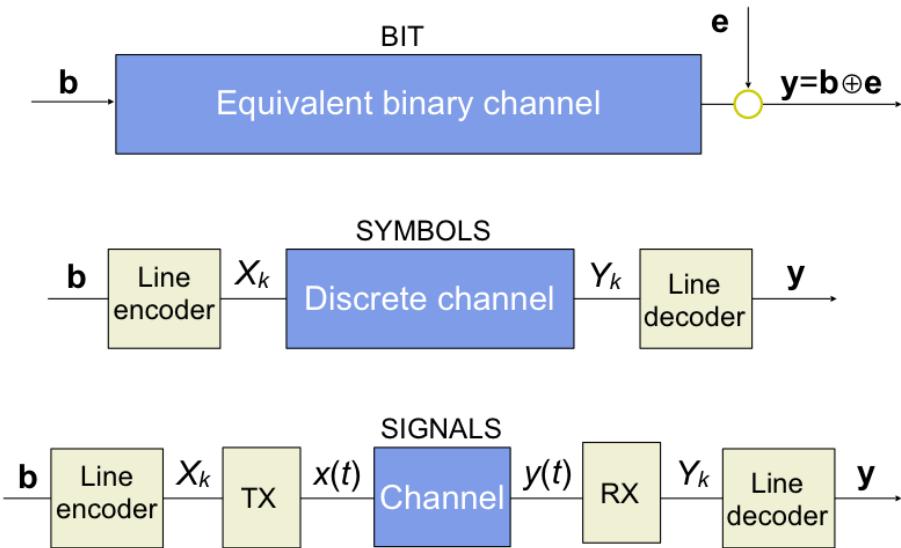
$$y_i = \begin{cases} b_i & e_i = 0 \\ \bar{b}_i & e_i = 1 \end{cases}$$

$$e_i = \begin{cases} 1 & p = 10^{-5} \\ 0 & (1-p) \end{cases} \quad BER = p$$

Gli errori binari sono indipendenti l'uno dall'altro.

16

## Channels

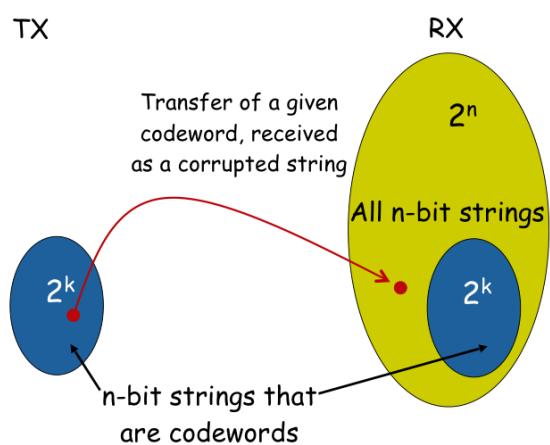


semplificazione del canale

Il principio base per fare rivelazione di errori, creando dei codici, è : vorrei trasmettere  $b$ , ma introduco prima un codificatore, un oggetto che trasforma il vettore  $b$  in un nuovo vettore  $c$  codificato (chiamato parola di codice) e quello che mando nel canale è  $c$  e non  $b$ . All'uscita del canale c'è un decodificatore che legge l'output  $y$  e dice qual è  $b$  e se ci sono errori.



## Codewords communication



Supponiamo di voler mandare un vettore lungo  $k$  bit, esistono quindi :

$2^k$  possibili vettori

Trasformo i  $k$  bit in un vettore di  $n$  bit, più grande.

Le possibili combinazioni di  $n$  bit sono  $2$  alla  $n$  di cui solamente  $2$  alla  $k$  sono parole di codice.

Esempio :

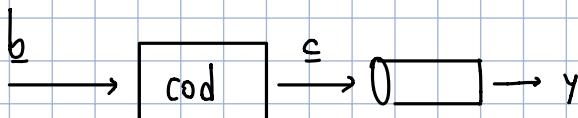
$k = 3$  Il codificatore ci dice quali sono le parole di codice (tutte diverse fra loro)

$$(2^k) \quad \underline{b} \quad | \quad \underline{c} \in (2^n) \quad |\underline{c}| = n \quad n = 5$$

000	00000
001	00110
010	01010
011	01100
100	10010
101	10100
110	11000
111	11110

$$2^k \quad 2^5 = 32$$

$$2^k \quad 2^3 = 8$$



Siccome il trasmettitore trasmette solo parole di codice, se non viene trasmessa una parola di codice allora c'è un errore.

Tuttavia anche se ricevo una parola di codice non posso dire con certezza che non ci siano errori.

DETECTION

$y \longrightarrow$  ERRORI ? SI  
NO

if  $y \in C$

then no errors  $\rightarrow$  DECOD :  $\hat{b}$

else si errori

cio' che il ricevitore pensa

$y \xrightarrow{\text{dec 1^a fase}} \underline{c} \xrightarrow{\text{dec 2^a fase}} \hat{b}$

if  $y \in C$

then  $C_{out} = y$

else  $C_{out} = \text{la più vicina a } y \quad (d_H(y, c^*) \leq d_H(y, c_j) \quad \forall j)$

mi interessa sapere quello che ricevo così che poi posso correggere

Ci deve essere quindi una regola per definire la distanza quando ricevo una parola non di codice che però voglio correggere.

## Distanza di Hemming

Si prendono due vettori binari e si paragonano, si contano quanti bit sono diversi in ogni posizione e il totale è la distanza di Hemming. In questo esempio è 9:

- $x = [0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0]$
- $y = [1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0]$

Definizione di distanza:

$A \rightarrow$  insieme  $x, y \in A$

$d(x, y) : A \times A \rightarrow \mathbb{R}^+$

1)  $d(x, x) = 0 \quad \forall x \in A$

2)  $d(x, y) = d(y, x) \quad \forall x, y \in A$

3)  $d(x, y) + d(y, z) \geq d(x, z) \quad \forall x, y, z \in A$

Il correttore prende la stringa  $y$  e va a cercare la parola di codice che ha la minima distanza di hemming dalla stringa  $y$ .

### Esempio:

Massimo di numeri rivelabili e correggibili con certezza.

Introduciamo la distanza minima delle parole di codice:

$$d_{\min} = \min \left\{ d(c_i, c_j), \forall i, j, i \neq j \right\}$$

• Se abbiamo un codice con distanza minima  $= d_{\min} \Rightarrow$  posiamo rivelare  $d_{\min} - 1$  errori

$\Rightarrow$  se gli errori in  $y \leq d_{\min} - 1$  allora gli errori sono sempre rivelabili.

• Il massimo di errori che si possono correggere sono  $\frac{(d_{\min} - 1)}{2}$  con la regola del più vicino.

Quindi  $\curvearrowleft$  se gli errori  $\leq d_{\min} - 1 \Rightarrow$  ci si accorge sempre degli errori

se gli errori  $\leq \frac{(d_{\min} - 1)}{2} \Rightarrow$  è possibile correggere sempre gli errori

Il potere di rivelazione e correzione di errore dipende quindi dalla distanza minima, più questa è grande più il codice è potente. La distanza tra parole di codice si fa aggiungendo ridondanza.

Esempio:

$$k = 3$$

$$2^3 = 8 \text{ stringhe di dati}$$

$n = 4 \rightarrow 2^4$  stringhe (16) tra le quali prendo 8 parole di codice.

$$n = 5 \rightarrow 2^5 = 32$$

$$n = 10 \rightarrow 2^{10} = 1024$$

$$2^k \xrightarrow{k} \boxed{\quad} \xrightarrow{2^k \leq} \boxed{\quad} \xrightarrow{n > k} (n - k = \text{RIDONDANZA})$$

Se  $n$  è troppo grande si trasmetterebbe troppa ridondanza.

1)  $d_{\min}$  più grande possibile

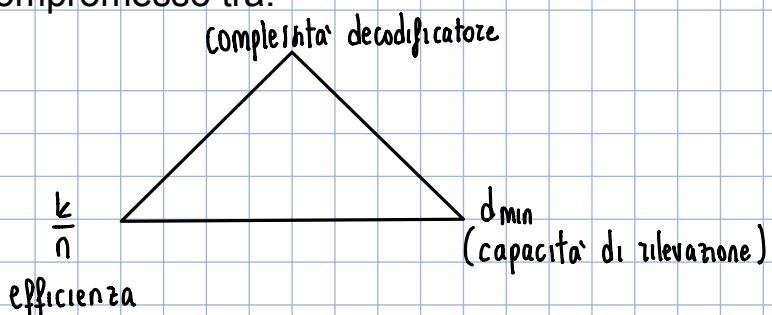
2) Posso aumentare  $d_{\min}$  aumentando  $n$  con  $k$  fisso ...

ma  $\frac{k}{n} \rightarrow 0$  per  $n \rightarrow \infty$   
e  $k$  è fisso

3) Se fisso  $\frac{k}{n} = \text{cost}$  se  $n \rightarrow \infty$  allora  $n$  può aumentare  $d_{\min}$ , codici più potenti.  
( $k \rightarrow \infty$ )

pago il prezzo nella  
complessità dell'algoritmo

Bisogna trovare un compromesso tra:



**Uno dei codici più semplici è :**

## Single Parity Check

È un particolare caso di codice in cui  $k$  è  $k$  e  $n = k+1$ , quindi vuol dire che il bit che aggiungiamo è uno solo.

Dati i bit di dati, la parola di codice è fatta così:

Info bits:  $b_1, b_2, b_3, \dots, b_k$

Check bit :  $b_{k+1} = b_1 \oplus b_2 \oplus b_3 \oplus \dots \oplus b_k$  [e' la somma di tutti i bit]

Quanti sono i bit uguali ad 1 nella parola di codice? È uguale al numero di bit che avevamo nei dati, se sono dispari quando poi andiamo a calcolare  $b(k+1)$  facendo la somma viene 1 e quindi poi la parola di codice viene pari. Se il numero di bit pari a 1 nei dati viene pari, la somma viene 0 e quindi il numero di bit nella parola di codice rimane pari.

Proprietà caratteristica di questo codice è che la parola di codice è sempre pari (parity check).

Esempio :

$$k = 7 \quad n = 8$$

**1001100**  **10011001**

$$b_8 = 1$$

1100110 , 11001100

$$b_0 = 0$$

Quando c'è questo codice allora la regola diventa:

$y \in C \leftrightarrow$  in  $y$  bit<sub>1</sub> in  $y$  e' pari  $\Rightarrow$   $y$  e' una parola di codice.