Appunti di Fondamenti di Comunicazioni e Internet

Canale 2 · Reti · Prof. Marco Polverini

Capitolo 5: Strato di collegamento

Autore: Tommaso C.

Appunti tratti dalle lezioni in aula e dal materiale del corso

---

Argomenti: Servizi dello strato di collegamento · Framing: flag, bit stuffing e byte stuffing
Duplexing · Errori binari e parole di codice · Parità singola
Distanza di Hamming e codice di Hamming (7,4) · Parità bidimensionale · Internet checksum · CRC: polinomi, generatore e parola di codice

Indice

1 Servizi dello strato di collegamento **1**

2 Framing **1**

2.1 Bit stuffing 2

2.2 Byte stuffing 3

3 Duplexing **3**

4 Errori binari e parole di codice **4**

4.1 Parole di codice 5

5 Parità singola **6**

5.1 Probabilità di un errore non rivelato 7

6 Distanza di Hamming **9**

6.1 Codice di Hamming (7,4) 11

7 Parità bidimensionale **12**

8 Internet checksum **14**

9 CRC: codici a ridondanza ciclica **15**

10 Esercizi **18**

11 Soluzioni **21**

1 Servizi dello strato di collegamento

Definizione – Servizi dello strato di collegamento

Lo strato di collegamento (2) offre allo strato di rete fino a quattro servizi:

1. il **framing**, cioè la creazione e la delimitazione del frame;
2. il **duplexing**, cioè l'uso del collegamento nelle due direzioni;
3. la **rivelazione e correzione degli errori** binari;
4. la **consegna affidabile**, che rimedia a perdite, errori binari, duplicati e frame fuori sequenza.

Uno strato di collegamento *può* offrire questi servizi, ma non è detto che li offra tutti: dipende dalla tecnologia. Ethernet, la più diffusa, rivela gli errori ma non li corregge e non offre la consegna affidabile.

La rivelazione degli errori e la consegna affidabile sono servizi distinti. La prima dice se un frame contiene errori binari; la seconda garantisce che i frame arrivino tutti, corretti, in ordine e una volta sola.

Lo strato di collegamento è realizzato in parte nella **scheda di rete** (NIC), insieme allo strato fisico, e in parte nel software di controllo del sistema operativo, eseguito dalla CPU. È quindi la frontiera fra hardware e software.

2 Framing

Definizione – Framing e flag

- Il **framing** crea il frame dal pacchetto che lo strato di rete consegna: aggiunge un'intestazione H_2 in testa e una coda (*trailer*) in fondo.
- Il **flag** è una sequenza fissa di bit che delimita il frame, in modo che il ricevitore riconosca dove comincia.



Il ricevitore vede solo una sequenza di bit, tutti uguali in apparenza. La lunghezza del frame, un campo presente in quasi ogni intestazione, dice dove finisce il frame solo se si sa già dove comincia: per questo serve il flag.

Definizione – Flag di esempio

Un flag possibile è il byte

$$\boxed{01111110}$$

cioè uno 0, sei 1 e uno 0. Funziona solo se la stessa sequenza non compare in nessun altro punto del frame, intestazione compresa.

Il pacchetto può contenere per caso la sequenza del flag, e allora il ricevitore sbaglia i confini del frame. Lo **stuffing** (riempimento) inserisce nel frame elementi artificiali che rompono queste sequenze; il ricevitore li toglie.

2.1 Bit stuffing

Proposizione – Bit stuffing e destuffing

- In trasmissione, dopo ogni sequenza di cinque 1 consecutivi del frame si inserisce uno 0, qualunque sia il bit successivo. Il flag non si modifica.
- In ricezione si contano gli 1 consecutivi. Dopo cinque 1 si guarda il bit successivo: se è 0 è un bit di stuffing e si toglie (**destuffing**); se è 1 la sequenza è un flag.

Dopo lo stuffing il frame non contiene mai sei 1 consecutivi fuori dai flag, quindi il flag non si può imitare.

Esempio – Bit stuffing di 32 bit

Nella sequenza originale

$$10111111111110110111110011111100$$

ci sono undici 1 consecutivi dopo i primi due bit, poi cinque 1 e poi sei 1. Lo stuffing inserisce quattro 0, evidenziati, e la sequenza trasmessa ha 36 bit:

$$1011111\underline{\mathbf{0}}11111\underline{\mathbf{0}}10110111111\underline{\mathbf{0}}00111111\underline{\mathbf{0}}100$$

Il ricevitore, dopo ogni gruppo di cinque 1, trova uno 0, lo toglie e riottiene la sequenza originale.

L'Esercizio 3 applica stuffing e destuffing a due sequenze più lunghe.

2.2 Byte stuffing

Il **byte stuffing** lavora a byte interi. Oltre al flag usa un secondo byte riservato, l'**escape**, che differisce dal flag negli ultimi due bit:

$$\text{flag} = 01111110, \qquad \text{escape} = 01111101.$$

Proposizione – Byte stuffing e destuffing

- In trasmissione, davanti a ogni byte del frame uguale al flag o all'escape si inserisce un escape.
- In ricezione, un escape si toglie e il byte che lo segue è un dato, anche se è uguale al flag o all'escape. Un flag non preceduto da un escape delimita il frame.

Esempio – Byte stuffing di tre byte

I dati sono i tre byte 01111110, 00110101, 01111101: il primo è uguale al flag, il terzo all'escape. Lo stuffing inserisce un escape davanti a ciascuno dei due:

01111101 01111110 00110101 01111101 01111101

Il ricevitore toglie il primo escape e tiene il flag che lo segue come dato; poi toglie il secondo escape e tiene l'escape che lo segue.

3 Duplexing

Definizione – Half duplex e full duplex

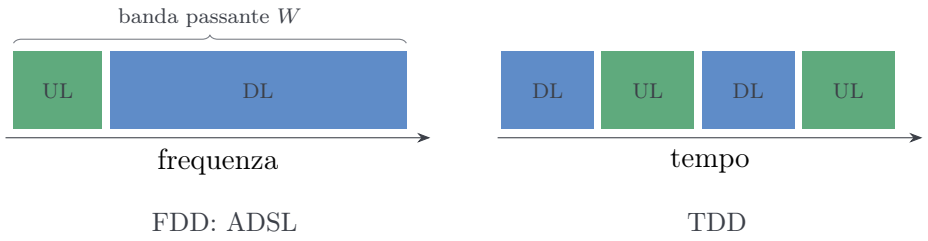
Un collegamento porta dati in **uplink** (UL), dall'utente verso la rete, e in **downlink** (DL), dalla rete verso l'utente.

- Un collegamento **half duplex** trasmette nelle due direzioni, ma una per volta.
- Un collegamento **full duplex** trasmette nelle due direzioni nello stesso momento.

Il **duplexing** è il servizio che separa le due direzioni sullo stesso collegamento.

Duplexing e multiplazione (Capitolo 2) usano le stesse tecniche, divisione nello spazio, in frequenza o nel tempo, ma con uno scopo diverso: la multiplazione divide una risorsa fra utenti, il duplexing fra le due direzioni.

Duplexing	Come separa le due direzioni	Esempio
spaziale	un mezzo per ogni direzione, per esempio una coppia di fili	cavo Ethernet, full duplex
in frequenza (**FDD**)	lo stesso mezzo, sempre attivo, con la banda passante divisa in due porzioni	DSL
nel tempo (**TDD**)	lo stesso mezzo, con intervalli di tempo alternati per le due direzioni	reti ottiche passive (PON)



Nell'ADSL (*Asymmetric* DSL) la ripartizione della banda è asimmetrica: il downlink ha più banda, e quindi più capacità, perché gli utenti scaricano più di quanto caricano.

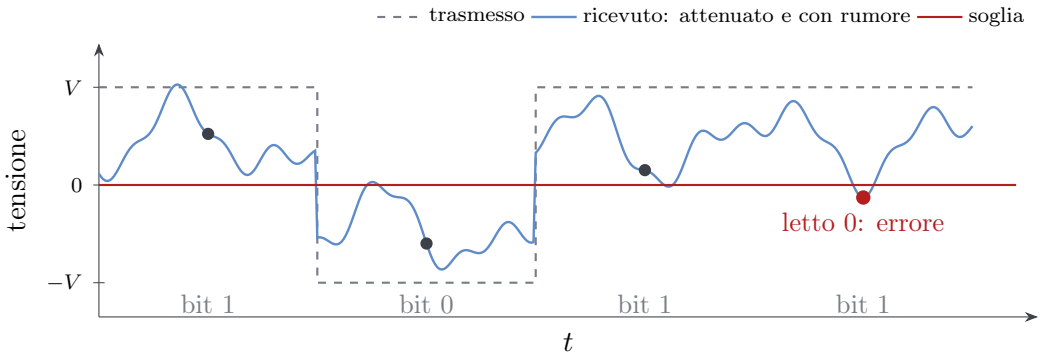
4 Errori binari e parole di codice

Definizione – Errore binario e BER

- Un **errore binario** è un bit ricevuto con il valore sbagliato: è stato trasmesso un 1 e il ricevitore legge 0, o viceversa.
- Il **BER** (*bit error rate*) p è la probabilità che un bit sia ricevuto errato.

Gli errori binari nascono da attenuazione e rumore. In un modello semplice un 1 è una tensione positiva e uno 0 una negativa, e il ricevitore decide con una **soglia** a zero.

1. Lungo il mezzo il segnale perde potenza (**attenuazione**).
2. Al segnale attenuato si somma il **rumore**, un disturbo casuale che non si può eliminare.
3. Se la somma passa dall'altra parte della soglia, il ricevitore legge il bit sbagliato.



Il **rapporto segnale-rumore** è l'indicatore principale della probabilità di errore binario. L'attenuazione riduce il segnale ma non il rumore, quindi abbassa il rapporto e rende più probabili gli errori.

Definizione – Vettore d'errore

Se si trasmette la stringa C di n bit e si riceve C', il **vettore d'errore** E è la stringa di n bit con $e_i = 1$ se il bit i è stato ricevuto errato ed $e_i = 0$ se è corretto. Vale

$$\boxed{C' = C \oplus E}$$

dove $\oplus$ è la somma bit per bit modulo 2: $0 \oplus 0 = 1 \oplus 1 = 0$ e $0 \oplus 1 = 1 \oplus 0 = 1$.

Per esempio, se si trasmette 1010 e si riceve 1110, il secondo bit è errato e $E = 0100$.

4.1 Parole di codice

Definizione – Parole di codice

Un **codice** ammette solo alcune stringhe di n bit, le **parole di codice** (*codeword*), come un vocabolario ammette solo alcune parole. Il trasmettitore invia solo parole di codice, e il ricevitore controlla se ciò che riceve è una parola di codice.

Proposizione – Che cosa conclude il ricevitore

- Se la stringa ricevuta **non** è una parola di codice, c'è sicuramente un errore.
- Se è una parola di codice, non si può concludere che sia corretta: gli errori possono trasformare una parola di codice in un'altra. È un **errore non rivelato**.

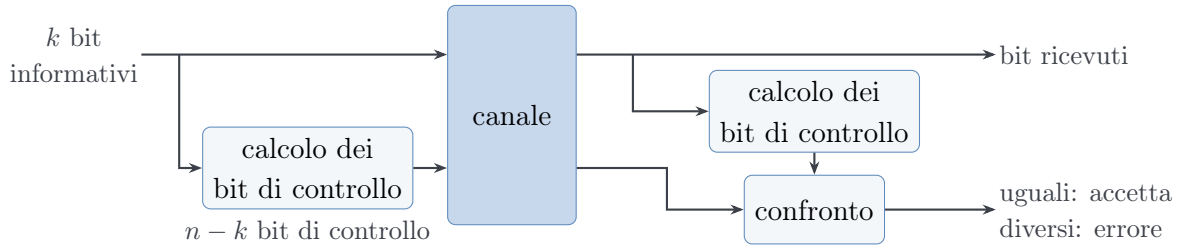
Definizione – Codificatore e overhead

Il **codificatore** calcola dai k **bit informativi** $n-k$ **bit di controllo** e forma la parola di codice di n bit. I bit di controllo non portano informazione; la loro quota è l'**overhead**:

$$\boxed{\text{overhead} = \frac{n-k}{n}}$$

La rivelazione funziona così.

1. Il trasmettitore calcola i bit di controllo dai bit informativi e invia entrambi.
2. Il canale può introdurre errori in qualunque bit, informativo o di controllo.
3. Il ricevitore ricalcola i bit di controllo dai bit informativi ricevuti e li confronta con i bit di controllo ricevuti: se coincidono accetta il frame, altrimenti rivela un errore.



Esempio – Codice a ripetizione

Ogni bit si trasmette dieci volte: 1 diventa 1111111111. Se una stringa ricevuta di dieci bit non è tutta di 1 o tutta di 0, c'è un errore. Il ricevitore può anche *correggerlo* scegliendo il valore della maggioranza, che però è solo il valore più probabile. L'overhead è 9/10.

Un buon codice protegge il messaggio il più possibile con meno bit di controllo possibile, perché i bit di controllo sono overhead.

5 Parità singola

Definizione – Codice a parità singola

Ai k bit informativi $b_1, \ldots, b_k$ si aggiunge un solo bit di controllo, il **bit di parità**, scelto in modo che la parola abbia un numero pari di 1:

$$\boxed{b_{k+1} = (b_1 + b_2 + \cdots + b_k) \bmod 2}$$

Quindi $n = k+1$ e l'overhead è $\dfrac{1}{k+1}$.

Il ricevitore conta gli 1 della parola ricevuta: se sono dispari, rivela un errore.

Esempio – Parità di 7 bit

I bit informativi 0101100 contengono tre 1, quindi $b_8 = 1$ e la parola trasmessa è 0101100 1, con quattro 1.

- Con un errore nel bit 3 si riceve 01**1**1100 1: cinque 1, dispari. L'errore è rivelato.
- Con errori nei bit 3 e 5 si riceve 01**1**1**0**00 1: quattro 1, pari. È una parola di codice, e l'errore non è rivelato.

Proposizione – Che cosa rivela la parità singola

La parità singola rivela ogni configurazione con un numero **dispari** di errori e non rivela nessuna configurazione con un numero **pari** di errori.

Dimostrazione

1. La parola trasmessa ha un numero pari di 1.
2. Ogni bit errato cambia di uno il numero di 1, quindi ne cambia la parità.
3. Dopo un numero dispari di errori il numero di 1 è dispari, e il ricevitore rivela l'errore. Dopo un numero pari torna pari, e la stringa ricevuta è una parola di codice.

□

Per esempio, se si trasmette 1011, la parola è 1011 1. Con $E = 00010$ si riceve 10101: tre 1, e l'errore è rivelato.

5.1 Probabilità di un errore non rivelato

Delle 2^n configurazioni d'errore possibili, metà ha un numero pari di 1: circa metà delle configurazioni d'errore non è rivelata. Le configurazioni però non sono ugualmente probabili.

Proposizione – Probabilità di una configurazione d'errore

Se i bit sono errati in modo **indipendente**, ciascuno con probabilità p, una configurazione precisa con i bit errati su n ha probabilità

$$\boxed{p^i(1-p)^{n-i} = (1-p)^n \left(\frac{p}{1-p}\right)^i}$$

Dimostrazione

1. Gli i bit errati hanno probabilità p^i, gli $n-i$ corretti $(1-p)^{n-i}$, e per l'indipendenza le probabilità si moltiplicano.
2. Moltiplicando e dividendo per $(1-p)^i$ si ottiene la seconda forma.

□

Il BER è piccolo, di solito fra 10^{-6} e 10^{-4}, quindi $p/(1-p) < 1$ e la probabilità diminuisce al crescere di i: le configurazioni con pochi errori sono le più probabili. Nella realtà gli errori non sono indipendenti, ma arrivano a gruppi (**burst**), perché un disturbo del canale dura per più bit.

Proposizione – Errore non rivelato con la parità singola

Con errori indipendenti, la probabilità che la parità singola non riveli un errore è la probabilità di un numero pari, e non nullo, di bit errati:

$$P_{\text{nr}} = \sum_{\substack{i=2,4,6,\dots \\ i \le n}} \binom{n}{i} p^i (1-p)^{n-i}$$

Il coefficiente binomiale $\binom{n}{i}$ conta le configurazioni con i errori, cioè i modi di scegliere le i posizioni errate.

Esempio – Parole di 32 bit

Con $n = 32$ e $p = 10^{-3}$ il primo termine, $i = 2$, domina:

$$P_{\text{nr}} \approx \binom{32}{2} (10^{-3})^2 (1-10^{-3})^{30} = 496 \cdot 10^{-6} \cdot 0{,}970 \approx 4{,}8 \times 10^{-4}.$$

Il termine con $i = 4$ vale circa $3{,}5 \times 10^{-8}$. Su un milione di parole al secondo, circa 480 al secondo arriverebbero errate senza che il ricevitore se ne accorga.

La parità singola ha due debolezze: usa un solo bit di controllo, e il modo in cui costruisce le parole di codice non le tiene lontane fra loro. Servono più bit di controllo, scelti bene.

6 Distanza di Hamming

Definizione – Distanza di Hamming

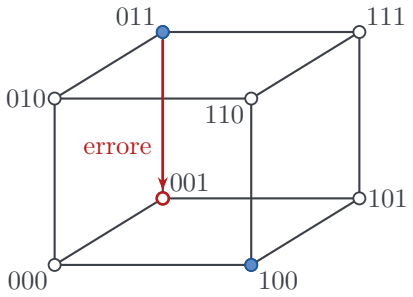
- La **distanza di Hamming** fra due stringhe di bit della stessa lunghezza è il numero di posizioni in cui differiscono.
- La **distanza** d di un codice è la minima distanza di Hamming fra due parole di codice diverse.

Per esempio 10110 e 11011 differiscono nelle posizioni 2, 3 e 5, quindi hanno distanza 3.

Gli errori più probabili cambiano pochi bit, cioè spostano la parola trasmessa in una stringa vicina. Se le parole di codice sono vicine fra loro, pochi errori portano su un'altra parola di codice e non vengono rivelati. Un buon codice **massimizza la distanza** fra le parole di codice.

Se tutte le 2^n stringhe fossero parole di codice, cioè senza bit di controllo, un solo errore porterebbe su un'altra parola di codice. Con $n = 4$ e due bit di controllo, invece, le parole di codice sono $2^2 = 4$ delle 16 stringhe, e conviene sceglierle il più lontano possibile.

Le stringhe di 3 bit sono i vertici di un cubo, e ogni spigolo cambia un solo bit. La distanza fra due stringhe è il minimo numero di spigoli che le separa.



Esempio – Correzione nel cubo

Il codice ha le due parole 011 e 100, a distanza 3. Si trasmette 011 e un errore sul secondo bit porta a 001.

- 001 non è una parola di codice: l'errore è rivelato.
- 001 dista 1 da 011 e 2 da 100: scegliendo la parola di codice più vicina, il ricevitore corregge l'errore.

Teorema – Rivelazione e correzione con distanza d

Un codice di distanza d:

- usato solo per rivelare, rivela ogni configurazione fino a $d-1$ errori;
- usato per correggere, con la parola di codice più vicina, corregge ogni configurazione fino a

$$\boxed{t = \left\lfloor \frac{d-1}{2} \right\rfloor}$$

  errori, cioè $\lfloor d/2 \rfloor$ errori se d è dispari e $d/2 - 1$ se d è pari;
- con d pari, rivela anche le configurazioni di $d/2$ errori, senza poterle correggere.

Dimostrazione

1. Due parole di codice distano almeno d. Con al più $d-1$ errori la stringa ricevuta non può essere un'altra parola di codice, quindi l'errore è rivelato.
2. Con al più t errori la stringa ricevuta dista al più t dalla parola trasmessa e almeno $d - t > t$ da ogni altra, perché $2t < d$. La parola più vicina è quella trasmessa.
3. Con d pari e $d/2$ errori la stringa ricevuta può distare $d/2$ da due parole di codice: non è una parola di codice, ma non si sa quale scegliere.

□

Esempio – Distanza 5

Il codice ha le due parole 00000 e 11111, quindi $d = 5$ e $t = 2$. Si trasmette 00000.

Errori	Ricevuta	Più vicina	Esito
1	10000	00000	rivelato e corretto
2	11000	00000	rivelato e corretto
3	11100	11111	rivelato, ma corretto in modo sbagliato

Usato solo per rivelare, lo stesso codice rivela fino a $d - 1 = 4$ errori.

6.1 Codice di Hamming (7,4)

Definizione – Codice di Hamming (7,4)

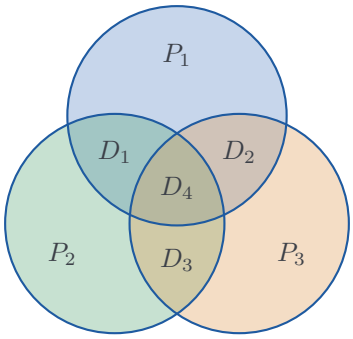
Il **codice di Hamming (7,4)** aggiunge a 4 bit informativi D_1, D_2, D_3, D_4 tre bit di parità P_1, P_2, P_3. La parola di codice ha 7 bit, nell'ordine

$$\boxed{P_1\ P_2\ D_1\ P_3\ D_2\ D_3\ D_4}$$

e ogni bit di parità rende pari il numero di 1 del suo gruppo:

- P_1 con D_1, D_2, D_4;
- P_2 con D_1, D_3, D_4;
- P_3 con D_2, D_3, D_4.

Il codice ha distanza $d = 3$: corregge un errore, oppure, usato solo per rivelare, rivela fino a due errori. Trasmettitore e ricevitore si accordano sullo schema; il codice di Hamming lo fissa in modo da massimizzare la distanza.



Esempio – Codifica di 1010

Con $D_1D_2D_3D_4 = 1010$:

- $D_1, D_2, D_4 = 1, 0, 0$ hanno un 1, quindi $P_1 = 1$;
- $D_1, D_3, D_4 = 1, 1, 0$ hanno due 1, quindi $P_2 = 0$;
- $D_2, D_3, D_4 = 0, 1, 0$ hanno un 1, quindi $P_3 = 1$.

La parola di codice è $P_1P_2D_1P_3D_2D_3D_4 = 1011010$.

Ogni bit informativo sta in un gruppo diverso di controlli, quindi i controlli che falliscono dicono quale bit è errato.

Proposizione – Correzione di un errore singolo

Con un solo bit errato falliscono esattamente i controlli dei gruppi che lo contengono:

Falliscono	Bit errato	Falliscono	Bit errato
solo P_1	P_1	P_1 e P_2	D_1
solo P_2	P_2	P_1 e P_3	D_2
solo P_3	P_3	P_2 e P_3	D_3
		tutti e tre	D_4

Il ricevitore inverte quel bit.

Esempio – Errore su D_2

Si trasmette 1011010 e il canale inverte D_2: si riceve 1011**1**10.

- Gruppo di P_1: $P_1, D_1, D_2, D_4 = 1, 1, 1, 0$, tre 1: fallisce.
- Gruppo di P_2: $P_2, D_1, D_3, D_4 = 0, 1, 1, 0$, due 1: torna.
- Gruppo di P_3: $P_3, D_2, D_3, D_4 = 1, 1, 1, 0$, tre 1: fallisce.

Falliscono P_1 e P_3, quindi il bit errato è D_2. Il ricevitore lo inverte e riottiene 1011010.

Con due errori il ricevitore vede che la stringa non è una parola di codice, ma se prova a correggerla sbaglia: si vede nell'Esercizio 1.

7 Parità bidimensionale

Definizione – Parità bidimensionale

Nella **parità bidimensionale** (o a blocchi) il messaggio si divide in righe della stessa lunghezza, disposte una sotto l'altra in una matrice.

1. Ogni riga riceve un bit di parità, che forma una colonna in più.
2. Ogni colonna riceve un bit di parità, che forma una riga in più.
3. Il bit d'angolo è la parità della riga di parità, uguale a quella della colonna di parità.

Esempio – Matrice di 20 bit

Il messaggio 10010 01000 10010 11011 diventa quattro righe di 5 bit:

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

La prima riga ha due 1, quindi il suo bit è 0; la prima colonna ha tre 1, quindi il suo bit è 1. I bit di controllo sono 10 su 30, e l'overhead è 1/3.

Il ricevitore ricostruisce la matrice e ricontrolla la parità di ogni riga e di ogni colonna. Con un solo bit errato fallisce una riga e fallisce una colonna: il bit errato è all'incrocio, e il ricevitore lo inverte.

Esempio – Un errore

Nella matrice dell'esempio precedente il bit in riga 2 e colonna 2 passa da 1 a 0.

1	0	0	1	0	0	
0	**0**	0	0	0	1	← dispari
1	0	0	1	0	0	
1	1	0	1	1	0	
1	0	0	1	1	1	
	↑					
	dispari					

Falliscono la riga 2 e la colonna 2, e il bit all'incrocio si corregge.

Proposizione – Che cosa rivela e corregge la parità bidimensionale

- Corregge ogni errore singolo.
- Rivela ogni configurazione di 2 errori e ogni configurazione con un numero dispari di errori, ma non le corregge.
- Non rivela alcune configurazioni di 4 errori, per esempio quattro bit ai vertici di un rettangolo.

Dimostrazione

1. Un numero dispari di errori ne mette un numero dispari in almeno una riga, che quindi fallisce.
2. Due errori in righe diverse fanno fallire quelle due righe; nella stessa riga stanno in colonne diverse, che falliscono.
3. Quattro errori ai vertici di un rettangolo ne mettono due in ogni riga e in ogni colonna coinvolta: tutte le parità tornano.

□

Tre configurazioni, con 2, 3 e 4 errori, sono nell'Esercizio 2.

8 Internet checksum

Definizione – Internet checksum

Il messaggio, lungo un multiplo di w bit, si divide in L blocchi di w bit, $b_0, b_1, \dots, b_{L-1}$, letti come numeri interi. Il **checksum** b_L è un blocco di w bit tale che

$$\boxed{b_L = -\left(b_0 + b_1 + \cdots + b_{L-1}\right) \bmod \left(2^w - 1\right)}$$

Nel protocollo IP $w = 16$.

Per costruzione la parola di codice, cioè i blocchi più il checksum, soddisfa

$$b_0 + b_1 + \cdots + b_{L-1} + b_L \equiv 0 \pmod{2^w - 1}.$$

Il ricevitore divide in blocchi ciò che riceve, checksum compreso, e li somma: se ottiene 0 accetta, altrimenti rivela un errore.

Nota – Aritmetica modulo $2^w - 1$

Modulo $2^w - 1$ si tolgono multipli di $2^w - 1$ finché il risultato sta fra 0 e $2^w - 2$. L'opposto $-x$ è il numero che sommato a x dà $2^w - 1$, cioè 0: con $w = 4$ si lavora modulo 15 e $-7 \equiv 8$, come sull'orologio.

Esempio – Checksum di 4 bit

Con $w = 4$ si lavora modulo $2^4 - 1 = 15$. I blocchi sono $b_0 = 1100 = 12$ e $b_1 = 1010 = 10$, quindi $L = 2$.

1. La somma è $12 + 10 = 22 \equiv 7 \pmod{15}$.
2. Il checksum è $b_L = b_2 \equiv -7 \equiv 8 \pmod{15}$, cioè $b_2 = 1000$.
3. Verifica: $12 + 10 + 8 = 30 \equiv 0 \pmod{15}$.

Il checksum è semplice da calcolare in software, e Internet lo usa nello strato di rete e in quello di trasporto. Si applica negli Esercizi 4, 5, 6 e 7.

9 CRC: codici a ridondanza ciclica

Definizione – Polinomio di una stringa

Una stringa di k bit si rappresenta con un **polinomio** di grado al più $k - 1$: i bit sono i coefficienti delle potenze di x, dalla più alta fino a x^0,

$$b_{k-1}\, b_{k-2} \ldots b_1\, b_0 \;\longleftrightarrow\; P(x) = b_{k-1}x^{k-1} + \cdots + b_1 x + b_0.$$

I coefficienti valgono 0 o 1 e si calcolano modulo 2.

Per esempio 1010 è $P(x) = x^3 + x$.

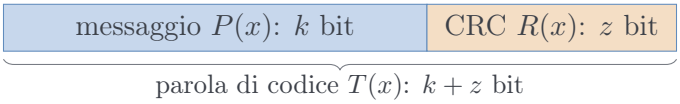
Nota – Aritmetica modulo 2

Calcolare modulo m significa tenere il resto della divisione per m: $12 \bmod 5 = 2$. Modulo 2 i coefficienti restano 0 e 1 e $1 + 1 = 0$, quindi **somma e differenza coincidono**: sono lo XOR bit a bit, e si scrivono sempre con $+$. Moltiplicare una stringa per x^z la sposta di z posizioni a sinistra, cioè aggiunge z zeri in coda.

Definizione – Polinomio generatore e CRC

Trasmettitore e ricevitore si accordano su un **polinomio generatore** $G(x)$ di grado z, cioè una stringa di $z+1$ bit che comincia e finisce con 1. Per proteggere il messaggio $P(x)$ il trasmettitore:

1. aggiunge z zeri in coda al messaggio: $x^z P(x)$;
2. divide $x^z P(x)$ per $G(x)$, ottenendo il quoziente $Q(x)$ e il resto $R(x)$:
$$x^z P(x) = Q(x)\,G(x) + R(x), \qquad \deg R < z;$$
3. trasmette la parola di codice
$$\boxed{T(x) = x^z P(x) + R(x)}$$
cioè i bit del messaggio seguiti dai z bit di $R(x)$, il **CRC**.



Proposizione – La parola di codice e il controllo del ricevitore

- $T(x)$ è divisibile per $G(x)$, con resto 0: $T(x) = Q(x)\,G(x)$.
- Il ricevitore divide per $G(x)$ la stringa ricevuta. Con resto diverso da 0 la stringa non è una parola di codice e rivela un errore. Con resto 0 è una parola di codice e il ricevitore accetta.

Dimostrazione

1. Da $x^z P(x) = Q(x)\,G(x) + R(x)$ segue $T(x) = x^z P(x) + R(x) = Q(x)\,G(x) + R(x) + R(x)$.
2. Modulo 2 somma e differenza coincidono, quindi $R(x) + R(x) = 0$.
3. Resta $T(x) = Q(x)\,G(x)$, un multiplo di $G(x)$.

□

Il canale somma alla parola trasmessa l'errore $E(x)$ (Sezione 4) e il ricevitore riceve $T(x) + E(x)$. Poiché $T(x)$ è un multiplo di $G(x)$, il resto di $T(x) + E(x)$ è il resto di $E(x)$: l'errore sfugge solo se $E(x)$ è un multiplo non nullo di $G(x)$, perché allora la stringa ricevuta è ancora una parola di codice. Si applica nell'Esercizio 8; l'Esercizio 4 calcola il CRC di un messaggio di 16 bit lavorando solo sulle stringhe dei coefficienti.

Esempio – Divisione fra polinomi

Con $G(x) = x^3 + x + 1$ si divide $x^6 + x^5$. A ogni passo si moltiplica $G(x)$ per la potenza di x che cancella il termine di grado più alto, e si somma:

1. $x^3\,G(x) = x^6 + x^4 + x^3$; la somma con $x^6 + x^5$ dà $x^5 + x^4 + x^3$;
2. $x^2\,G(x) = x^5 + x^3 + x^2$; la somma dà $x^4 + x^2$;
3. $x\,G(x) = x^4 + x^2 + x$; la somma dà x.

Il resto x ha grado $1 < 3$: ci si ferma. Il quoziente è $x^3 + x^2 + x$, il resto è $R(x) = x$.

Esempio – Parola di codice del messaggio 1100

Messaggio 1100, cioè $P(x) = x^3 + x^2$, e $G(x) = x^3 + x + 1$ (1011), con $z = 3$.

1. $x^3 P(x) = x^6 + x^5$, cioè 1100 000: il messaggio spostato di tre posizioni.
2. Il resto di $x^6 + x^5$ per $G(x)$ è x, come nell'esempio precedente. Su tre bit è 010.
3. La parola di codice è $T(x) = x^6 + x^5 + x$, cioè $\boxed{1100\,010}$.

Verifica: $T(x) = (x^3 + x^2 + x)\,G(x)$. Il CRC aggiunge $z = 3$ bit ai $k = 4$ del messaggio: l'overhead è 3/7.

Il CRC si calcola con un circuito a registri a scorrimento, che esegue la divisione mentre i bit passano: il calcolo avviene durante la trasmissione e deve durare nanosecondi, troppo poco per il software. I generatori usati in pratica hanno grado alto: il CRC-16 ha $G(x) = x^{16} + x^{12} + x^5 + 1$, quindi $z = 16$ bit di controllo, e il CRC-32 ha grado 32.

Riepilogo

- **Servizi**: framing, duplexing, rivelazione e correzione degli errori, consegna affidabile; una tecnologia può offrirne solo alcuni.
- **Framing**: il flag 01111110 delimita il frame; il bit stuffing aggiunge uno 0 dopo cinque 1, il byte stuffing un escape davanti a flag ed escape.
- **Duplexing**: half duplex alterna le direzioni, full duplex le usa insieme; separazione spaziale, FDD o TDD.
- **Parole di codice**: $n-k$ bit di controllo su n, overhead $(n-k)/n$; $C' = C \oplus E$.
- **Parità singola**: rivela i numeri dispari di errori; P_{nr} è la somma sui numeri pari $i \geq 2$ di $\binom{n}{i} p^i (1-p)^{n-i}$.
- **Distanza e Hamming (7,4)**: un codice rivela fino a $d-1$ errori e ne corregge fino a $\lfloor (d-1)/2 \rfloor$; Hamming ha $d = 3$ e i controlli falliti indicano il bit errato.
- **Bidimensionale e checksum**: la prima corregge un errore singolo; il secondo rende la somma dei blocchi 0 modulo $2^w - 1$.
- **CRC**: $T(x) = x^z P(x) + R(x)$, con $R(x)$ resto di $x^z P(x)$ per $G(x)$; $T(x)$ è un multiplo di $G(x)$ e il ricevitore accetta se il resto è 0.

10 Esercizi

Esercizio 1 – Codice di Hamming (7,4) con due errori

Si trasmette la parola di codice di Hamming (7,4) del dato 1010, cioè 1011010. Il canale inverte P_2 e D_2, e si riceve 1111110.

a. Quali controlli di parità falliscono?

b. Quale bit inverte un ricevitore che corregge un errore, e che cosa ottiene?

c. Perché la correzione sbaglia?

Soluzione a pagina 21 →

Esercizio 2 – Parità bidimensionale con 2, 3 e 4 errori

Si trasmette la matrice

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

e si ricevono, in tre casi, le matrici seguenti (bit errati evidenziati).

(a)

1	0	0	1	0	0
0	**0**	0	0	0	1
1	0	0	1	0	0
1	**0**	0	1	1	0
1	0	0	1	1	1

(b)

1	0	0	1	0	0
0	**0**	0	**1**	0	1
1	0	0	1	0	0
1	**0**	0	1	1	0
1	0	0	1	1	1

(c)

1	0	0	1	0	0
0	**0**	0	**1**	0	1
1	0	0	1	0	0
1	**0**	0	**0**	1	0
1	0	0	1	1	1

Per ogni caso, dire quali righe e colonne falliscono, se l'errore è rivelato e se correggendo il bit all'incrocio si riottiene la matrice trasmessa.

Soluzione a pagina 21 →

Esercizio 3 – Bit stuffing e destuffing

1. Applicare la tecnica di riempimento dei protocolli orientati ai bit (bit stuffing) alla sequenza

   $$01110111111001110111010111111110111101111111.$$

2. Si supponga che venga ricevuta la sequenza

   $$01111110111011111011111011111011110111011000111110101000011111110.$$

   Si cancellino i bit addizionali e si ricostruisca il frame originale.

Soluzione a pagina 22 →

Esercizio 4 – Checksum e CRC dello stesso messaggio

Per la stringa $M = 1011000101101010$ calcolare:

1. il valore dell'Internet checksum a 8 bit;
2. il valore del CRC relativo al polinomio generatore $x^3 + 1$.

Soluzione a pagina 22 →

Esercizio 5 – Checksum di un'intestazione con parole di 4 bit

Si consideri un'intestazione con parole da 4 bit:

$$b_0 = 1001 = 9, \quad b_1 = 1100 = 12, \quad b_2 = 1010 = 10, \quad b_3 = 0011 = 3,$$

con operazioni modulo 15. Si calcoli la quinta parola, che costituisce il checksum.

Soluzione a pagina 23 →

Esercizio 6 – Appello del 12/01/2023, Canale 1, Prof.ssa Francesca Cuomo, domanda 4

Siano date le 4 stringhe di 4 bit riportate di seguito: b1) 1001; b2) 1011; b3) 0101; b4) 1010. Qual è la parola di codice calcolata mediante l'internet checksum?

a. 1111
b. 0101
c. 1010

Soluzione a pagina 23 →

Esercizio 7 – Appello del 08/02/2023, Canale 1, Prof.ssa Francesca Cuomo, domanda 6

Siano date le 4 stringhe di 4 bit riportate di seguito: b1) 1001; b2) 1011; b3) 0110; b4) 1010. Qual è la parola di codice calcolata mediante l'internet checksum?

a. 1001
b. 0110
c. 1111

Soluzione a pagina 24 →

Esercizio 8 – Appello del 12/06/2023, Canale 1, Prof.ssa Francesca Cuomo, esercizio

Si applichi alla stringa $P = 1110$ il meccanismo di generazione di una stringa binaria lato emettitore con CRC ottenuto attraverso un polinomio generatore $G(x) = x^3 + x + 1$. Si derivi:

1. la stringa binaria T emessa lato emettitore;
2. una stringa d'errore E_1 che sommata a T *non* dia errore in ricezione; E_1 deve essere diversa da $E = 0001011$;
3. una stringa d'errore E_2 che sommata a T dia errore in ricezione.

Soluzione a pagina 24 →

11 Soluzioni

Soluzione dell'Esercizio 1 – Codice di Hamming (7,4) con due errori

a. Nella stringa ricevuta $P_1P_2D_1P_3D_2D_3D_4 = 1111110$:

- gruppo di P_1: $P_1, D_1, D_2, D_4 = 1, 1, 1, 0$, tre 1: fallisce;
- gruppo di P_2: $P_2, D_1, D_3, D_4 = 1, 1, 1, 0$, tre 1: fallisce;
- gruppo di P_3: $P_3, D_2, D_3, D_4 = 1, 1, 1, 0$, tre 1: fallisce.

Falliscono tutti e tre: la stringa non è una parola di codice, e l'errore è rivelato.

b. Con tutti e tre i controlli falliti la tabella della correzione indica D_4, l'unico bit nei tre gruppi. Il ricevitore lo inverte e ottiene 1111111, una parola di codice: ora ogni gruppo ha quattro 1. Il dato letto è 1111 invece di 1010.

c. Gli errori sono 2, più di $t = \lfloor (3-1)/2 \rfloor = 1$. La stringa ricevuta dista 2 dalla parola trasmessa e 1 da 1111111, quindi la parola più vicina è quella sbagliata. Usato solo per rivelare, il codice avrebbe segnalato l'errore senza correggerlo.

← *Torna all'Esercizio 1, pagina 18*

Soluzione dell'Esercizio 2 – Parità bidimensionale con 2, 3 e 4 errori

Si conta il numero di 1 di ogni riga e di ogni colonna, bit di parità compreso: fallisce chi ne ha un numero dispari.

(a) **Due errori**, in riga 2 e riga 4 della colonna 2. La riga 2 ha un solo 1 e la riga 4 tre: falliscono. La colonna 2 ha perso due 1 e resta pari, e nessuna colonna fallisce. L'errore è rivelato, ma senza una colonna non c'è un incrocio: non si sa quali bit correggere.

(b) **Tre errori**. La riga 2 ha due errori e resta pari (0 0 0 1 0 1); la riga 4 ne ha uno e fallisce. La colonna 2 ha due errori e resta pari; la colonna 4 ne ha uno e fallisce $(1, 1, 1, 1, 1)$. L'errore è rivelato, ma l'incrocio è il bit in riga 4 e colonna 4, che è corretto: invertirlo dà una matrice con quattro bit sbagliati.

(c) **Quattro errori**, ai vertici del rettangolo delle righe 2 e 4 e delle colonne 2 e 4. Ogni riga e ogni colonna coinvolta ha due errori e resta pari: non fallisce niente, la matrice ricevuta è una parola di codice e l'errore non è rivelato.

← *Torna all'Esercizio 2, pagina 18*

Soluzione dell'Esercizio 3 – Bit stuffing e destuffing

1. Stuffing. La sequenza è il contenuto del frame, ancora senza flag. Si contano gli 1 consecutivi da sinistra e si inserisce uno 0 dopo ogni cinque 1:

- i primi tre 1 non danno problemi;
- poi ci sono sei 1: dopo il quinto si inserisce uno 0;
- poi tre 1, tre 1 e un 1 isolato;
- poi otto 1: dopo il quinto si inserisce uno 0, e ne restano tre;
- poi quattro 1, e infine sei 1: dopo il quinto si inserisce uno 0.

Con i tre 0 evidenziati la sequenza trasmessa ha $43 + 3 = 46$ bit:

$$01110111110100111011101011111\underline{0}1110111101111\underline{0}1$$

2. Destuffing. Il frame ricevuto comincia e finisce con un flag: dopo cinque 1 c'è un altro 1, e poi lo 0. Si tolgono i due flag 01111110, e nel contenuto si toglie lo 0 che segue ogni gruppo di cinque 1 (evidenziato):

$$11101111\underline{0}1111\underline{0}1111\underline{0}1111011000111\underline{0}101000$$

Il gruppo di quattro 1 prima di 0110 non ha stuffing: quello 0 resta. Il frame originale ha $44 - 4 = 40$ bit:

$$\boxed{1110111111111111111111011000111111101000}$$

Verifica: lo stuffing di questa sequenza ridà il contenuto ricevuto.

← *Torna all'Esercizio 3, pagina 19*

Soluzione dell'Esercizio 4 – Checksum e CRC dello stesso messaggio

1. Checksum. Con $w = 8$ bit si lavora modulo $2^8 - 1 = 255$. Il messaggio ha 16 bit, un multiplo di 8, e si divide in $L = 2$ parole:

$$b_0 = 10110001 = 128+32+16+1 = 177, \qquad b_1 = 01101010 = 64+32+8+2 = 106.$$

La somma è $177 + 106 = 283 \equiv 283 - 255 = 28 \pmod{255}$. È la somma, non ancora il checksum, che ne è l'opposto:

$$b_L \equiv -28 \equiv 255 - 28 = 227 \pmod{255}.$$

In binario, $227 = 128 + 64 + 32 + 2 + 1$, cioè $b_L = \boxed{11100011}$. Verifica: $283 + 227 = 510 = 2 \cdot 255 \equiv 0 \pmod{255}$.

2. CRC. $G(x) = x^3 + 1$ ha grado $z = 3$: la stringa dei coefficienti è 1001, e il resto ha tre bit. Si accodano tre 0 al messaggio e si divide lavorando solo sui coefficienti.

A ogni passo si scrive 1001 sotto il primo 1 rimasto e si somma modulo 2, finché restano meno di quattro bit:

passo	stringa dopo la somma
	10110001011010 000
1	00100001011010 000
2	00000101011010 000
3	00000011111010 000
4	00000000110010 000
5	00000000010110 000
6	00000000000100 10 000
7	00000000000000 000

Al settimo passo la stringa si annulla: $x^3M(x)$ è un multiplo di $G(x)$, e il resto è $R = \boxed{000}$. La stringa trasmessa è M seguito da 000.

Lo stesso messaggio ha quindi due ridondanze diverse: 8 bit con il checksum, 3 con il CRC.

← *Torna all'Esercizio 4, pagina 19*

Soluzione dell'Esercizio 5 – Checksum di un'intestazione con parole di 4 bit

Con $w = 4$ si lavora modulo $2^4 - 1 = 15$, e le parole sono già convertite. La somma è

$$9 + 12 + 10 + 3 = 34 \equiv 34 - 30 = 4 \pmod{15},$$

e il checksum è l'opposto: $b_4 \equiv -4 \equiv 15 - 4 = 11 \pmod{15}$. In 4 bit, $11 = 8+2+1$, cioè $b_4 = \boxed{1011}$. Verifica: $34 + 11 = 45 = 3 \cdot 15 \equiv 0 \pmod{15}$.

← *Torna all'Esercizio 5, pagina 19*

Soluzione dell'Esercizio 6 – Appello del 12/01/2023, Canale 1, Prof.ssa Francesca Cuomo, domanda 4

La «parola di codice» chiesta è il checksum b_L, con $L = 4$ blocchi: le stringhe b1, …, b4 del testo sono $b_0, \ldots, b_3$. Con blocchi di $w = 4$ bit si lavora modulo $2^4 - 1 = 15$.

1. I blocchi valgono $1001 = 9$, $1011 = 11$, $0101 = 5$, $1010 = 10$.
2. La somma è $9 + 11 + 5 + 10 = 35 \equiv 35 - 30 = 5 \pmod{15}$.
3. Il checksum è $b_L \equiv -5 \equiv 10 \pmod{15}$, cioè 1010.
4. Verifica: $35 + 10 = 45 = 3 \cdot 15 \equiv 0 \pmod{15}$.

La b, $0101 = 5$, è la somma senza il segno meno; la a, 1111, vale $15 \equiv 0$ e sarebbe il checksum solo se la somma fosse già 0. Risposta: $\boxed{\text{c}}$.

← *Torna all'Esercizio 6, pagina 20*

Soluzione dell'Esercizio 7 – Appello del 08/02/2023, Canale 1, Prof.ssa Francesca Cuomo, domanda 6

Come nell'esercizio precedente, si cerca il checksum b_L modulo $2^4 - 1 = 15$.

1. I blocchi valgono $1001 = 9$, $1011 = 11$, $0110 = 6$, $1010 = 10$.
2. La somma è $9 + 11 + 6 + 10 = 36 \equiv 36 - 30 = 6 \pmod{15}$.
3. Il checksum è $b_L \equiv -6 \equiv 9 \pmod{15}$, cioè 1001.
4. Verifica: $36 + 9 = 45 \equiv 0 \pmod{15}$.

La b, $0110 = 6$, è la somma senza il segno meno; la c, $1111 \equiv 0$, non annulla la somma. Risposta: $\boxed{\text{a}}$.

← *Torna all'Esercizio 7, pagina 20*

Soluzione dell'Esercizio 8 – Appello del 12/06/2023, Canale 1, Prof.ssa Francesca Cuomo, esercizio

1. Stringa emessa. Il messaggio è $P(x) = x^3 + x^2 + x$ e $z = 3$, quindi $x^3 P(x) = x^6 + x^5 + x^4$, cioè 1110 000. Si divide per $G(x) = x^3 + x + 1$:

1. $x^3\,G(x) = x^6 + x^4 + x^3$; la somma dà $x^5 + x^3$;
2. $x^2\,G(x) = x^5 + x^3 + x^2$; la somma dà x^2.

Il resto x^2 ha grado $2 < 3$, cioè $R = 100$. La stringa emessa è $T = 1110\,100 = \boxed{1110100}$.

2. Errore non rivelato. Un errore sfugge se e solo se è un multiplo non nullo di $G(x)$. $E = 0001011$ è $G(x)$ stesso. Un altro multiplo è $x\,G(x) = x^4 + x^2 + x$, cioè $E_1 = \boxed{0010110}$. La stringa ricevuta è

$$T + E_1 = 1110100 + 0010110 = 1100010,$$

che è la parola di codice del messaggio 1100 dell'esempio precedente: il resto è 0 e il ricevitore accetta, anche se tre bit sono sbagliati. Va bene qualunque multiplo $H(x)\,G(x)$ con $H(x) \neq 0, 1$: con $H(x) = x + 1$ si ottiene $E_1(x) = x^4 + x^3 + x^2 + 1$, cioè 0011101, e $T + E_1 = 1101001$ ha resto 0.

3. Errore rivelato. Basta un errore che non sia un multiplo di $G(x)$. Un solo bit invertito va bene: con $E_2 = \boxed{0000001}$ la stringa ricevuta è 1110101 e il resto è quello di $E_2(x) = 1$, cioè $001 \neq 000$. Il ricevitore rivela l'errore. Un altro modo è cambiare

un solo bit di un E_1 non rivelato: la differenza fra i due errori è un bit, che non è un multiplo di $G(x)$, quindi il resto non è più 0.

← *Torna all'Esercizio 8, pagina 20*