ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Programmazione di Reti
Laboratorio #5

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

Esercizio su CRC (Cyclic Redundancy Check)

CRC (Cyclic Redundancy Check)

Il CRC (Ciclic Redundancy Check) o codice a ridondanza ciclica è un codice adatto a riscontrare errori di tipo "**burst**" o a raffica, ovvero errori che compromettono più bit consecutivi.

Questo tipo di codice viene anche chiamato **codice polinomiale** in quanto i bit di dati da controllare possono essere considerati come coefficienti (di valore 0 o 1) di un polinomio che chiameremo **M(x)**.

Per calcolare il CRC, oltre al polinomio che rappresenta l'informazione da trasmettere, abbiamo bisogno di un polinomio detto **generatore** che chiameremo **G(x)** e che dovrà rispettare alcune regole:

- **G(x)** deve essere noto sia al mittente che al destinatario
- I bit di **G(x)** di ordine più alto e più basso devono essere a 1
- il grado di **M(x)** deve essere maggiore di quello di G(x).

CRC (Cyclic Redundancy Check)

Calcoliamo il CRC

Definiamo le sequenze di bit che utilizzeremo (dati e generatore) e i rispettivi polinomi:

$M(x) = 1110$ → **Messaggio da inviare**

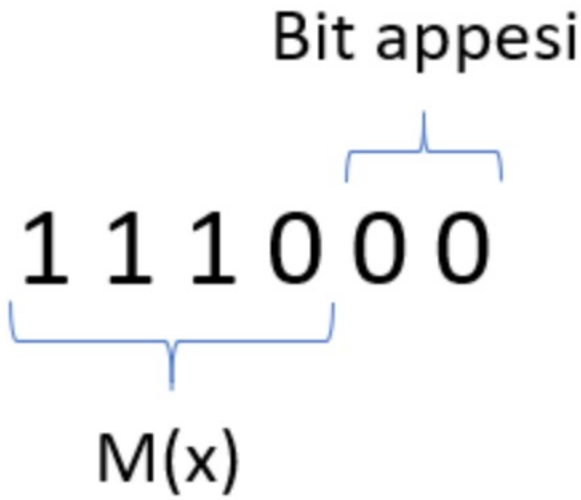
$M(x) = 1 \cdot x^3 + 1 \cdot x^2 + 1 \cdot x + 0 \cdot 1$ → GRADO 3

$G(x) = 101$ → **Polinomio CRC**

$G(x) = 1 \cdot x^2 + 0 \cdot x + 1 \cdot 1$ → GRADO 2

CRC (Cyclic Redundancy Check)

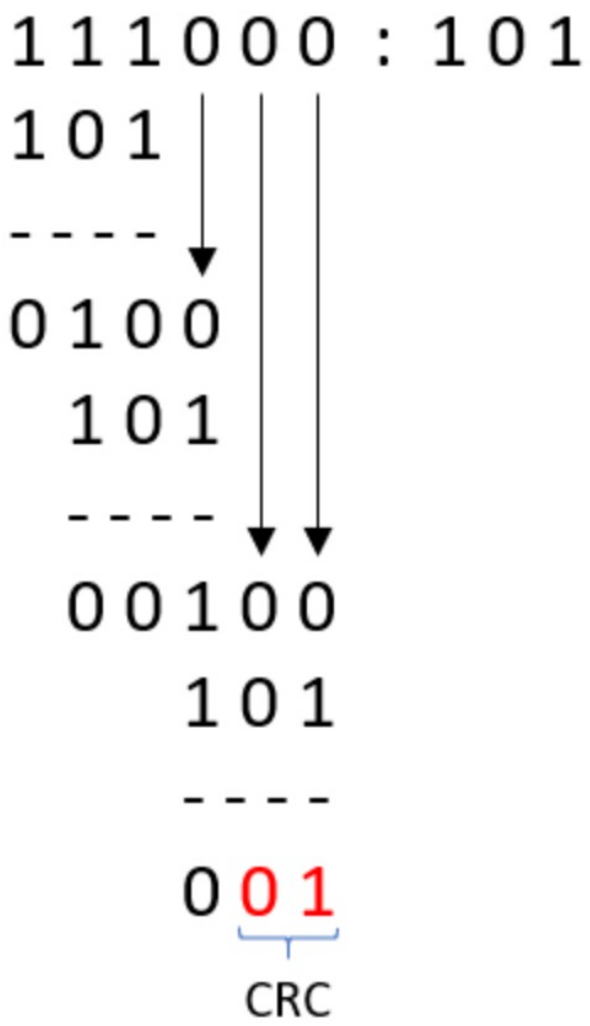
A questo punto “appendiamo” ai bit di M(x) un numero di zeri pari al grado del polinomio G(x):



Dividiamo la nuova sequenza cosi ottenuta per G(x).
La parte che ci interessa è il **resto**.

Per fare questa divisione possiamo avvalerci dell’operatore logico **XOR** (OR esclusivo).

CRC (Cyclic Redundancy Check)

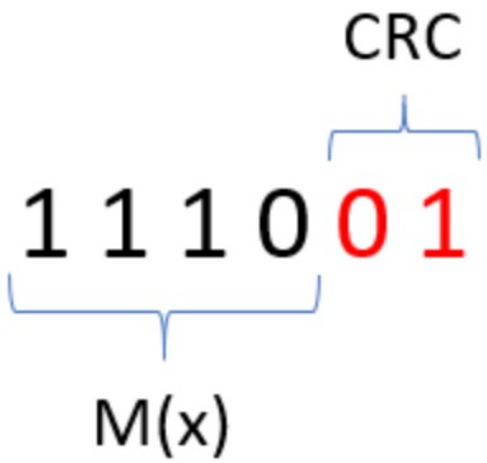


A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

CRC (Cyclic Redundancy Check)

Aggiungiamo il CRC alla parte dati:

Dopo aver calcolato il codice di controllo CRC «appendiamolo» alla nostra sequenza di dati:



La sequenza cosi ottenuta verrà spedita al destinatario.

CRC (Cyclic Redundancy Check)

Una volta che la sequenza è arrivata a destinazione dobbiamo **verificarne la correttezza**, quindi dividiamo la sequenza per il generatore G(x), che come abbiamo detto prima è noto sia al mittente che al destinatario.

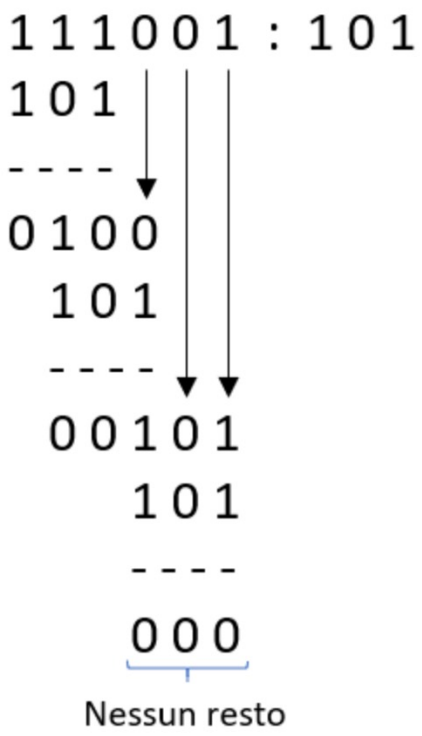
Per questa operazione valgono le stesse regole che abbiamo usato in precedenza.

Anche in questo caso dobbiamo porre la nostra attenzione sul resto della divisione.

Se il **resto è diverso da 0** si è verificato un **errore**.

CRC (Cyclic Redundancy Check)

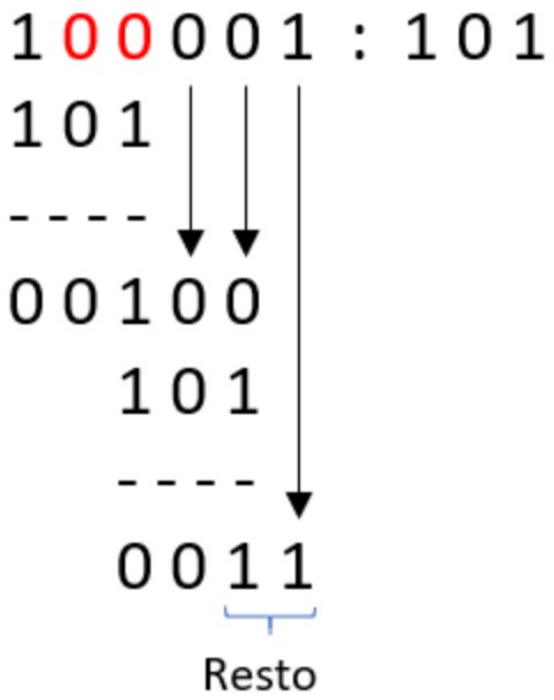
Verifica integrità dati



Il **risultato è privo di resto**, per cui i dati ricevuti sono **corretti**.

CRC (Cyclic Redundancy Check)

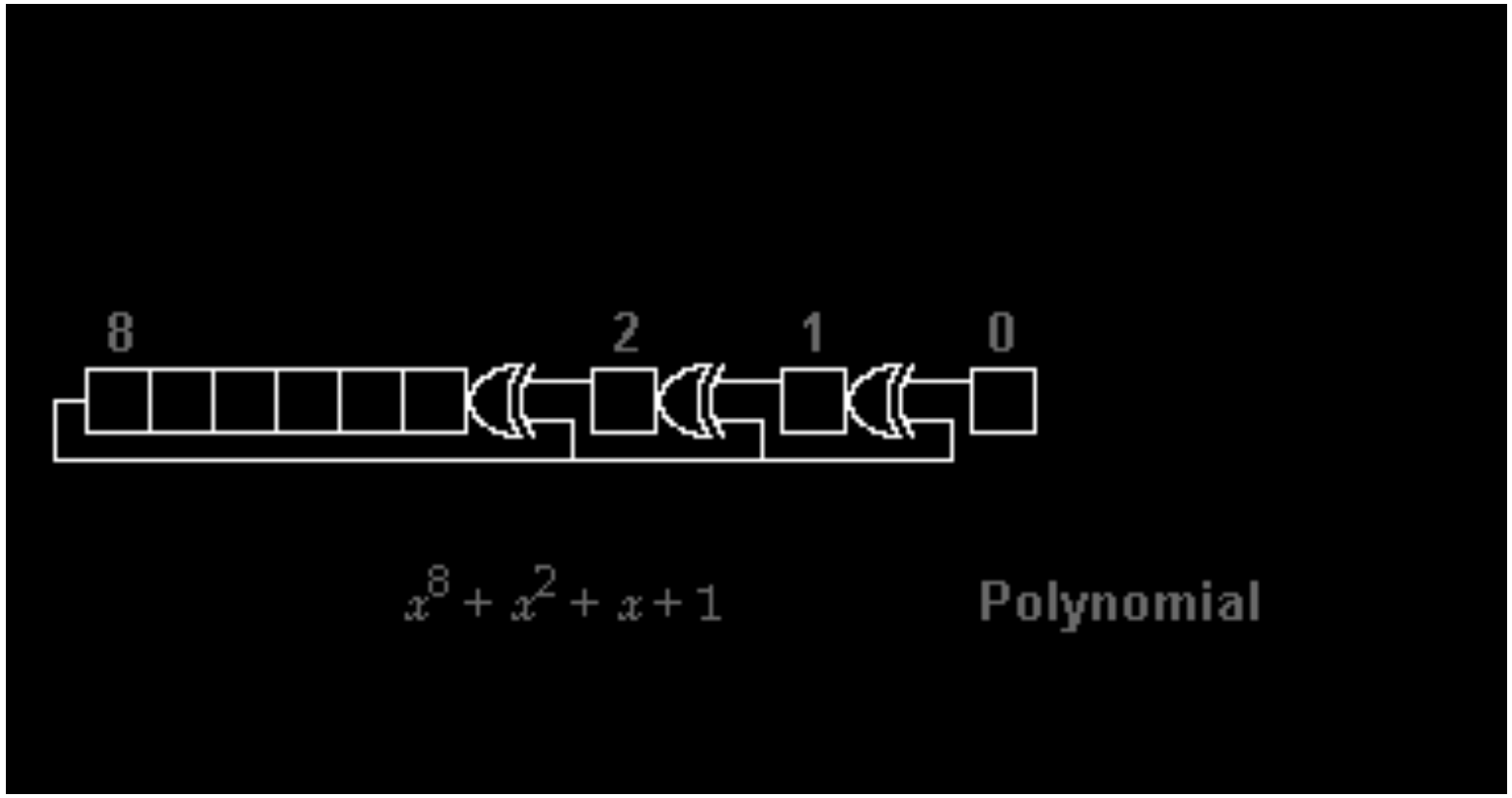
Ora proviamo a vedere cosa succederebbe se la sequenza subisse delle modifiche durante la trasmissione (in rosso sono evidenziate le modifiche):



In questo caso la divisione ci da un resto diverso da zero, quindi i dati ricevuti sono ERRORATI

CRC (Cyclic Redundancy Check)

la scelta del polinomio generatore è molto importante e negli anni si sono affermati diversi tipi di **standard**.



CRC (Cyclic Redundancy Check)- LATO RICEVITORE

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 12 14:47:29 2023

@authors: Andrea PIRODDi, Franco CALLEGATI, Roberto GIRAU
"""

# Importiamo il modulo socket
import socket
#definiamo la funzione XOR
def xor(a, b):

    # initializziamo la lista che chiamiamo result
    result = []

    # Scorriamo tutti i bits, se ci sono bits uguali
    # allora XOR è 0, altrimenti 1 (vedi tabella di verità nelle slide)
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)
```

CRC (Cyclic Redundancy Check)- LATO RICEVITORE

```
26
27
28  # Definiamo la funzione che calcola il resto della divisione binaria
29  def mod2div(dividend, divisor):
30
31      # Numero di bits su cui applicare la XOR (dimensione del divisore).
32      pick = len(divisor)
33
34      # applichiamo lo Slicing al dividendo in funzione della
35      # lunghezza appropriata per ciascun passo
36      tmp = dividend[0 : pick]
37
38      while pick < len(dividend):
39
40          if tmp[0] == '1':
41
42              # rimpiazziamo il dividendo con il risultato della
43              # XOR e tiriamo giù 1 bit
44              tmp = xor(divisor, tmp) + dividend[pick]
45
46          else: # se il bit più a sinistra è '0'
47
48              # se il bit più a sinistra del dividendo (o la parte
49              # usata in ciascuno step) è 0, non possiamo
50              # usare il divisore regolare; dobbiamo usare un divisore
51              # all-0s.
52              tmp = xor('0'*pick, tmp) + dividend[pick]
53
54          # incrementiamo pick per muoverci in avanti
55          pick += 1
56
57      # Per gli ultimi n bits, eseguiamo l'operazione
58      # conclusiva onde evitare di avere un errore del tipo Index Out of Bound
59
60      if tmp[0] == '1':
61          tmp = xor(divisor, tmp)
62      else:
63          tmp = xor('0'*pick, tmp)
64
65      checkword = tmp
66      return checkword
```

CRC (Cyclic Redundancy Check)- LATO RICEVITORE

```
74
75    # Creiamo un oggetto socket
76    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
77    key = "1001"
78    # Definiamo la porta su cui vogliamo connetterci
79    port = 12345
80    server_address = ('localhost',port)
81    print ('\n\r starting up on %s port %s' % server_address)
82    s.bind(server_address)
83
84    while True:
85        print('\n\r waiting to receive message...')
86        data, address = s.recvfrom(1024)
87
88        print('received %s bytes from %s' % (len(data), address))
89        print (data.decode())
90        print(decodeData(data.decode(),key))
91        if decodeData(data.decode(),key)=='0'*(len(key)-1):
92            print("it's ok")
93        else:
94            print("it's not ok")
95
96
97
98    # chiudiamo la connessione
99    s.close()
```

CRC (Cyclic Redundancy Check)- LATO TRASMETTITORE

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 12 14:45:43 2023

@authors: Andrea PIRODDi, Franco CALLEGATI, Roberto GIRAU
"""

# Importiamo il modulo socket
import socket

def xor(a, b):

    # initializziamo la lista result
    result = []

    # Scorriamo tutti i bits, se ci sono bits uguali
    # allora XOR è 0, altrimenti 1 (vedi tabella di verità nelle slide)
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)
```

CRC (Cyclic Redundancy Check)- LATO TRASMETTITORE

```
# Definiamo la funzione che calcola il resto della divisione binaria
def mod2div(dividend, divisor):

    # Numero di bits su cui applicare la XOR (dimensione del divisore).
    pick = len(divisor)

    # applichiamo lo Slicing al dividendo in funzione della
    # lunghezza appropriata per ciascun passo
    tmp = dividend[0 : pick]

    while pick < len(dividend):

        if tmp[0] == '1':

            # rimpiazziamo il dividendo con il risultato della
            # XOR e tiriamo giù 1 bit
            tmp = xor(divisor, tmp) + dividend[pick]

        else: # se il bit più a sinistra è '0'

            # se il bit più a sinistra del dividendo (o la parte
            # usata in ciascuno step) è 0, non possiamo
            # usare il divisore regolare; dobbiamo usare un divisore
            # all-0s.
            tmp = xor('0'*pick, tmp) + dividend[pick]

        # incrementiamo pick per muoverci in avanti
        pick += 1

    # Per gli ultimi n bits, eseguiamo l'operazione
    # conclusiva onde evitare di avere un errore del tipo Index Out of Bound
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)

    checkword = tmp
    return checkword
```

CRC (Cyclic Redundancy Check)- LATO TRASMETTITORE

```
67  # Funzione usata dal lato mittente per applicare il CRC
68  # appendendo il resto della divisione
69  # al termine del messaggio.
70  def encodeData(data, key):
71
72      l_key = len(key)
73
74      # Appende n-1 zeri al termine del messaggio
75      appended_data = data + '0'*(l_key-1)
76      remainder = mod2div(appended_data, key)
77
78      # Appende il resto al messaggio originale
79      codeword = data + remainder
80      return codeword
81
82  server_address = ('localhost', 10000)
83  # Creiamo il socket UDP
84  s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
85  # Definiamo la porta su cui collegarci
86  port = 12345
87
88
89
90  input_string = input("Enter data you want to send->")
91  #s.sendall(input_string)
92  data =(''.join(format(ord(x), 'b') for x in input_string))
93  print("Entered data in binary format :",data)
94  key = "1001"
95
96  ans = encodeData(data,key)
97  print("Encoded data to be sent to server in binary format :",ans)
98  s.sendto(ans.encode(),('127.0.0.1', port))
99
100
101 # receviamo i dati dal server
102 print("Received feedback from server :",s.recv(1024).decode())
103
104 # chiudiamo la connessione
105 s.close()
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

CRC (Cyclic Redundancy Check)

RICEVITORE

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/apirodd/Downloads/codice/laboratorio_5/CRC_receiver.py',
wdir='/Users/apirodd/Downloads/codice/laboratorio_5')

 starting up on localhost port 12345

 waiting to receive message...
received 24 bytes from ('127.0.0.1', 53306)
100010111101101101110010
000
it's ok

 waiting to receive message...
```

TRASMETTITORE

```
Python 3.8.1 (default, Jan  8 2020, 16:15:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]: runfile('/Users/apirodd/Downloads/codice/laboratorio_5/CRC_sender.py',
wdir='/Users/apirodd/Downloads/codice/laboratorio_5')

Enter data you want to send->Evn
Entered data in binary format : 10001011101101101110
Encoded data to be sent to server in binary format : 100010111101101101110010
Received feedback from server : it's ok

In [2]:
```

GO-BACK-N ARQ

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

GO-BACK-N ARQ

Il protocollo Go-Back-N, chiamato anche Go-Back-N Automatic Repeat reQuest, è un protocollo a livello ***Data Link***, che utilizza un metodo a finestra scorrevole per la consegna affidabile e sequenziale di frame di dati.

È un caso di protocollo a finestra scorrevole che deve inviare la dimensione della finestra di N e ricevere la dimensione della finestra di 1.

Potete trovare un simulatore grafico al seguente indirizzo:

https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

GO-BACK-N ARQ

Selective Repeat / Go Back N

configuration

protocol
- Go back N
- Selective Repeat

choosing a new protocol restarts the simulation

window size
5
sets the window size for the windows

end to end delay
5000
time a packet takes from one station to the other

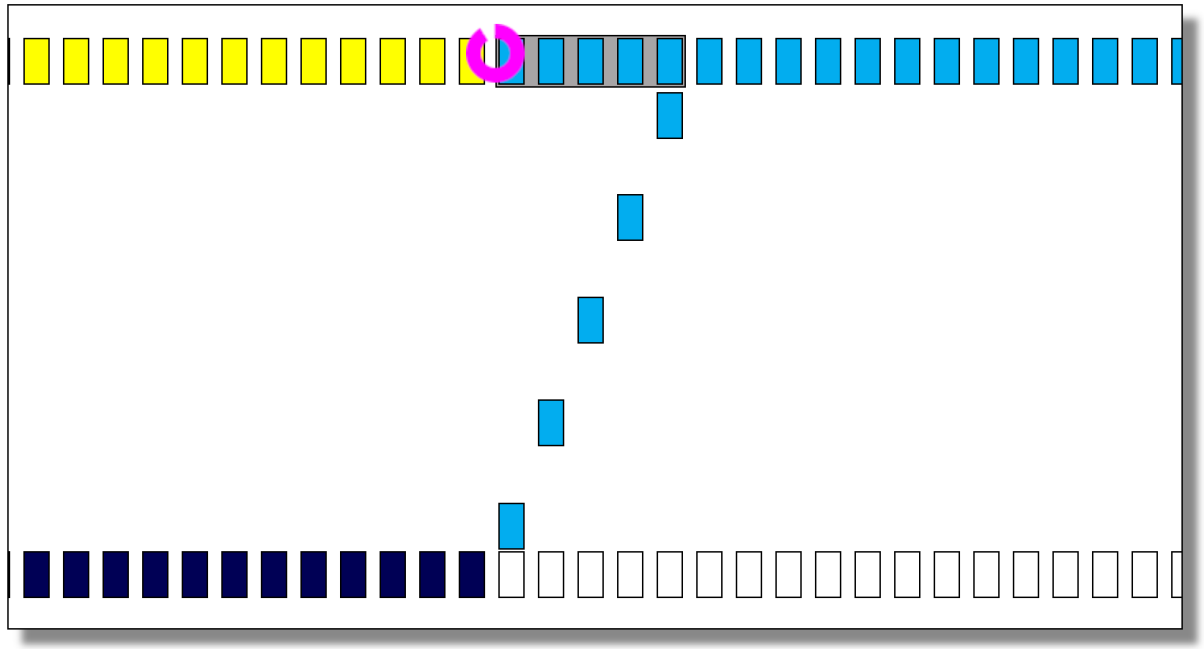
timeout
11000

scroll mode
Typewriter style
change the style the window scrolls

number of packets emited per minute
60
the number of packets the upper layer tries to send per minute

automatic emission of packets
stop
starts or stops the automatic emission of packets by the upper layer

legend
- no data received yet
- data buffered (ready to send, delivered or sent but no ack received yet)
- ack
- transmission confirmed
- data has been delivered to upper network layer



coded by Johannes Kessler 2012

GO-BACK-N ARQ - Principio di funzionamento

Go – Back – N ARQ prevede l'invio di più frame prima di ricevere l'acknowledgement per il primo frame.

I frame sono numerati in sequenza e sono in numero finito.

Il numero massimo di frame che possono essere inviati dipende dalla dimensione della finestra di invio.

Se **l'Acknowledgement** di un frame non viene ricevuto entro un periodo di tempo concordato, tutti i frame a partire da quel frame vengono ritrasmessi.

La dimensione della finestra di invio determina il numero di sequenza dei frame in uscita. Se il numero di sequenza dei frame è un campo di n bit, allora l'intervallo di numeri di sequenza che possono essere assegnati va da $0\ a\ 2^n - 1$.

GO-BACK-N ARQ

Di conseguenza, la dimensione della finestra di invio è $2^n - 1$.

Pertanto, per adattarsi a una dimensione della finestra di invio di $2^n - 1$, viene scelto un numero di sequenza di n bit.

I numeri di sequenza sono numerati come modulo-n.

Ad esempio, se la dimensione della finestra di invio è 4, i numeri di sequenza saranno 0, 1, 2, 3, 0, 1, 2, 3, 0, 1 e così via.

Il numero di bit nel numero di sequenza è 2 per generare la sequenza binaria 00, 01, 10, 11.

La dimensione della finestra di ricezione è 1.

GO-BACK-N ARQ - TRASMETTITORE

```python
# GO-BACK-N-Automatic Repeat reQuest

# Sender.py
"""
Created on Thursday Apr 19 14:47:29 2023

@authors: Franco CALLEGATI, Andrea PIRODDi,Roberto GIRAU
"""
import time, socket, sys
# definizione funzione di conversione decimale in binario
def decimalToBinary(n):
    return n.replace("0b", "0")

#definizione della funzione di trasposizione da utf-8 in binario
def binarycode(s):
    a_byte_array = bytearray(s, "utf8")
    byte_list = []

    for byte in a_byte_array:
        binary_representation = bin(byte)
        byte_list.append(decimalToBinary(binary_representation))

    #print(byte_list)
    a=""
    for i in byte_list:
        a=a+i
    return a
#messaggio di benvenuto
print("\n Benvenuti nella Chat Room\n")
print("Inizializzazione....\n")
time.sleep(1)
#creazione del socket utilizzando làip address reale della macchina
s = socket.socket()
host = socket.gethostname()
ip = socket.gethostbyname(host)
port = 1235
s.bind((host, port))
print(host, "(", ip, ")\n")
name = input(str("Inserisci il tuo nome: "))

s.listen(1)
print("\n In attesa di connessioni in entrata...\n")
conn, addr = s.accept()
print("Ricevuta connessione dal", addr[0], "(", addr[1], ")\n")
```

Funzione per la conversione da decimale in binario

Funzione per passare da utf-8 in binario

GO-BACK-N ARQ - TRASMETTITORE

```
45
46      s_name = conn.recv(1024)
47      # recupero il nome di chi si è collegato in chat
48      s_name = s_name.decode()
49      print(s_name, "si è collegato alla chat room\n Usa [e] per uscire dalla chat room\n")
50      conn.send(name.encode())
51
52      while True:
53          message = input(str("Io : "))
54          conn.send(message.encode())
55          if message == "[e]":
56              message = "Abbandona la chat room!"
57              conn.send(message.encode())
58              print("\n")
59              break
60          message=binarycode(message)
61          print(message)
62          f=str(len(message))
63          conn.send(f.encode())
64
65          i=0
66          j=0
67          j=int(input("Inserisci la dimensione della finestra -> "))
68
69
70          b=""
71
72          j=j-1
73          f=int(f)
74          k=j
```

Stampiamo a video il messaggio in binario che vogliamo inviare

f è la variabile che contiene ilnumero di bit di cui è composto il messaggio

GO-BACK-N ARQ - TRASMETTITORE

```
    while i!=f:
        while(i!=(f-j)):
            conn.send(message[i].encode())
            b=conn.recv(1024)
            b=b.decode()
            print(b)
            if(b!="ACK Lost"):
                time.sleep(1)
                print("Ack RICEVUTO! La finestra scorrevole è nel range da "+(str(i+1))+" a "+str(k+1))
                print("Ora inviamo il pacchetto successivo")
                i=i+1
                k=k+1
                time.sleep(1)
            else:
                time.sleep(1)
                print("Ack del data bit NON RICEVUTO! La finestra scorrevole resta nel range da "+(str(i+1))+" a "+str(k+1))
                print("Ora reinviamo lo stesso pacchetto")
                time.sleep(1)
        while(i!=f):

            conn.send(message[i].encode())
            b=conn.recv(1024)
            b=b.decode()
            print(b)
            if(b!="ACK Lost"):
                time.sleep(1)
                print("Ack RICEVUTO! La finestra scorrevole è nel range da "+(str(i+1))+" a "+str(k))
                if ((str(i+1))!=str(k)):
                    print("Ora inviamo il pacchetto successivo")
                i=i+1
                time.sleep(1)
            else:
                time.sleep(1)
                print("Ack del data bit NON RICEVUTO! La finestra scorrevole resta nel range da "+(str(i+1))+" a "+str(k))
                print("Ora reinviamo lo stesso pacchetto")
                time.sleep(1)
```

GO-BACK-N ARQ - RICEVITORE

```python
# GO-BACK-N-Automatic Repeat reQuest

# Receiver.py
"""
Created on Thursday Apr 19 14:47:29 2023

@authors: Franco CALLEGATI, Andrea PIRODDi,Roberto GIRAU
"""
import time, socket, sys
import random
#definisco la funzione per convertire il binario in testo utf-8
def bin2text(s): return "".join([chr(int(s[i:i+8],2)) for i in range(0,len(s),8)])


#messaggio di benvenuto
print("\n Benvenuto nella Chat Room\n")
print("Inizializzazione....\n")
time.sleep(1)

s = socket.socket()
shost = socket.gethostname()
ip = socket.gethostbyname(shost)
print(shost, "(", ip, ")\n")
host = input(str("Inserisci indirizzo IP del server: "))
name = input(str("\n Inserisci il tuo nome: "))
port = 1235
print("\n Tentativo di connessione al ", host, "(", port, ")\n")
time.sleep(1)
s.connect((host, port))
print("Connesso...\n")

s.send(name.encode())
s_name = s.recv(1024)
s_name = s_name.decode()
print(s_name, "si è unito alla Chat\n Usa [e] per uscire dalla chat room\n")
```

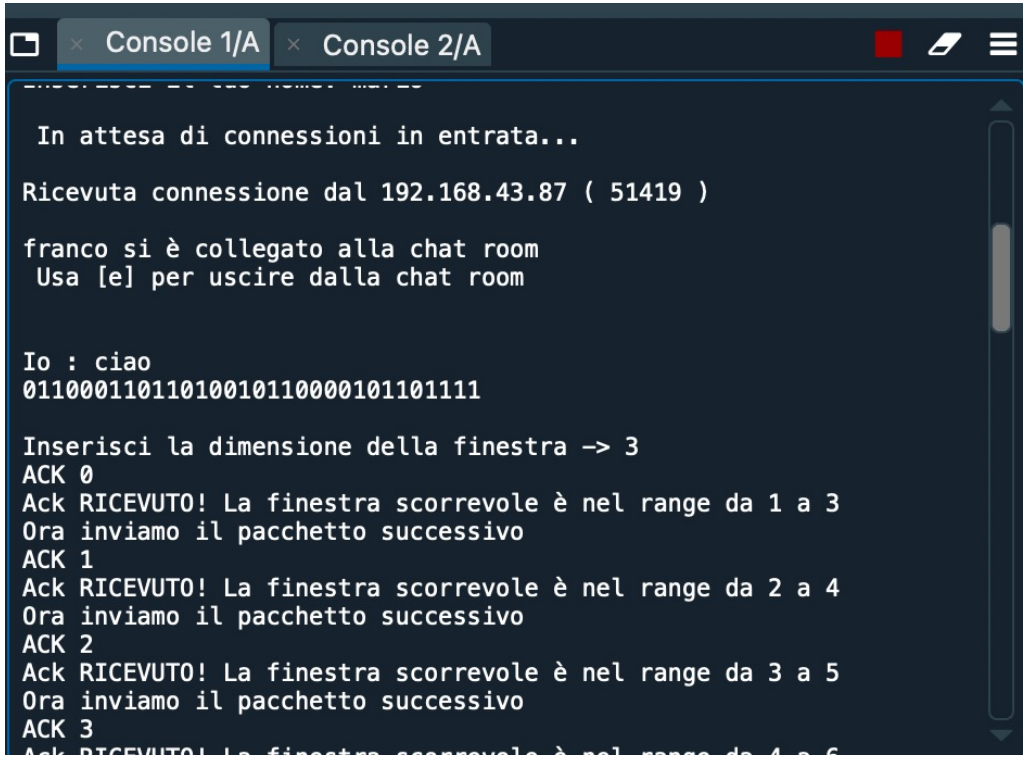
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

GO-BACK-N ARQ - RICEVITORE

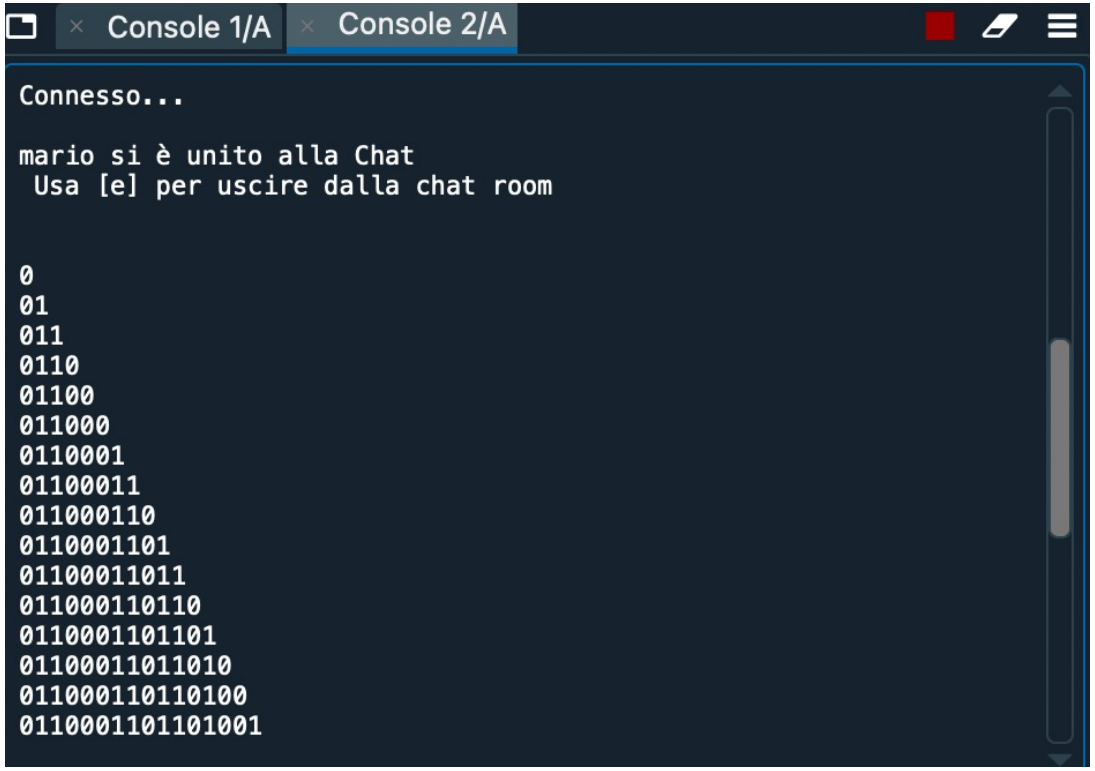
```
while True:

    m=s.recv(1024)
    m=m.decode()
    k=s.recv(1024)
    k=k.decode()
    k=int(k)
    i=0
    a=""
    b=""
    f=random.randint(0,1)
    message=""
    while i!=k:


        f=random.randint(0,1)
        if(f==0):
            b="ACK Lost"
            message = s.recv(1024)
            message = message.decode()
            s.send(b.encode())

        elif(f==1):
            b="ACK "+str(i)
            message = s.recv(1024)
            message = message.decode()
            print(a)
            s.send(b.encode())
            a=a+message
            i=i+1


    print("Il messaggio ricevuto è :", bin2text(a))
```

GO-BACK-N ARQ

trasmettitore

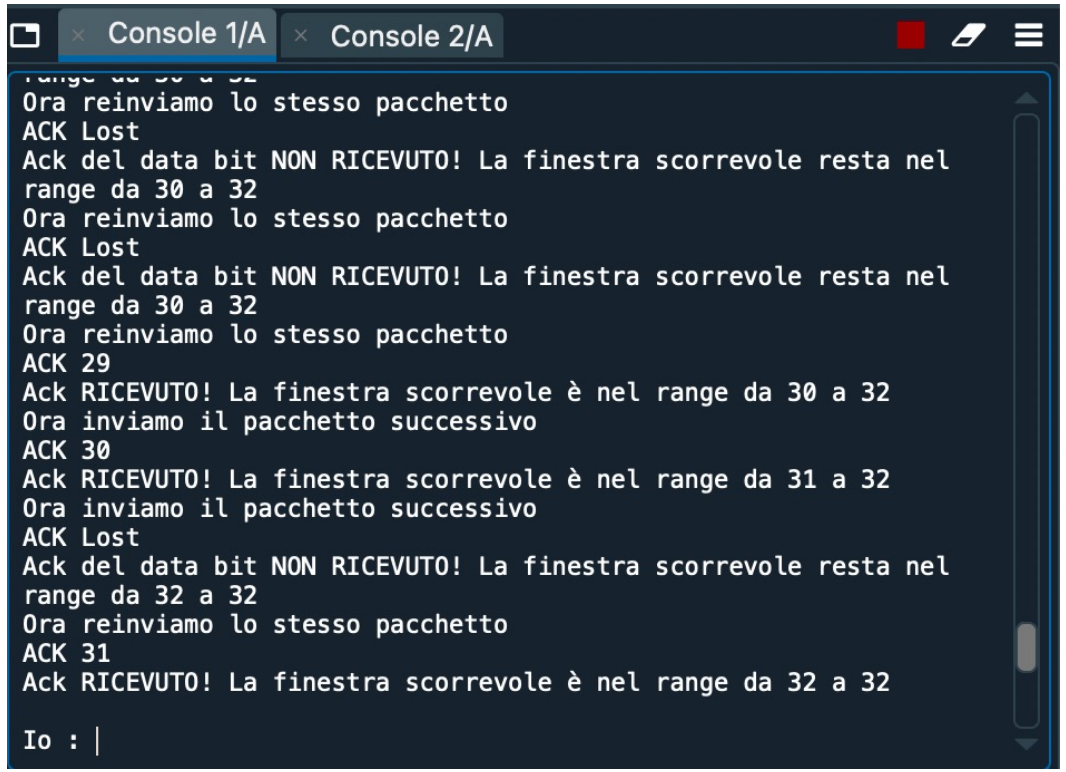


ricevitore

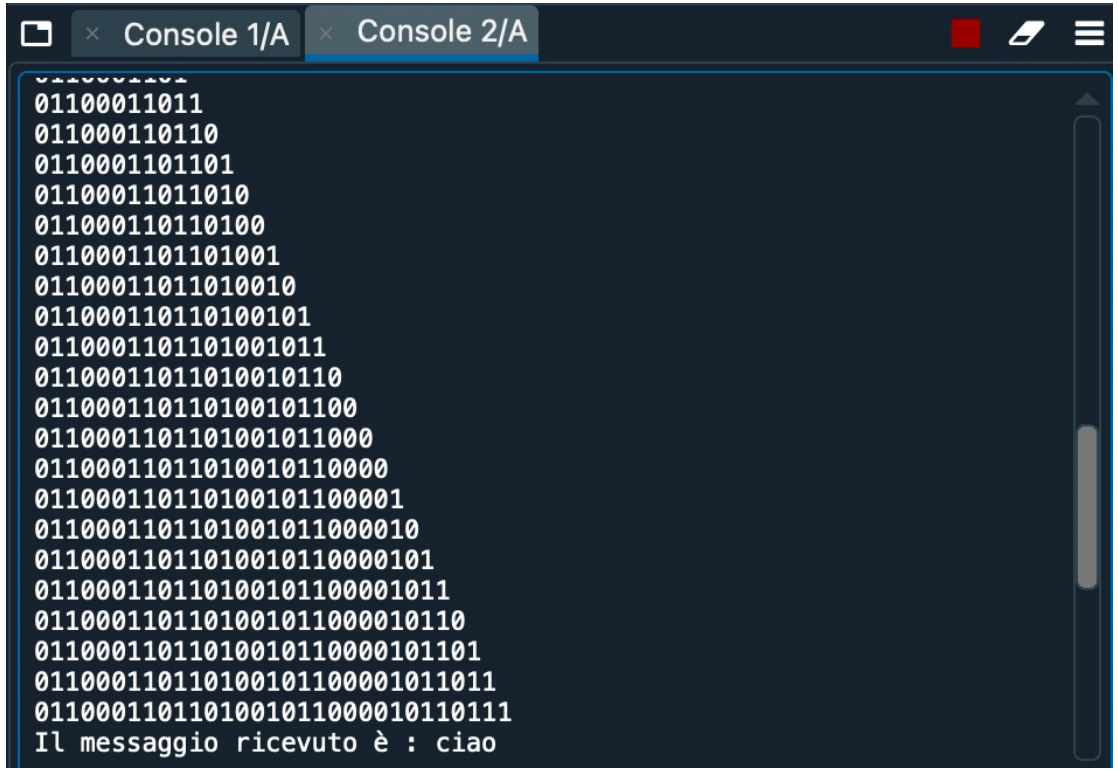


GO-BACK-N ARQ

trasmettitore



ricevitore



Esercizio 6 – Ritardi di Trasferimento

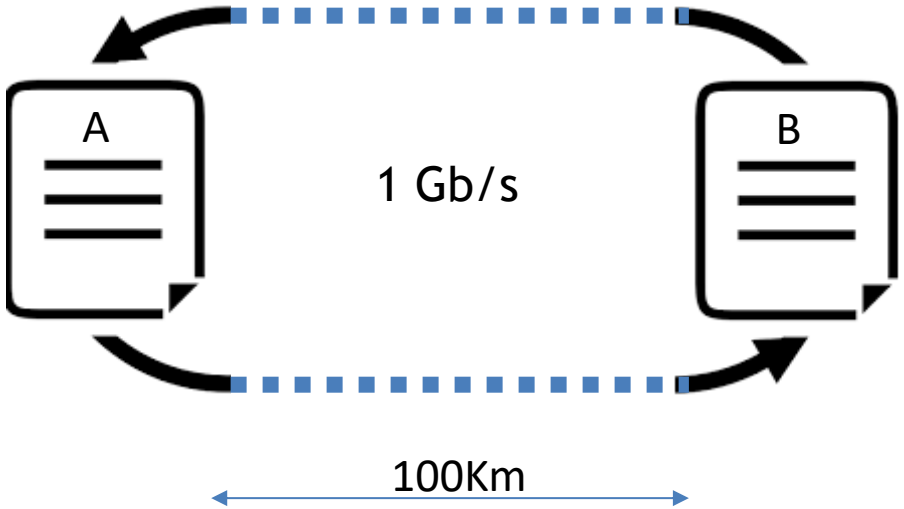
Esercizio 6 – Ritardi di Trasferimento

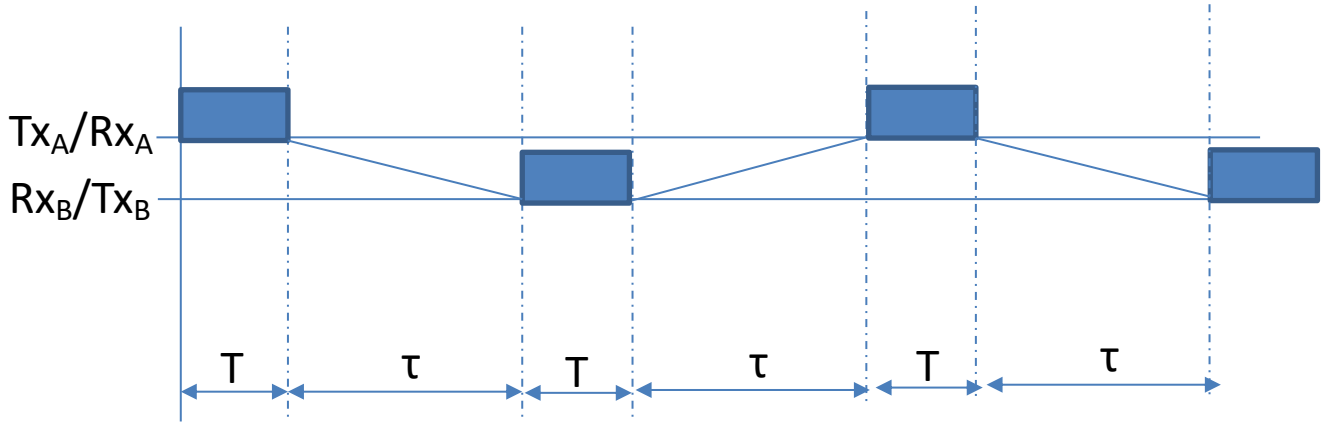
Due clienti di un internet service provider stanno utilizzando una connessione su un canale half-duplex lungo 100 km alla velocità di trasmissione di 1 Gb/s.
I due utenti debbono trasferirsi l'un l'altro dei files, ed eseguono la trasmissione in *half-duplex* commutando il canale ogni 10000 bit ricevuti.

Quesiti:

- Assumendo che la commutazione del canale avvenga in un tempo nullo e che la velocità del segnale sia di 200000 km/s, a che velocità effettiva trasmettono i files?
- Si calcoli il totale tempo di trasmissione di un file di 1Gbyte.

Esercizio 6 – Ritardi di Trasferimento – soluzione 1/2





Tempo di Trasmissione $T = \frac{L}{R} = \frac{10000\ bit}{10^9\ bit/s} = 10^{-6}s = 10\ \mu s$

Tempo di Propagazione $\tau = \frac{100\ km}{200 \cdot 10^3 km/s} = 0.5 \cdot 10^{-3} = 500\ \mu s$

«A» trasmette il suo blocco di dati, aspetta che sia arrivato, commuta il canale ed aspetta il blocco trasmesso da «B»

Ogni utente trasmette 10000 bit in un periodo di 2 x 510 μs = 1,02 ms

Il rate di trasferimento effettivo è: R = 10 kbit/1,02 ms = 9,8 Mbit/s

Esercizio 6 – Ritardi di Trasferimento – soluzione 2/2

File da trasferire 1Gbyte = 8Gbit

Tempo complessivo necessario a trasferire il file da 1 Gbyte

$$\boldsymbol{T = {8\ Gbit}/{9.8 \cdot 10^{-3} Gbit/s} = 0.816 \cdot 10^{3} s = 816\ s}$$

Oppure, si considerino il numero complessivo di trasmissioni necessarie per inviare un file di 8Gbit

$$N = \frac{8 \cdot 10^{9}\ bit}{10 \cdot 10^{3} bit} = 0.8 \cdot 10^{6} = 800000$$

Ogni trasmissione dura 1,02ms

$$\boldsymbol{T = 800000 \cdot 1.02\ ms = 816000\ ms = 816\ s}$$

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Esercizio 7 – Ritardi di Trasferimento

Esercizio 7 – Ritardi di Trasferimento

Un sistema trasmissivo della velocità di 100 kb/s presenta una lunghezza di 600 km tra Tx ed Rx. Fra i due elementi di testa sono presenti due router, ciascuno dei quali presenta una latenza (latency), ossia un tempo di accodamento in uscita, pari al tempo di trasmissione, ed una capacità di 100kb/s. Si consideri trascurabile il tempo di elaborazione.

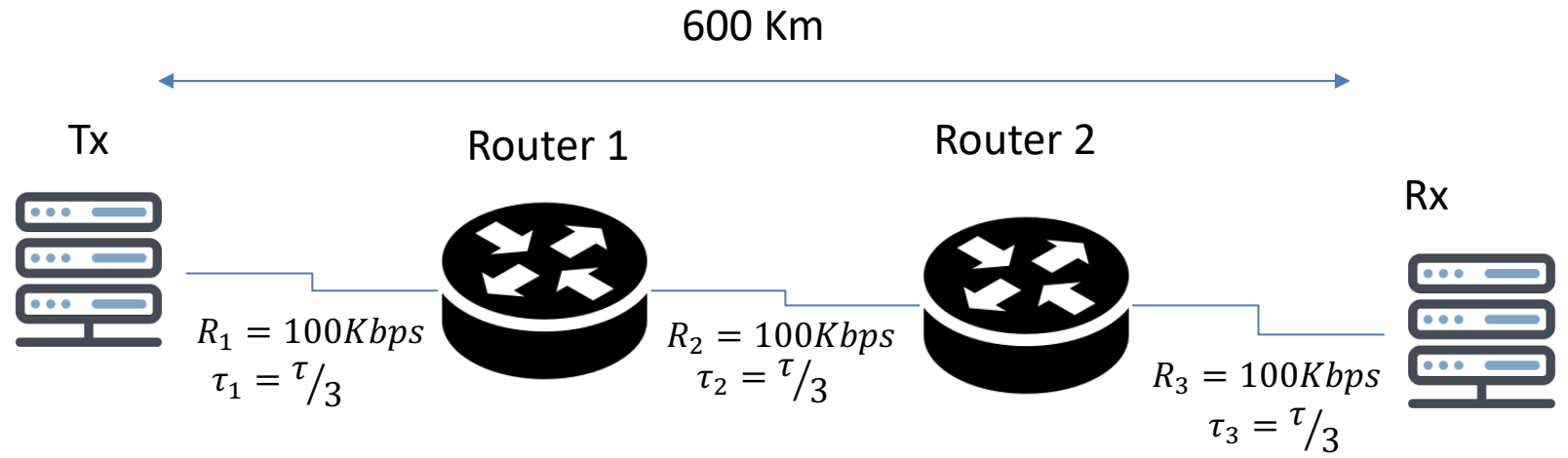
Quesiti:

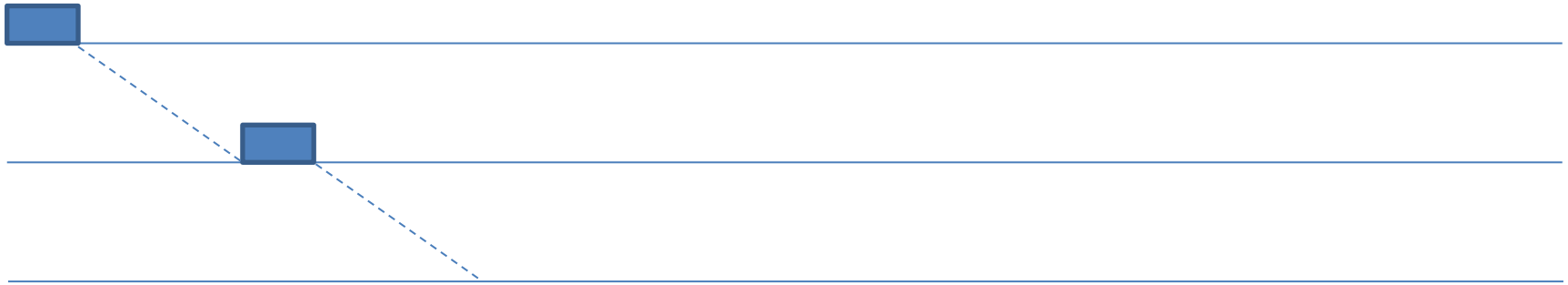
Si chiede di calcolare il ritardo totale nella trasmissione di un pacchetto di 3000 bit, assumendo un ritardo di propagazione di 5μs: nei due casi in cui nei router si applichi:

- la modalità ***store and forward***
- la modalità ***cut-through***, considerando che la lunghezza dell'*header* sia di 200 bit (a parità di dimensione totale del pacchetto).

Esercizio 7 – Ritardi di Trasferimento – Soluzione 1/5

Scenario: STORE & FORWARD → il pacchetto deve essere completamente ricevuto prima di essere ritrasmesso





Esercizio 7 – Ritardi di Trasferimento – Soluzione 2/5

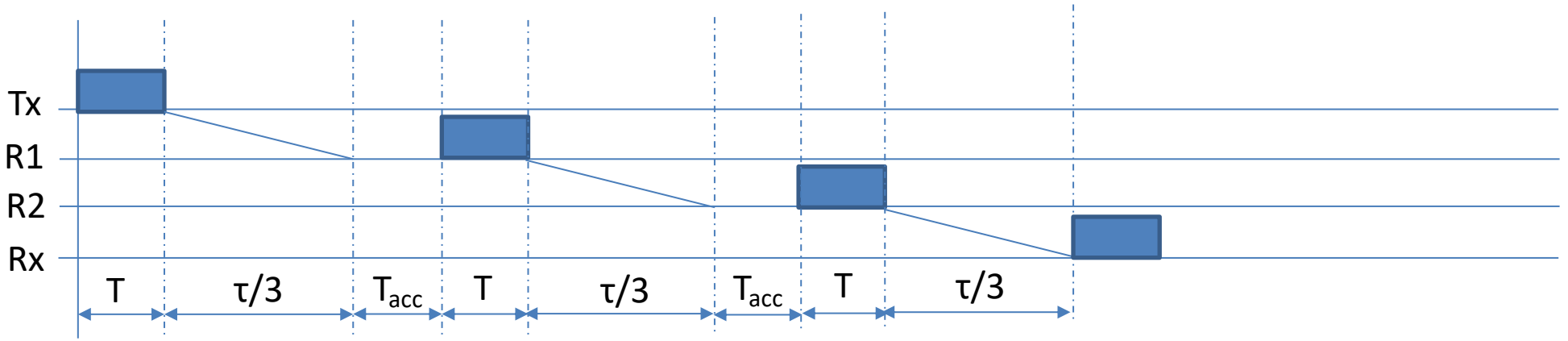
Scenario: STORE & FORWARD → il pacchetto deve essere completamente ricevuto prima di essere ritrasmesso

Osservazioni preliminari:

- Abbiamo 3 elementi trasmissivi quindi 3 tempi di trasmissione → $3 \cdot T = 3 \cdot \frac{L}{R} = 3 \cdot \frac{3000\ bit}{100 \cdot 10^3 bps} = 3 \cdot 30 \cdot 10^{-3} s = 90\ ms$
- Abbiamo 2 elementi che introducono latenza per accodamento e ritrasmissione → $2 \cdot T_{acc} = 2 \cdot T = 60\ ms$
- Abbiamo il tempo di propagazione fisico → $\tau = {}^{5\mu s}/_{Km} \cdot 600\ Km = 3000\ \mu s = 3.0\ ms$

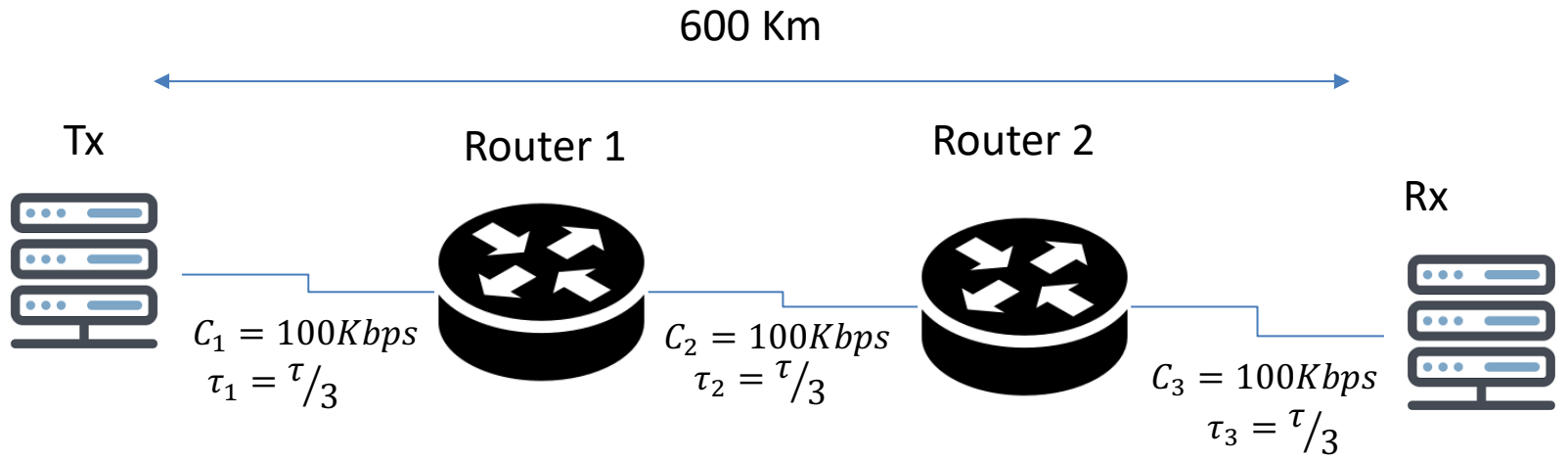
Sappiamo che il tempo di accodamento L = T, quindi facendo la somma otteniamo che:

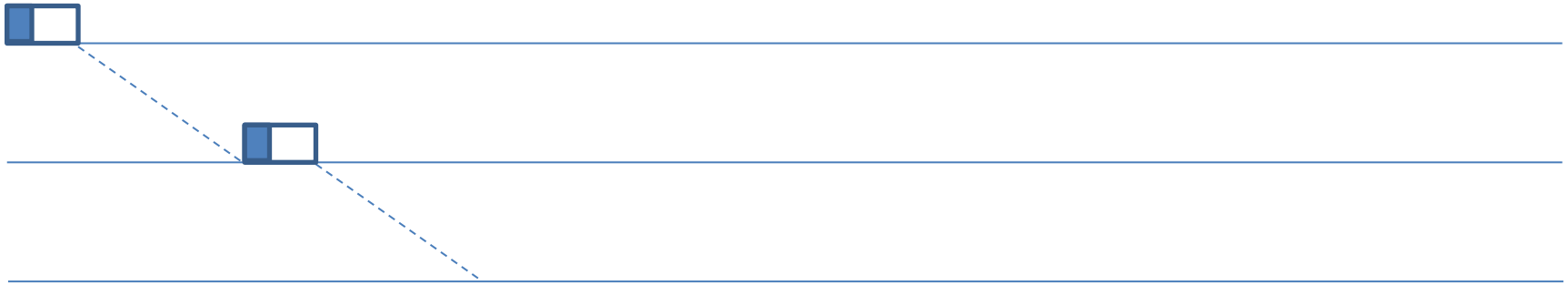
$$\boldsymbol{T_{tot} = 3 \cdot T + 2 \cdot T + \tau = 5 \cdot T + \tau = 5 \cdot 30ms + 3ms = 153ms}$$



Esercizio 7 – Ritardi di Trasferimento – Soluzione 3/5

Scenario: Cut & Through→ il pacchetto viene ritrasmesso alla completa ricezione dell'header



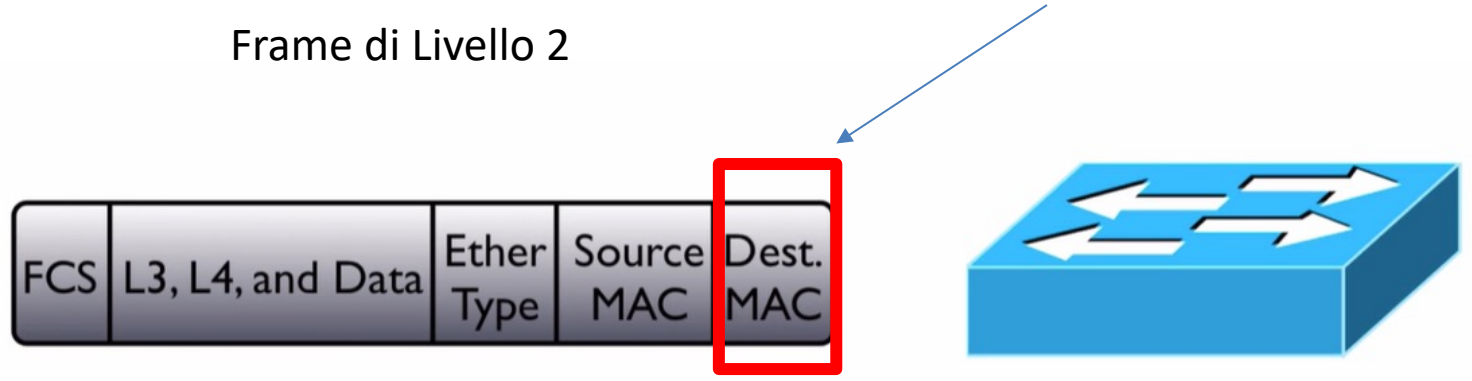


Esercizio 7 – Ritardi di Trasferimento – Soluzione 4/5

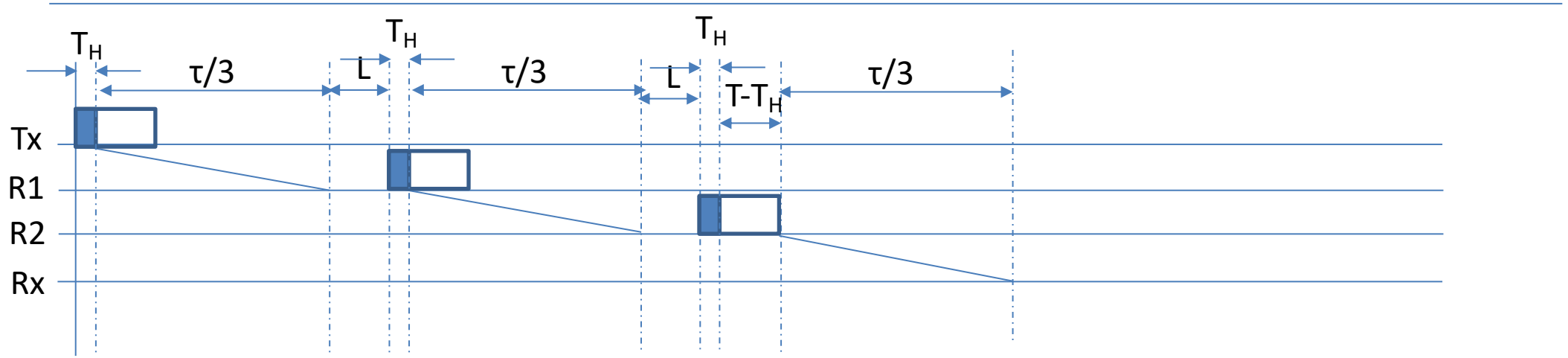
Scenario: Cut & Through→ il pacchetto viene ritrasmesso alla completa ricezione dell'header

Osservazioni preliminari:

Nel caso *cut-through* il tempo di trasmissione dell'*header* si paga tre volte (l'*header* è sempre trasmesso con modalità *store and forward*) mentre il tempo di trasmissione del resto si paga una sola volta. Il tempo totale è dunque



Esercizio 7 – Ritardi di Trasferimento – Soluzione 5/5



Tempo di trasmissione dell'Header

Tempo di trasmissione del resto del pacchetto

Tempo di accodamento dell'Header

Tempo di ritrasmissione del resto del pacchetto

Tempo di ritrasmissione dell'Header

Tempo di accodamento dell'Header

Tempo di ritrasmissione del resto del pacchetto

Tempo di ritrasmissione dell'Header

$$T_{tot} = \underbrace{T_H + (T - T_H)}_{\text{Server 1}} + \underbrace{T_H + (T - T_H) + T_H}_{\text{router 1}} + \underbrace{T_H + (T - T_H) + T_H}_{\text{router 2}} + \tau$$

Esercizio 7 – Ritardi di Trasferimento – Soluzione 5/5

$$T_H + (T - T_H) + T_H + (T - T_H) + T_H + T_H + (T - T_H) + T_H + \tau$$

$$T = \frac{L}{R} = \frac{3000\ bit}{100 \cdot 10^3 bps} = 30 \cdot 10^{-3} s = 30\ ms$$

$$\tau = {}^{5\mu s}/_{Km} \cdot 600\ Km = 3000\ \mu s = 3.0\ ms$$

$$\boldsymbol{T_{tot} = 3 \cdot T + 2 \cdot T_H + \tau = 90\ ms + 4\ ms + 3\ ms = 97 ms}$$

$$T_H = \frac{200\ bit}{100 \cdot 10^3 bit/sec} = 2 \cdot 10^{-3}\ \text{sec} = 2ms$$