



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Programmazione di Reti

Laboratorio 6

Andrea Piroddi

Dipartimento di Informatica, Scienza e Ingegneria

E-MAIL



E-Mail

La posta elettronica viene introdotta per la prima volta negli anni '60, tuttavia è diventata disponibile nella struttura attuale a partire dagli anni '70.

Protocolli utilizzati nei sistemi di posta elettronica

La comunicazione e-mail viene effettuata tramite tre protocolli in generale:

- IMAP
- POP
- SMTP



E-Mail - IMAP

IMAP è l'acronimo di **Internet Mail Access Protocol**.

Questo protocollo viene utilizzato durante la ricezione di un'e-mail.

Quando si utilizza IMAP:

- le e-mail saranno presenti nel server
- Le email non verranno scaricate nella casella di posta dell'utente
- Le email verranno successivamente eliminate dal server. Questo aiuta ad avere meno memoria utilizzata nel computer locale.



E-Mail - POP

POP è l'acronimo di Post Office Protocol.

Anche questo protocollo viene utilizzato per le e-mail in arrivo.

La differenza principale tra i due protocolli (imap e pop) è che POP scarica l'intera e-mail nel computer locale ed elimina i dati sul server una volta scaricati.

Questo è utile quando il server ha poca memoria libera.

La versione corrente di POP è POP3.



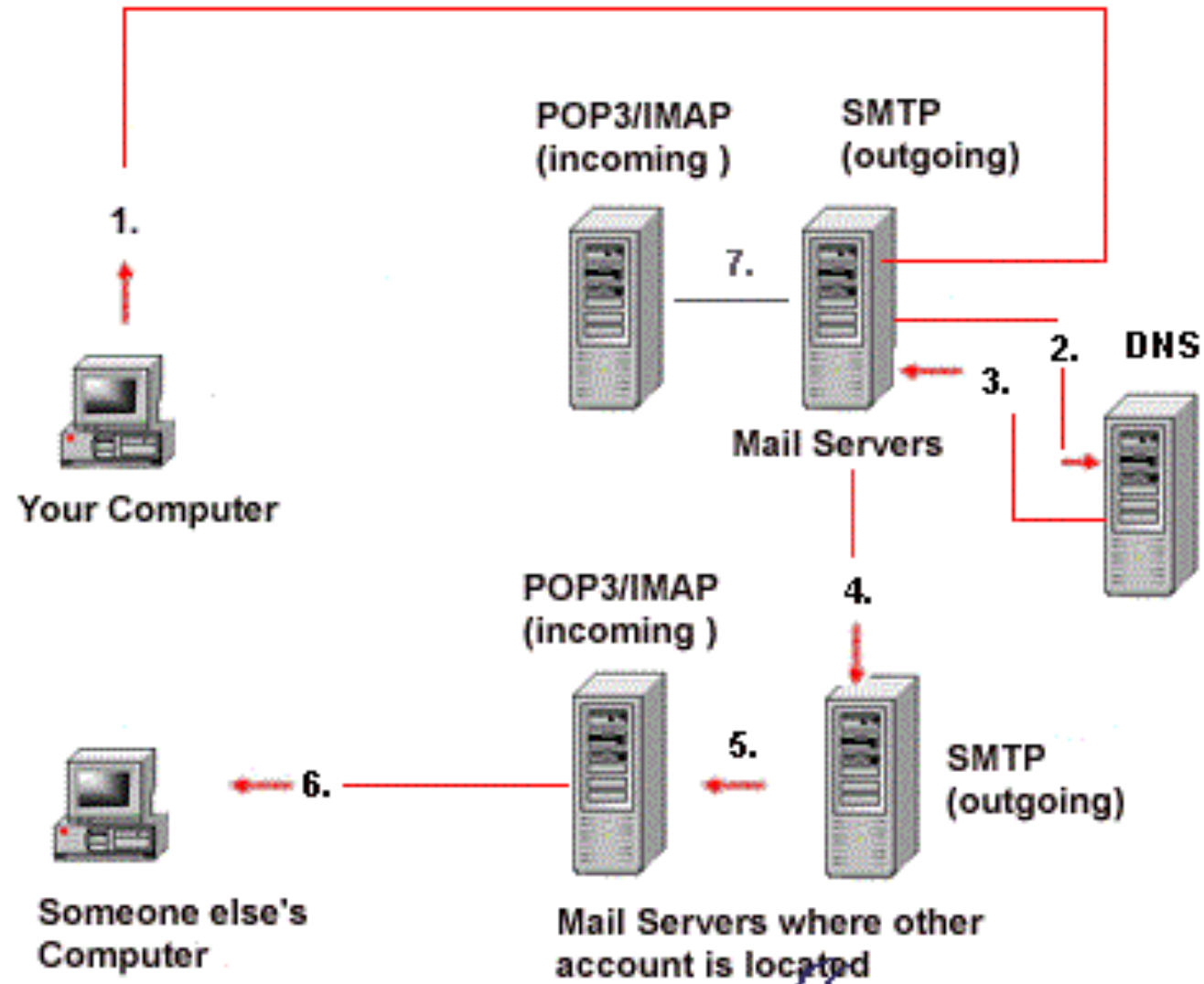
E-Mail - SMTP

SMTP sta per Simple Mail Transfer Protocol.

L'e-mail vengono inviate utilizzando questo protocollo.



E-Mail - architettura



E-Mail - Server

Un server di posta è un software.

Il software gestisce la ricezione delle e-mail in arrivo dagli utenti locali (persone all'interno dello stesso dominio) e dai mittenti remoti e inoltra le e-mail in uscita per la consegna.

Un computer su cui è installata un'applicazione di questo tipo può anche essere chiamato server di posta.

Nella figura precedente si vedono due server di posta.

I due server di posta utilizzati per le e-mail in uscita sono chiamati **MTA**, Mail Transfer Agent.

Gli altri due server di posta utilizzati per la posta in arrivo, che utilizzano i protocolli POP3/IMAP, sono chiamati **MDA**, Mail Delivery Agent.



E-Mail – HOW it Works

Il mittente inserisce l'indirizzo e-mail del destinatario insieme al messaggio utilizzando un'applicazione di posta elettronica (client SMTP). L'E-mail quindi verrà inviata all'MTA (Mail Transfer Agent). Questa comunicazione avviene tramite il protocollo SMTP.

Il passaggio successivo è la ricerca DNS. Il sistema invia una richiesta per conoscere l'MTA corrispondente del destinatario. Questo sarà fatto con l'aiuto del record MX. Nel DNS, in corrispondenza del dominio del destinatario, sarà presente un record MX (Mail Exchanger record). Questo specifica il server di posta di un dominio. Quindi, dopo la ricerca DNS, viene data una risposta al server di posta contenente l'indirizzo IP del server di posta del destinatario.

Il passaggio successivo è il trasferimento del messaggio tra i server di posta (server SMTP). Per questa comunicazione viene utilizzato il protocollo SMTP.

Ora, questo messaggio viene trasferito al MDA di destinazione e quindi viene trasferito al computer locale del destinatario. Qui possono essere utilizzati due protocolli. Se utilizziamo POP3, l'intera e-mail verrà scaricata sul computer locale e la copia sul server verrà eliminata. Se il protocollo utilizzato è IMAP, il messaggio e-mail viene archiviato nel server di posta stesso.

Se si è verificato un errore nell'invio dell'e-mail, le e-mail subiranno un ritardo. C'è una coda di posta in ogni server di posta. Queste mail saranno in sospeso nella coda della posta. Il server di posta continuerà a tentare di inviare nuovamente l'e-mail. Una volta che l'invio dell'e-mail fallisce in modo permanente, il server di posta può inviare un messaggio e-mail di ritorno all'indirizzo e-mail del mittente.



E-Mail – SMTP comandi e significato

Command	Action taken
HELO	Accepts the greeting from the client and stores it in <code>seen_greeting</code> . Sets server to base command mode.
EHLO	Accepts the greeting from the client and stores it in <code>seen_greeting</code> . Sets server to extended command mode.
NOOP	Takes no action.
QUIT	Closes the connection cleanly.
MAIL	Accepts the “MAIL FROM:” syntax and stores the supplied address as <code>mailfrom</code> . In extended command mode, accepts the RFC 1870 SIZE attribute and responds appropriately based on the value of <code>data_size_limit</code> .
RCPT	Accepts the “RCPT TO:” syntax and stores the supplied addresses in the <code>rcpttos</code> list.
RSET	Resets the <code>mailfrom</code> , <code>rcpttos</code> , and <code>received_data</code> , but not the greeting.
DATA	Sets the internal state to DATA and stores remaining lines from the client in <code>received_data</code> until the terminator <code>"\r\n.\r\n"</code> is received.
HELP	Returns minimal information on command syntax
VERFY	Returns code 252 (the server doesn't know if the address is valid)
EXPN	Reports that the command is not implemented.



E-Mail – Esempio client SMTP

```
import smtplib
import email.utils
from email.mime.text import MIMEText

# Creiamo il messaggio mail
msg = MIMEText('Questo è il corpo del messaggio.')
msg['To'] = email.utils.formataddr(('DESTINATARIO', 'destinatario@example.com'))
msg['From'] = email.utils.formataddr(('MITTENTE', 'mittente@example.com'))
msg['Subject'] = 'Prova Invio Mail'

server = smtplib.SMTP('127.0.0.1', 1027)

server.set_debuglevel(True) # mostriamo le comunicazioni con il server
try:
    server.sendmail('mittente@example.com', ['destinatario@example.com'], msg.as_string())
finally:
    server.quit()
```



E-Mail – server SMTP

Documentazione la trovate: <https://docs.python.org/3/library/smtpd.html>

peer è l'indirizzo dell'host remoto

mailfrom è l'originatore della mail

rcpttos sono i destinatari della mail

```
from datetime import datetime
import asyncore
from smtpd import SMTPServer

class EmlServer(SMTPServer):
    no = 0
    def process_message(self, peer, mailfrom, rcpttos, data, mail_options=None, rcpt_options=None):
        filename = '%s-%d.eml' % (datetime.now().strftime('%Y%m%d%H%M%S'),
                                   self.no)
        f = open(filename, 'wb')
        f.write(data)
        f.close
        print ('%s saved.' % filename)
        self.no += 1

def run():
    EmlServer(('127.0.0.1', 1027), None)
    try:
        asyncore.loop()
    except KeyboardInterrupt:
        pass

if __name__ == '__main__':
    run()
```

data è una stringa contenente il contenuto dell'e-mail



E-Mail – esempio

- Lanciate il codice smtp_server.py
- Lanciate il codice smtp_client.py

Nella console del client vedremo il seguente scambio di pacchetti

```
send: 'ehlo 87.43.168.192.in-addr.arpa\r\n'
reply: b'250-87.43.168.192.in-addr.arpa\r\n'
reply: b'250-SIZE 33554432\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250 HELP\r\n'
reply: retcode (250); Msg: b'87.43.168.192.in-addr.arpa\nSIZE 33554432\n8BITMIME\nHELP'
send: 'mail FROM:<mittente@example.com> size=256\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'rcpt TO:<destinatario@example.com>\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
send: 'data\r\n'
reply: b'354 End data with <CR><LF>.<CR><LF>\r\n'
reply: retcode (354); Msg: b'End data with <CR><LF>.<CR><LF>'
data: (354, b'End data with <CR><LF>.<CR><LF>')
send: b'Content-Type: text/plain; charset="utf-8"\r\nMIME-Version: 1.0\r\nContent-
Transfer-Encoding: base64\r\nTo: DESTINATARIO <destinatario@example.com>\r\nFrom: MITTENTE
<mittente@example.com>\r\nSubject: Prova Invio Mail\r\n\r
\nUXVlc3RvIM0oIGlsIGNvcnBvIGRlbCBtZXNzYWdnaW8u\r\n.\r\n'
reply: b'250 OK\r\n'
reply: retcode (250); Msg: b'OK'
data: (250, b'OK')
send: 'quit\r\n'
reply: b'221 Bye\r\n'
reply: retcode (221); Msg: b'Bye'
```



SPEED TEST



Come funziona uno Speed Test

Quando si avvia un test di velocità, si verificano più cose.

Innanzitutto, il **client** determina la posizione e il server di test più vicino a voi: questa parte è importante, per evitare che i rallentamenti siano legati alla lunghezza del tragitto e non all'effettiva efficienza della connessione.



Alcune versioni, come *Speedtest.net* di **Ookla**, hanno un'opzione per scegliere il server. Speed Test invia un semplice segnale (ping) al server il quale risponderà. Il test misura quel round trip in millisecondi.

Al termine del ping, inizia il test di download. Il client apre più connessioni al server e tenta di scaricare una piccola porzione di dati. A questo punto, vengono misurate due cose: **il tempo impiegato per acquisire il frammento di dati e la quantità di risorse di rete utilizzate.**



Come funziona uno Speed Test

Se il client rileva che è ha risorse disponibili, apre più connessioni al server e scarica più dati.

L'idea generale è di caricare la connessione Internet e vedere quanto può trasportare simultaneamente.

Immaginate il servizio Internet come un'autostrada con un limite di velocità.

Aprire connessioni aggiuntive è come aggiungere più corsie all'autostrada.

Il limite di velocità non è cambiato, ma più auto possono attraversare lo stesso spazio a una velocità maggiore;

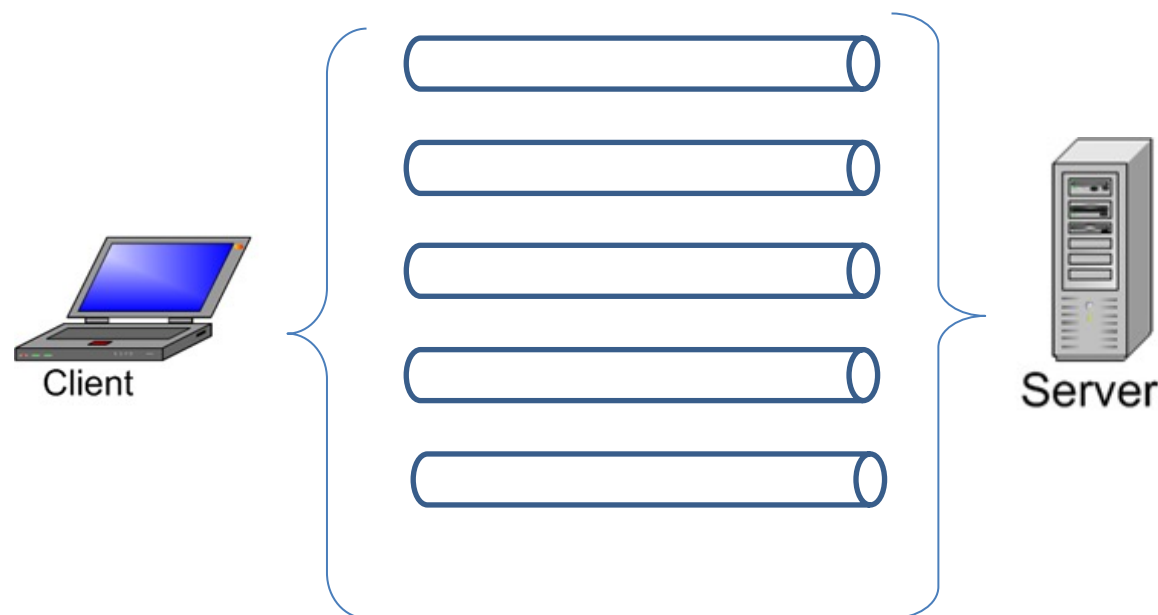
quindi, la 50^a auto arriverà prima usando un'autostrada a quattro corsie rispetto a quella su una a due corsie.



Come funziona uno Speed Test

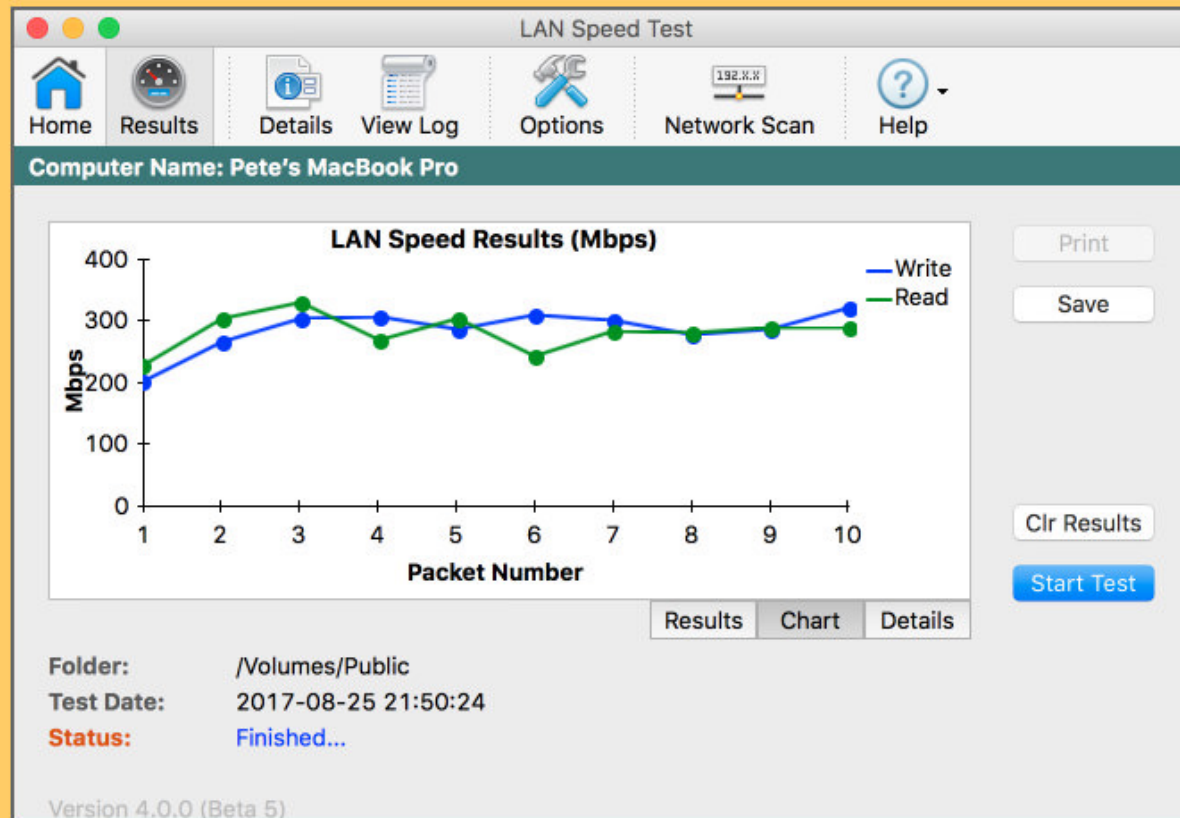
Una volta che il **client** ha accertato che dispone delle connessioni per testare il servizio Internet, scarica ulteriori blocchi di dati, misura la quantità scaricata nel tempo assegnato e presenta **una velocità di download**.

Il test di upload è essenzialmente lo stesso processo del test di download ma al contrario. Invece di estrarre i dati dal server al PC, il client carica i dati dal PC al server.



Calcolo Speed Test

Nell'esempio che segue, effettueremo solo una connessione come esempio, ovviamente è possibile moltiplicare quanto vedremo su più connessioni in parallelo per avere una idea più precisa dell'effettivo Throughput disponibile.




Speed Test in Python



Speed Test in Python


```
import sys, time
from socket import *
```

```
MY_PORT = 50000 + 42
BUFSIZE = 1024
```



```
def main():
    if len(sys.argv) < 2:
        usage()
    if sys.argv[1] == '-s':
        server()
    elif sys.argv[1] == '-c':
        client()
    else:
        usage()

def usage():
    sys.stdout = sys.stderr
    print ('Usage:      (on host_A) throughput -s [port]')
    print ('and then: (on host_B) throughput -c count host_A [port]')
    sys.exit(2)
```



Definiamo la porta

Definiamo la dimensione del Buffer

Creiamo due funzioni una per la gestione degli input da riga di comando finalizzata alla esecuzione dello script in modalità server o client e l'altra per la gestione di errori in fase di input

Si ricordi che sys.argv è un vettore di stringhe → sys.argv=[]



Speed Test Lato Server



Speed Test Lato Server

```
def server():
    if len(sys.argv) > 2:
        port = eval(sys.argv[2])
    else:
        port = MY_PORT
    s = socket(AF_INET, SOCK_STREAM)
    s.bind(('', port))
    s.listen(1)
    print ('Server ready...')
    while 1:
        conn, (host, remoteport) = s.accept()
        while 1:
            data = conn.recv(BUFSIZE)
            if not data:
                break
            del data
            conn.send(('OK\n').encode())
            conn.close()
        print ('Done with', host, 'port', remoteport)
```

- Lato server dobbiamo ovviamente definire la porta di ascolto
 - La funzione ***eval()*** serve a trasformare il valore di ingresso (considerato stringa) in un valore intero
 - Se non assegno una porta in fase di esecuzione del comando, lo script prende quella dichiarata nella variabile **MY_PORT**
- Se non arrivano dati sul socket lo script termina, altrimenti cicla assegnando in ogni ciclo alla variabile data il contenuto del buffer, che poi svuota prima di rimettersi in ascolto.



Speed Test Lato Client

```
def client():
    if len(sys.argv) < 4:
        usage()
    count = int(eval(sys.argv[2]))
    host = sys.argv[3]
    if len(sys.argv) > 4:
        port = eval(sys.argv[4])
    else:
        port = MY_PORT
    testdata = 'x' * (BUFSIZE-1) + '\n'
    t1 = time.time()
    s = socket(AF_INET, SOCK_STREAM)
    t2 = time.time()
    s.connect((host, port))
    t3 = time.time()
    i = 0
    while i < count:
        i = i+1
        s.send(testdata.encode())
    s.shutdown(1) # Send EOF
    t4 = time.time()
    data = s.recv(BUFSIZE)
    t5 = time.time()
    print (data)
    print ('Raw timers:', t1, t2, t3, t4, t5)
    print ('Intervals:', t2-t1, t3-t2, t4-t3, t5-t4)
    print ('Total:', t5-t1)
    print ('Throughput:', round((BUFSIZE*count*0.001) / (t5-t1), 3), 'K/sec.')
```

- Se la lunghezza del vettore sys.argv è inferiore a 4 significa che mancano informazioni in ingresso ossia il numero di pacchetti da inviare e l'indirizzo e porta del server
- Se la porta viene indicata verrà selezionata quella, altrimenti verrà aperta la connessione verso quella definita nello script
- Quindi il client «riempie» il buffer di 1024 byte con una stringa di tutte x e un «a capo» finale (dummy burst) → xxxxxx....+'\n'

- E Comincia ad inviare i dummy burst verso il server. Quando il contatore ha terminato, invia l'EOF.



Calcolo del THROUGHPUT

All'istante t_5 il Client ha terminato l'invio di tutti burst ed esegue il calcolo seguente:
Il tempo totale di Trasmissione è:

$$T = t_5 - t_1$$

Quantità di **bytes** trasmessi complessivamente:

$$Bytes_{tot} = n^o pacchetti\ inviati \times 1024 bytes = COUNT \times 1024 bytes$$

$$Throughput\ in\ \frac{Kbyte}{sec} = \frac{Bytes_{tot}}{T} = \frac{COUNT \times 1024}{T} \times 0.001 \left[\frac{kB}{s} \right]$$



Calcolo del RTT (Round Trip Time)

Per calcolare il Round Trip Time possiamo prendere l'istante di invio dell' EOF (t_4) da parte del client e l'istante (t_5) in cui il client riceve il messaggio OK dal server

$$RTT = t_5 - t_4 [s]$$

```
Linux ubuntunet2008 2.6.35-32-generic-pae #68-Ubuntu SMP Tue Mar 27 18:04:42 UTC 2012 i686 GNU/Linux
Ubuntu 10.10

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

New release 'natty' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Apr 24 15:51:20 2020 from 192.168.1.141
apirodd@ubuntunet2008:~$ python BandwidthClient_Server.py -c 10000 192.168.1.141 11000
OK

Raw timers: 1587743148.59 1587743148.59 1587743148.59 1587743155.7 1587743155.74
Intervals: 4.50611114502e-05 0.00301384925842 7.10320401192 0.0448861122131
Total: 7.1511490345
Throughput: 1431.938 K/sec.
```



Risultati

Nel nostro esempio abbiamo ottenuto un **Throughput** di quasi

1.5MB/s (ossia circa 12Mb/s)

E un

$$RTT = 1587743155.74 - 1587743155.7 = 0.0448 \text{ [sec]} \cong \mathbf{45ms}$$



Wireshark per tracciare il transito dei pacchetti

The image shows a Wireshark packet capture window titled 'speed_test.pcapng'. The packet list pane on the left shows several packets. A green box highlights packets 84 through 97. A text box on the right says 'Pacchetti di instaurazione del tunnel'. The packet details pane shows the structure of packet 84: Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and NetBIOS Session Service / SMB (Server Message Block Protocol). The packet bytes pane shows the raw data of packet 84.

No.	Time	Source	Destination	Protocol	Length	Info
82	4.608255	192.168.1.144	224.0.0.251	MDNS	87	Standard query 0x0000 PTR _spotify-connect._tcp.local "OU" question
83	4.610104	192.168.1.144	239.255.255.250	SSDP	167	M-SEARCH * HTTP/1.1
84	5.160947	192.168.1.141	192.168.1.117	SMB	107	Echo Request
85	5.172713	192.168.1.117	192.168.1.141	SMB	107	Echo Response
86	5.212528	192.168.1.141	192.168.1.117	TCP	54	55391 → 445 [ACK] Seq=54 Ack=54 Win=4559 Len=0
87	5.610180	192.168.1.141	192.168.1.117	SSH	94	Client: Encrypted packet (len=40)
88	5.625852	192.168.1.117	192.168.1.141	SSH	94	Server: Encrypted packet (len=40)
89	5.652843	192.168.1.117	192.168.1.141	TCP	74	60520 → 11000 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=1164635391 TSecr=0 WS=64
90	5.653087	192.168.1.141	192.168.1.117	TCP	66	11000 → 60520 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
91	5.654331	192.168.1.117	192.168.1.141	TCP	60	60520 → 11000 [ACK] Seq=1 Ack=1 Win=5888 Len=0
92	5.654914	192.168.1.117	192.168.1.141	TCP	1078	60520 → 11000 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=1024
93	5.655583	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=1025 Ack=1 Win=5888 Len=1460
94	5.655585	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=2485 Ack=1 Win=5888 Len=1460
95	5.655709	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=3945 Win=65536 Len=0
96	5.657365	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=3945 Ack=1 Win=5888 Len=1460
97	5.657475	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=5405 Win=65536 Len=0

> Frame 84: 107 bytes on wire (856 bits), 107 bytes captured (856 bits) on interface \Device\NPF_{A2E3F05E-2636-424E-BC62-F71E17BA3406}, id 0
> Ethernet II, Src: IntelCor_53:0e:5d (b4:d5:bd:53:0e:5d), Dst: HewlettP_18:87:50 (1c:c1:de:18:87:50)
> Internet Protocol Version 4, Src: 192.168.1.141, Dst: 192.168.1.117
> Transmission Control Protocol, Src Port: 55391, Dst Port: 445, Seq: 1, Ack: 1, Len: 53
> NetBIOS Session Service
> SMB (Server Message Block Protocol)

0000 1c c1 de 18 87 50 b4 d5 bd 53 0e 5d 08 00 45 00P...S.]..E-
0010 00 5d b8 18 40 00 00 06 be 2f c0 a8 01 8d c0 a8 .].@.../.....
0020 01 75 d8 5f 01 bd 20 b8 c3 6e 9b 54 28 26 50 18 .u....n-T(&P-
0030 11 cf ba 35 00 00 00 00 00 31 ff 53 4d 42 2b 00 ...5....1SMB+
0040 00 00 00 18 43 c0 00 00 00 00 00 00 00 00 00C.....
0050 00 00 ff ff ff fe 00 00 fe ff 01 01 00 0c 00 4a]..
0060 6c 4a 6d 49 68 43 6c 42 73 72 00 00 00 00 00 00 1JmIhClB sr..

Come si vede nella figura al lato la connessione tra il client

192.168.1.117

(speedtester) e il server

192.168.1.141

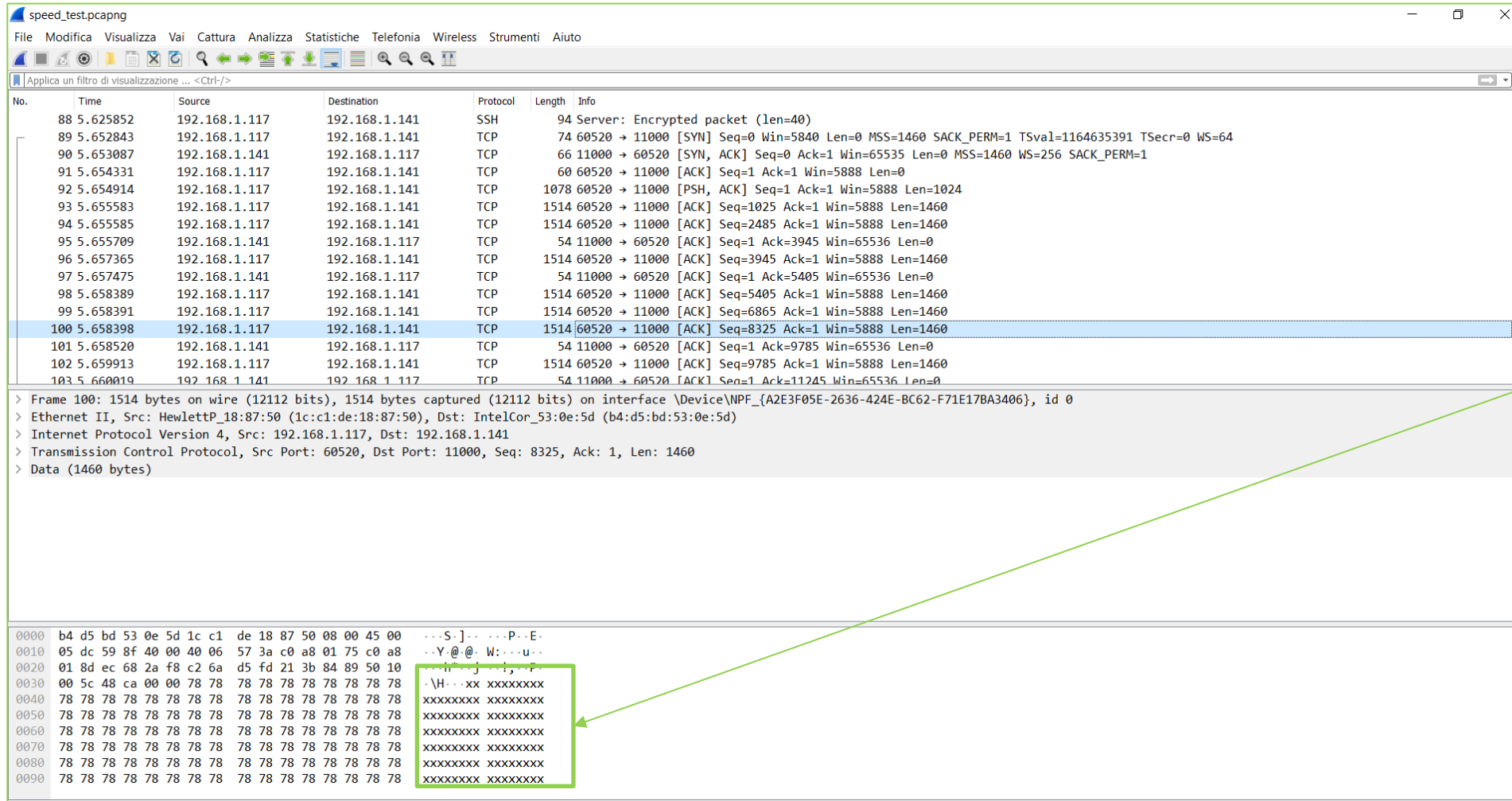
(speedtester)

avviene tramite protocollo SSH

Si ricordi che nel nostro speed test è il client ad inviare i pacchetti verso il server



Wireshark per tracciare il transito dei pacchetti



speed_test.pcapng

File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonica Wireless Strumenti Aiuto

Applica un filtro di visualizzazione ... <Ctrl-F>

No.	Time	Source	Destination	Protocol	Length	Info
88	5.625852	192.168.1.117	192.168.1.141	SSH	94	Server: Encrypted packet (len=40)
89	5.652843	192.168.1.117	192.168.1.141	TCP	74	60520 → 11000 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=1164635391 TSecr=0 WS=64
90	5.653087	192.168.1.141	192.168.1.117	TCP	66	11000 → 60520 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
91	5.654331	192.168.1.117	192.168.1.141	TCP	60	60520 → 11000 [ACK] Seq=1 Ack=1 Win=5888 Len=0
92	5.654914	192.168.1.117	192.168.1.141	TCP	1078	60520 → 11000 [PSH, ACK] Seq=1 Ack=1 Win=5888 Len=1024
93	5.655583	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=1025 Ack=1 Win=5888 Len=1460
94	5.655585	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=2485 Ack=1 Win=5888 Len=1460
95	5.655709	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=3945 Win=65536 Len=0
96	5.657365	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=3945 Ack=1 Win=5888 Len=1460
97	5.657475	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=5405 Win=65536 Len=0
98	5.658389	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=5405 Ack=1 Win=5888 Len=1460
99	5.658391	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=6865 Ack=1 Win=5888 Len=1460
100	5.658398	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=8325 Ack=1 Win=5888 Len=1460
101	5.658520	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=9785 Win=65536 Len=0
102	5.659913	192.168.1.117	192.168.1.141	TCP	1514	60520 → 11000 [ACK] Seq=9785 Ack=1 Win=5888 Len=1460
103	5.660019	192.168.1.141	192.168.1.117	TCP	54	11000 → 60520 [ACK] Seq=1 Ack=11245 Win=65536 Len=0

> Frame 100: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{A2E3F05E-2636-424E-BC62-F71E17BA3406}, id 0

> Ethernet II, Src: HewlettP_18:87:50 (1c:c1:de:18:87:50), Dst: IntelCor_53:0e:5d (b4:d5:bd:53:0e:5d)

> Internet Protocol Version 4, Src: 192.168.1.117, Dst: 192.168.1.141

> Transmission Control Protocol, Src Port: 60520, Dst Port: 11000, Seq: 8325, Ack: 1, Len: 1460

> Data (1460 bytes)

0000 b4 d5 bd 53 0e 5d 1c c1 de 18 87 50 08 00 45 00 ...S-]-...P-E-
0010 05 dc 59 8f 40 00 40 06 57 3a c0 a8 01 75 c0 a8 ...Y.@-W:...u-
0020 01 8d ec 68 2a f8 c2 6a d5 fd 21 3b 84 89 50 10 ...h-j-...P-
0030 00 5c 48 ca 00 00 78 78 78 78 78 78 78 78 78 ...H-...xx xxxxxxxx
0040 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx
0050 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx
0060 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx
0070 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx
0080 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx
0090 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxx xxxxxxxx

Si vede il contenuto dei pacchetti inviati dal client.

Nella cartella del Laboratorio è presente il file di tracciamento di Wireshark



ARP



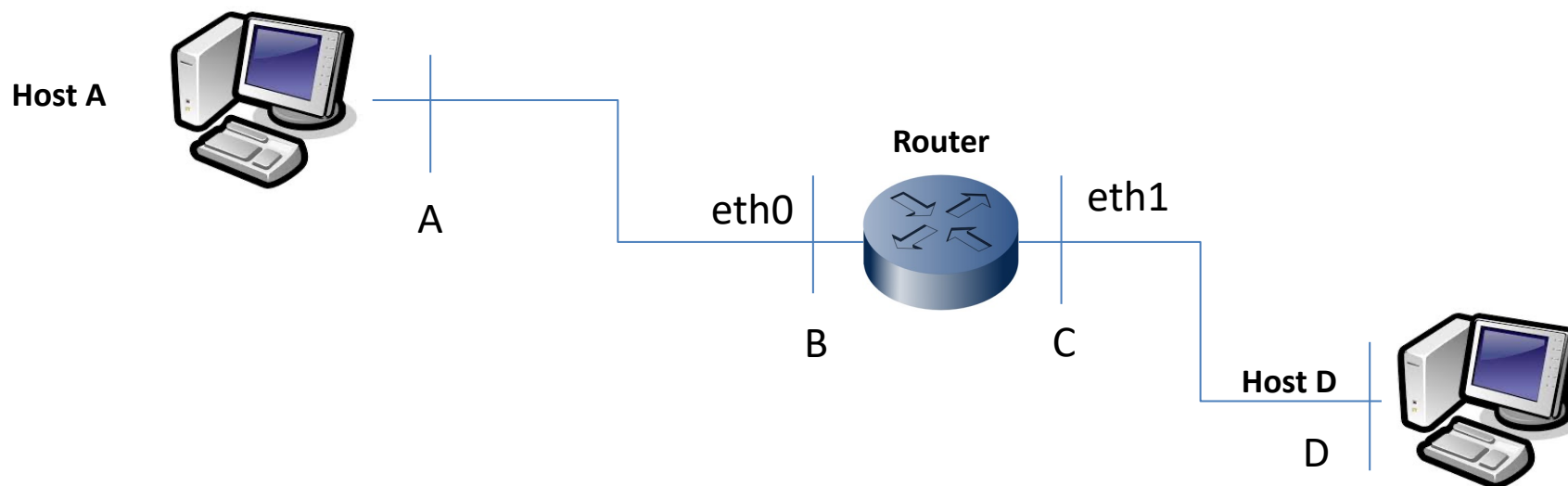
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Esercizio 4



Esercizio 4 – Tabella ARP

Si consideri la rete in figura dove le interfacce sono identificate con lettere maiuscole



Si indichino con IP-x e MAC-x, con $x=[A,B,C,D]$, gli indirizzi IP e ethernet delle interfacce



Esercizio 4 – Tabella ARP

Si supponga che le ARP table di A, di B e del router siano vuote

L'host A deve inviare un pacchetto IP verso l'indirizzo IP-D.

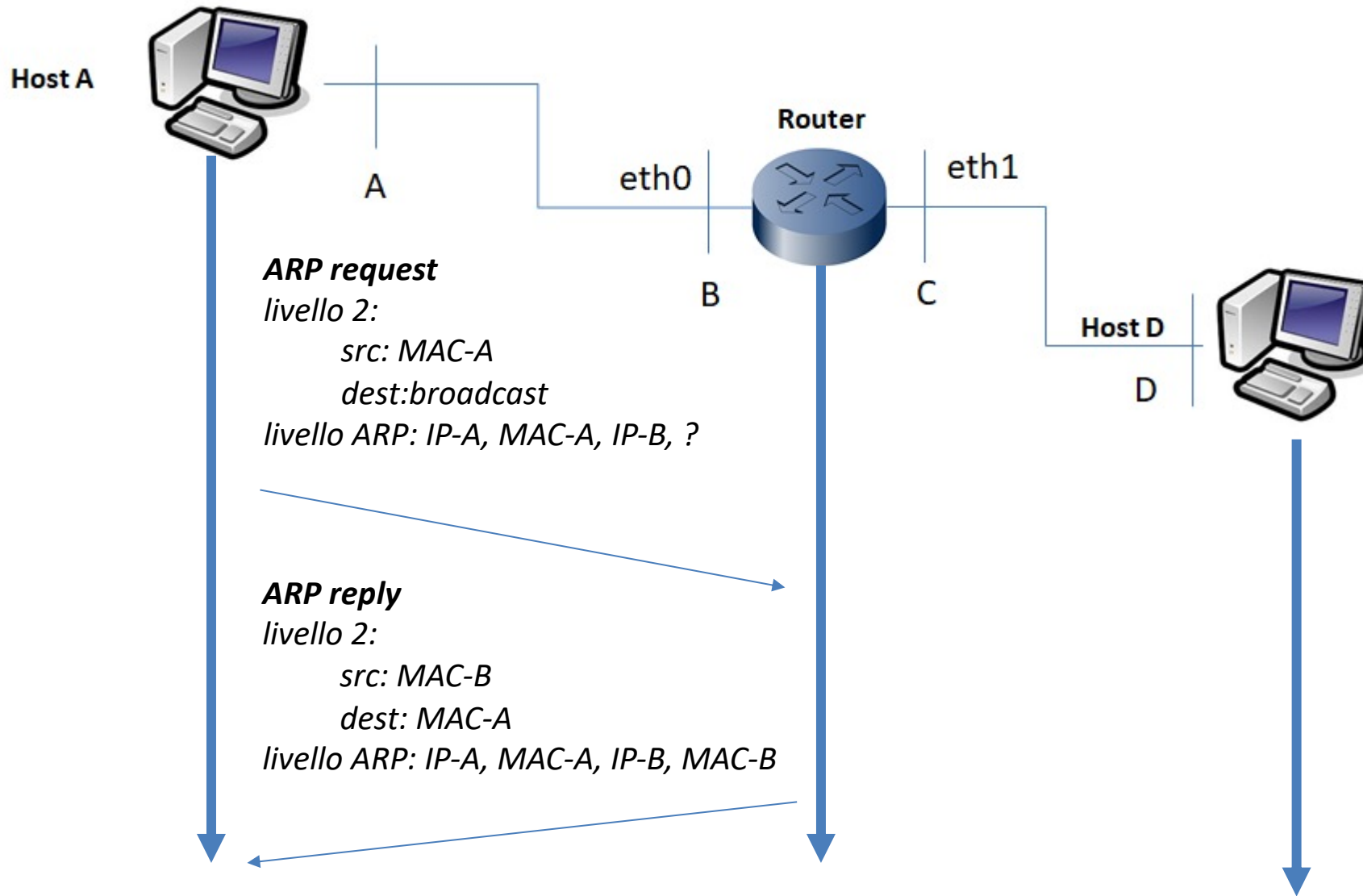
Ricevuto il pacchetto l'host D deve inviare un pacchetto di risposta verso A.

Quesiti

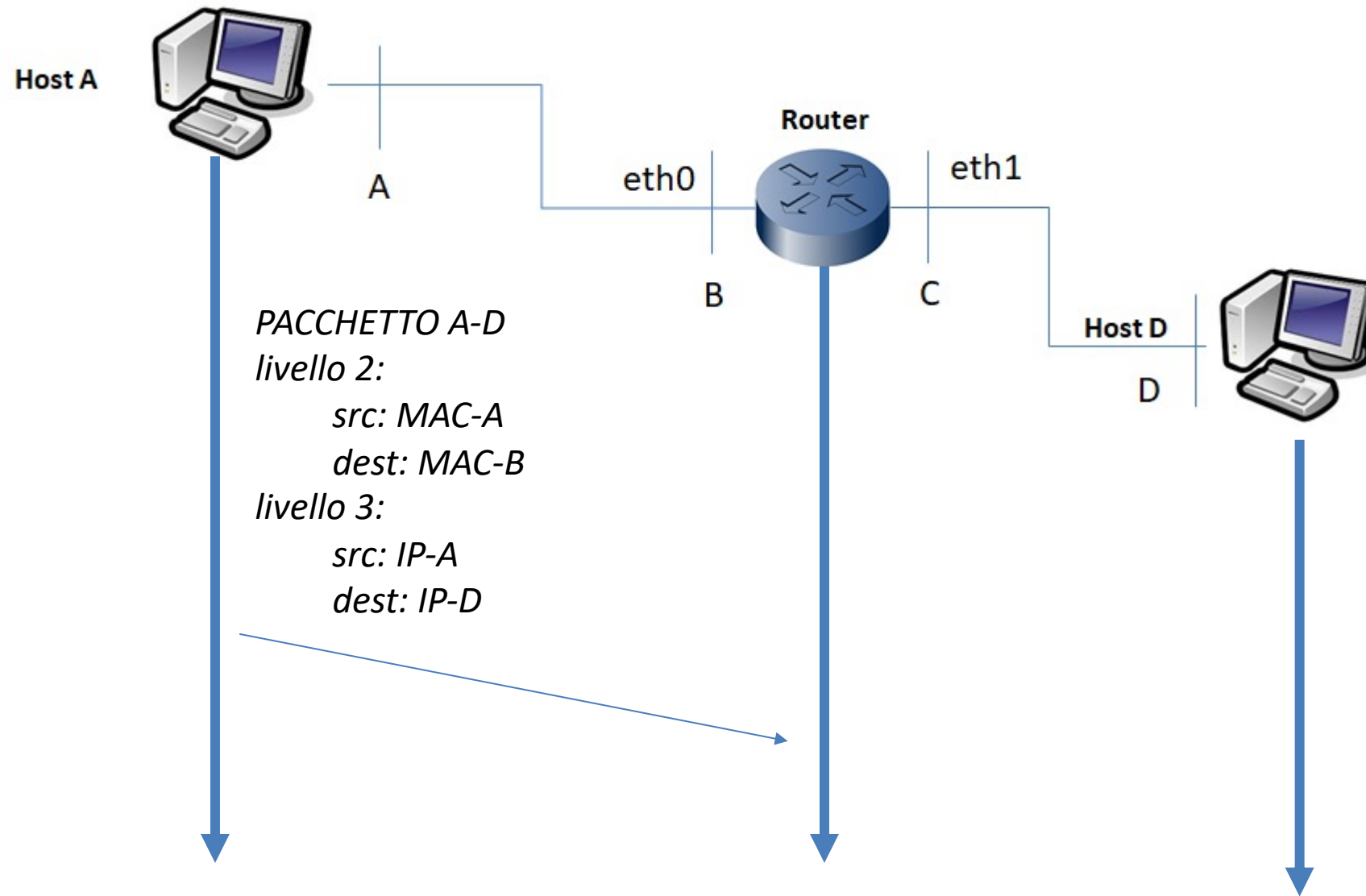
1. si indichino graficamente i pacchetti che vengono trasmessi e per ciascuno di essi (su ognuna delle reti ethernet attraversate) gli indirizzi contenuti nelle PDU di livello 2 (ethernet) e 3 (IP o ARP)



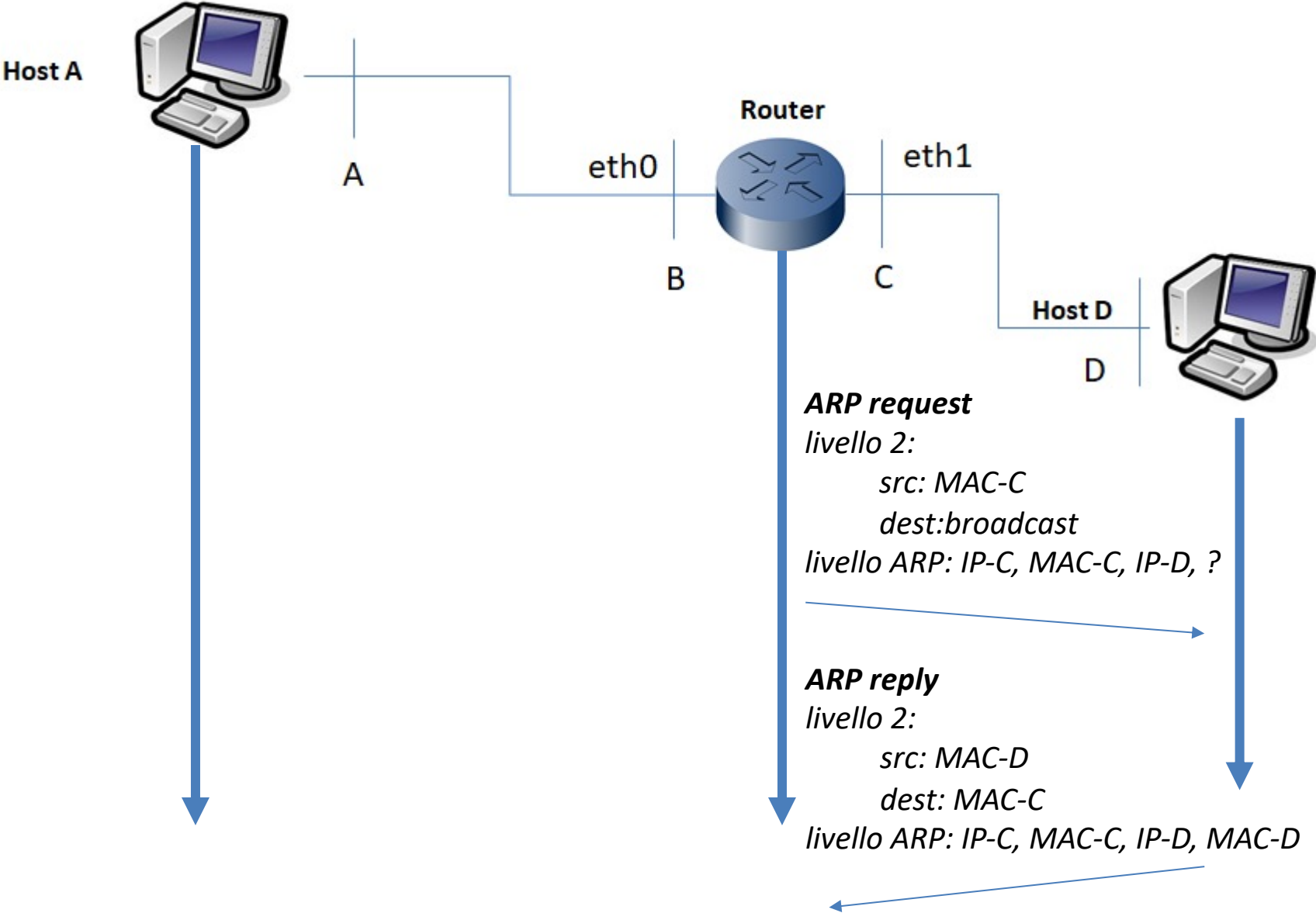
Esercizio 4 – Tabella ARP - soluzione



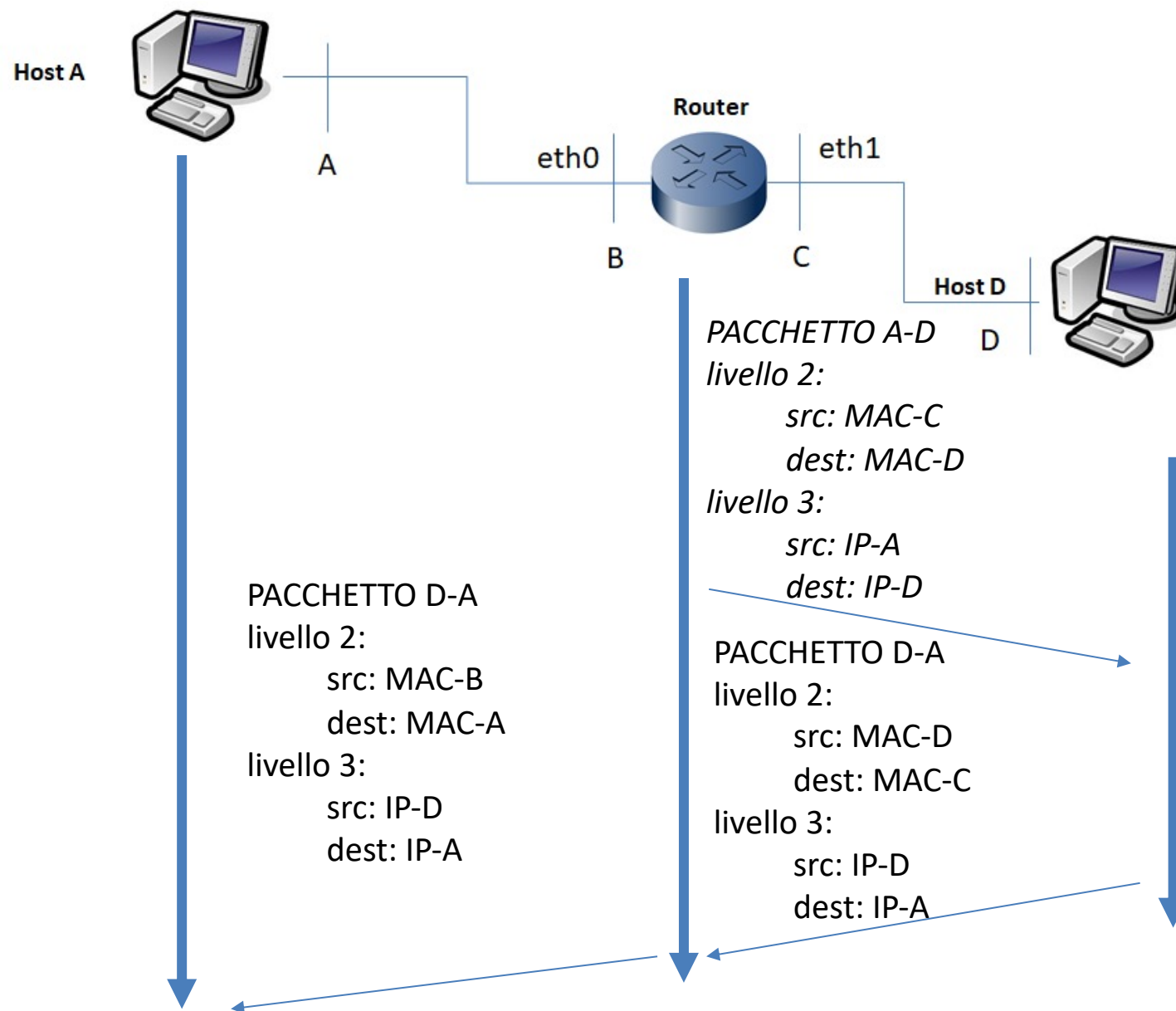
Esercizio 4 – Tabella ARP - soluzione



Esercizio 4 – Tabella ARP - soluzione



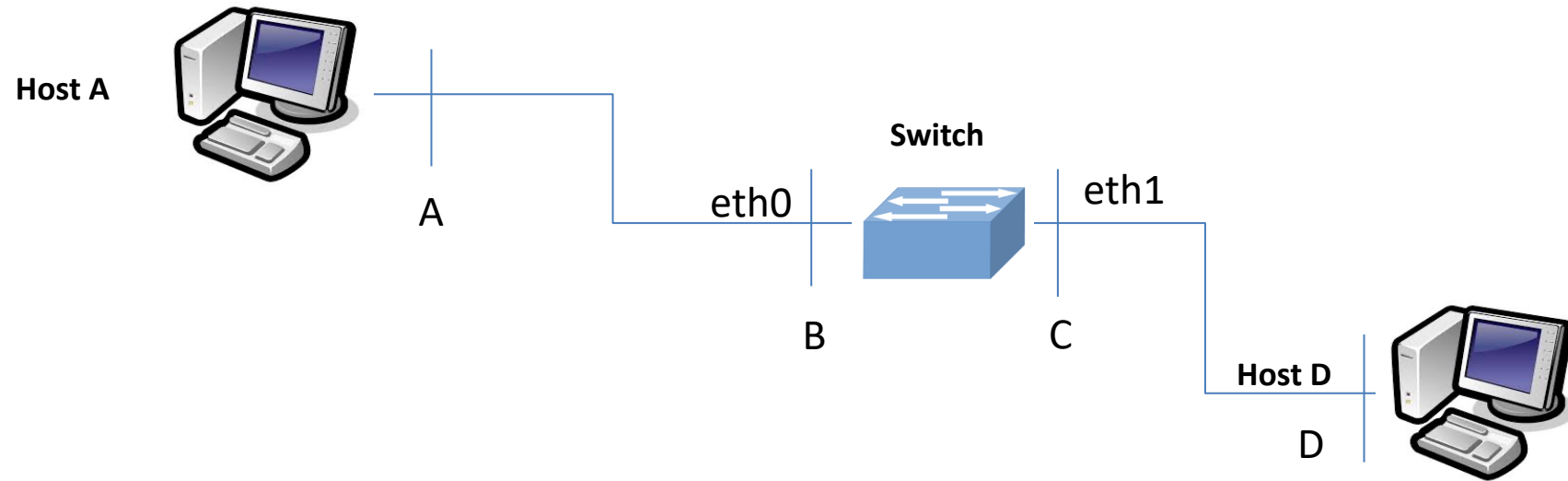
Esercizio 4 – Tabella ARP - soluzione



Esercizio 5



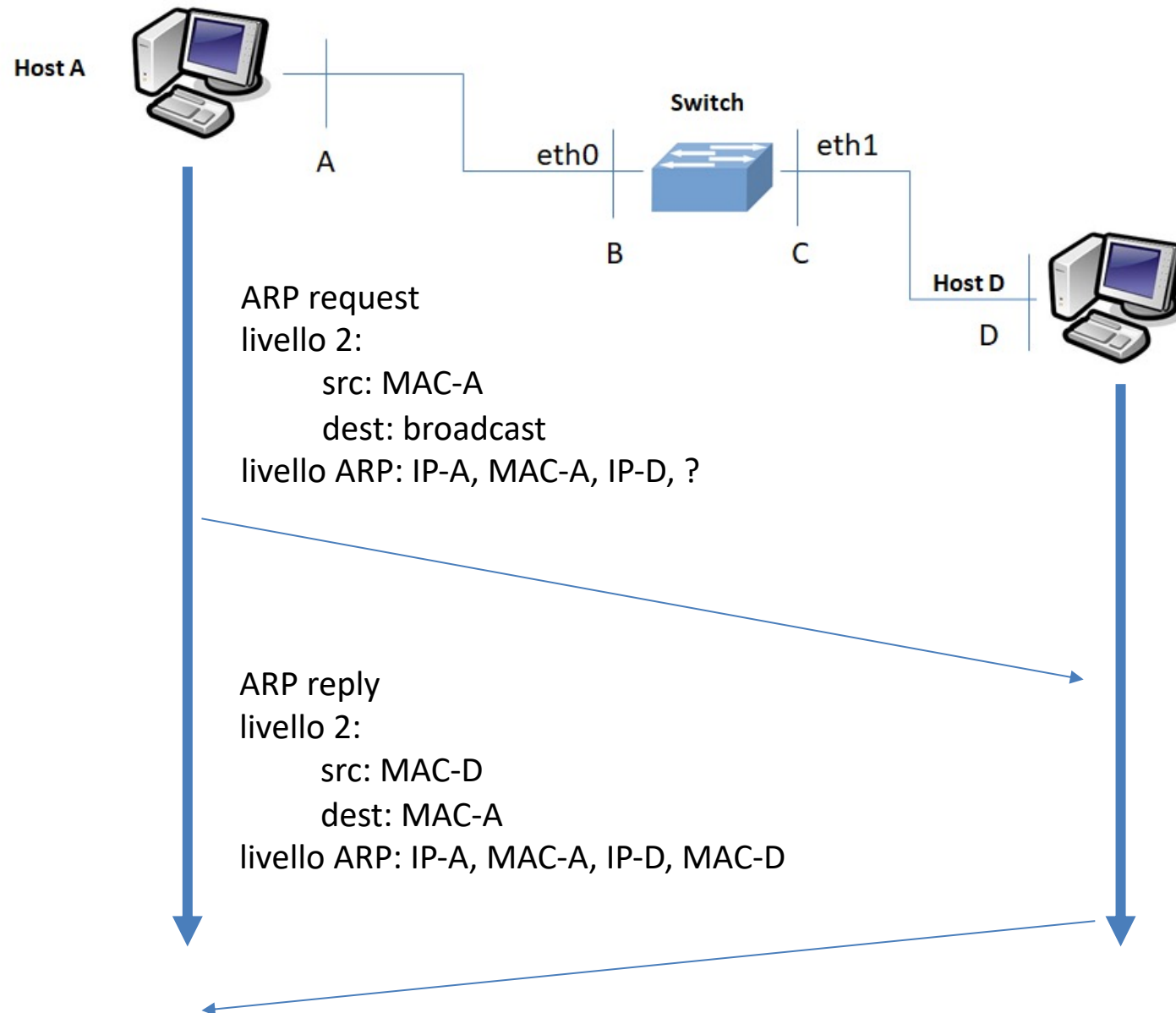
Esercizio 5 – Tabella ARP



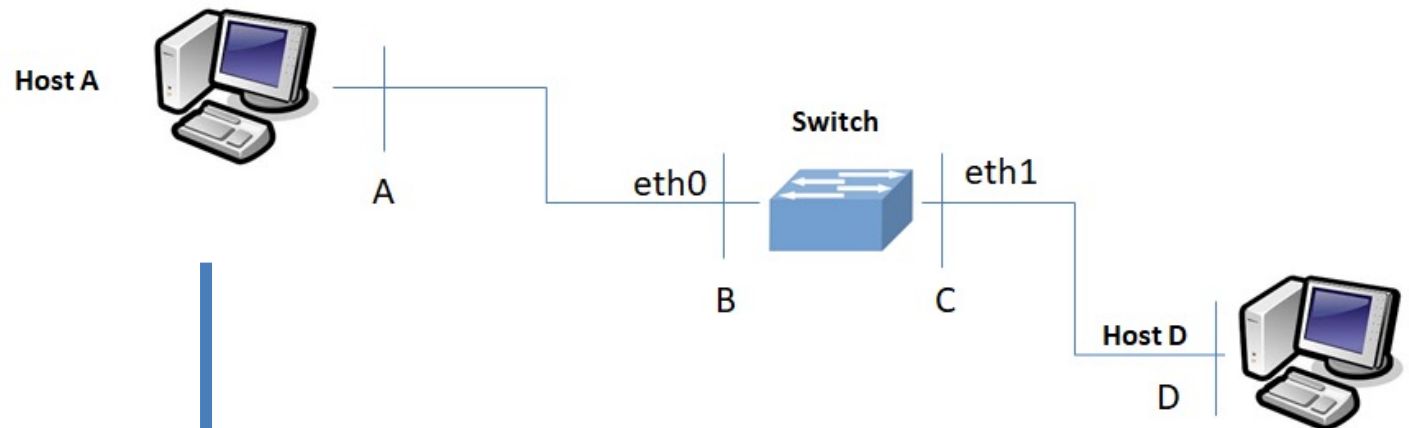
Lo stesso nel caso in cui il nodo al centro sia un bridge (layer 2 switch)



Esercizio 5 – Tabella ARP - soluzione



Esercizio 5 – Tabella ARP - soluzione



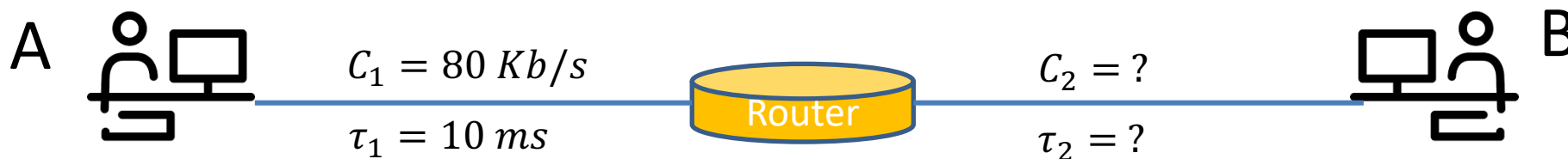
PACCHETTO A-D
livello 2:
src: MAC-A
dest: MAC-D
livello 3:
src: IP-A
dest: IP-D

PACCHETTO D-A
livello 2:
src: MAC-D
dest: MAC-A
livello 3:
src: IP-D
dest: IP-A



Esercizio 2

Si consideri il collegamento in figura



QUESITO

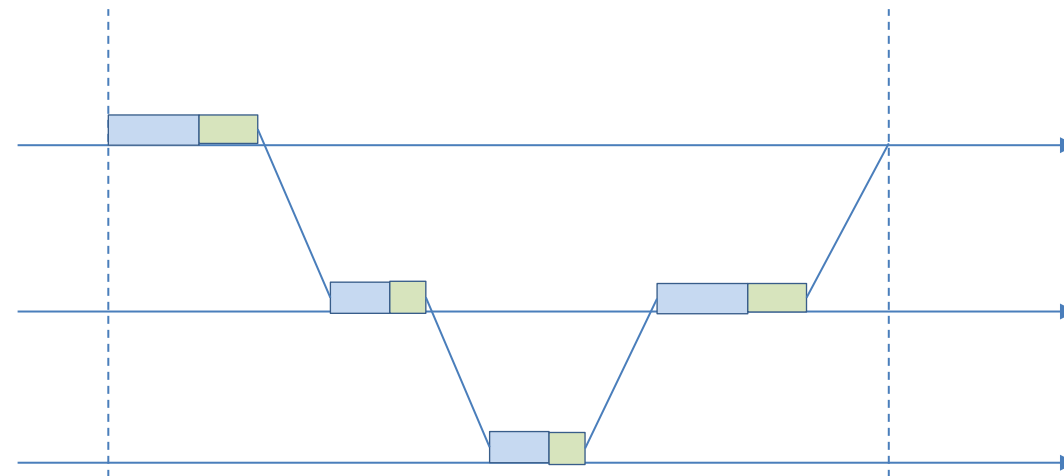
- A vuole conoscere la capacità e il ritardo di propagazione del link 2 e allo scopo invia a B 2 messaggi di echo: M_1 di lunghezza $L_1 = 1000 \text{ [byte]}$, ed M_2 di lunghezza $L_2 = 1500 \text{ [byte]}$; per ognuno di essi misura un Round-Trip-Time (RTT) pari a 780 [ms] e 1130 [ms] rispettivamente. Nella risposta, B utilizza messaggi con le stesse lunghezze. Calcolare C_2 e τ_2 nell'ipotesi che le lunghezze degli header siano trascurabili.



Soluzione Esercizio 2 – Quesito 1

$$RTT_1 = 2 \cdot \left(\frac{L_1}{c_1} + \frac{L_1}{c_2} + \tau_1 + \tau_2 \right) = 780 [ms]$$

$$RTT_2 = 2 \cdot \left(\frac{L_2}{c_1} + \frac{L_2}{c_2} + \tau_1 + \tau_2 \right) = 1130 [ms]$$



Da cui si ricava che

$$\frac{780 [ms]}{2} - \left(\frac{L_1}{c_1} + \frac{L_1}{c_2} \right) = (\tau_1 + \tau_2) \text{ sostituendo nella seconda equazione}$$

$$\frac{1130 [ms]}{2} = \left(\frac{L_2}{c_1} + \frac{L_2}{c_2} \right) + \frac{780 [ms]}{2} - \left(\frac{L_1}{c_1} + \frac{L_1}{c_2} \right) = \left(\frac{12000 [bit]}{80 \cdot 10^3 [\frac{bit}{s}]} + \frac{12000 [bit]}{c_2} \right) + \frac{780 [ms]}{2} - \left(\frac{8000 [bit]}{80 \cdot 10^3 [\frac{bit}{s}]} + \frac{8000 [bit]}{c_2} \right)$$

$$175 [ms] = 50 [ms] + \frac{4000 [bit]}{c_2}$$

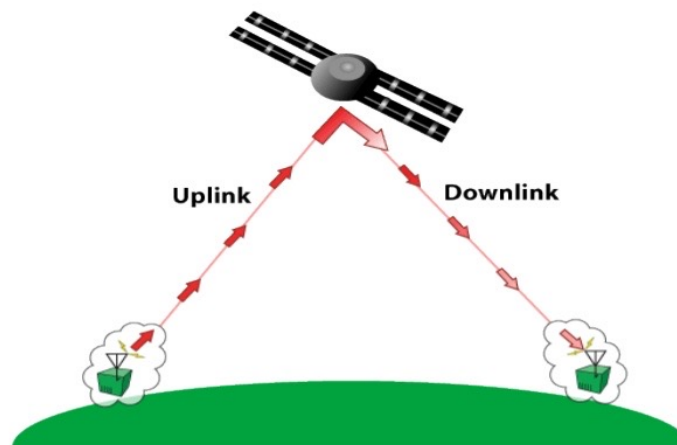
$$c_2 = \frac{4000 [bit]}{125 [ms]} = \frac{4000 [bit]}{125 \cdot 10^{-3} [s]} = \mathbf{32 [\frac{Kb}{s}]}$$

$$\tau_2 = \frac{780 [ms]}{2} - \left(\frac{L_1}{c_1} + \frac{L_1}{c_2} \right) - \tau_1 = 390 [ms] - 100 [ms] - 250 [ms] - 10 [ms] = \mathbf{30 [ms]}$$



Esercizio 3

Si consideri un canale via satellite della capacità di 1 [Mb/s] e che il tempo di propagazione attraverso il satellite geostazionario richieda 250 [ms] . Si suppongano gli ACK trascurabili.

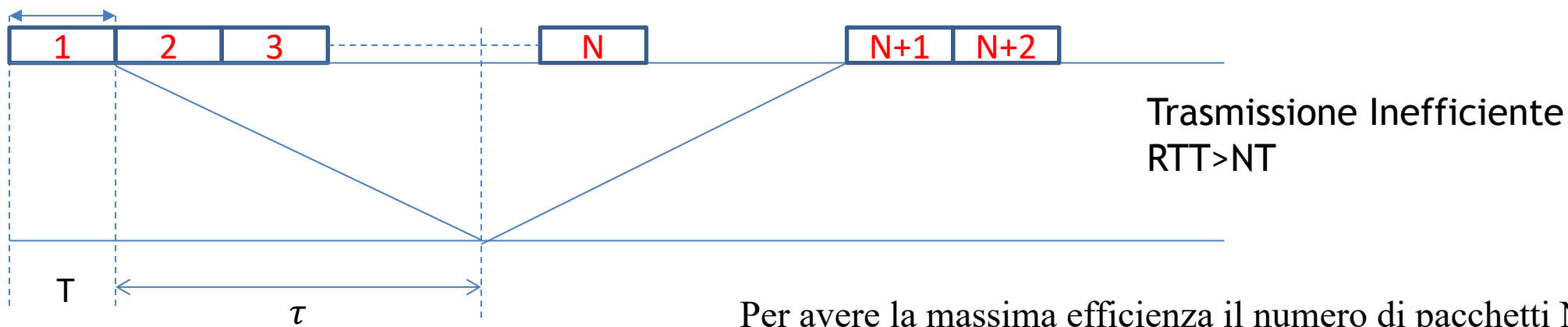


- Si chiede di dimensionare la minima finestra di trasmissione di un protocollo Go-BACK-N (con timeout) in modo che sia consentita la massima efficienza temporale del canale quando vengano trasmesse trame di 2000 [bit] in assenza di errori.
- Si calcoli poi la massima efficienza trasmissiva che si avrebbe nel caso in cui il meccanismo ARQ fosse di tipo STOP and WAIT

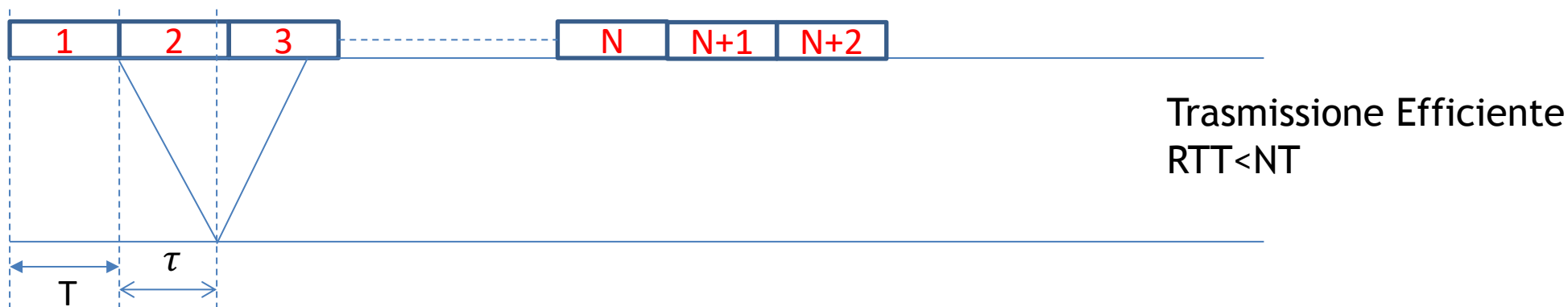


Soluzione Esercizio 3 – Quesito 1

L'efficienza del meccanismo Go-BACK-N dipende dal rapporto tra RTT e lunghezza della finestra.



Per avere la massima efficienza il numero di pacchetti N nella finestra deve essere tale che il loro tempo di trasmissione copra il tempo di andata e ritorno del primo pacchetto



Soluzione Esercizio 3 – Quesito 1

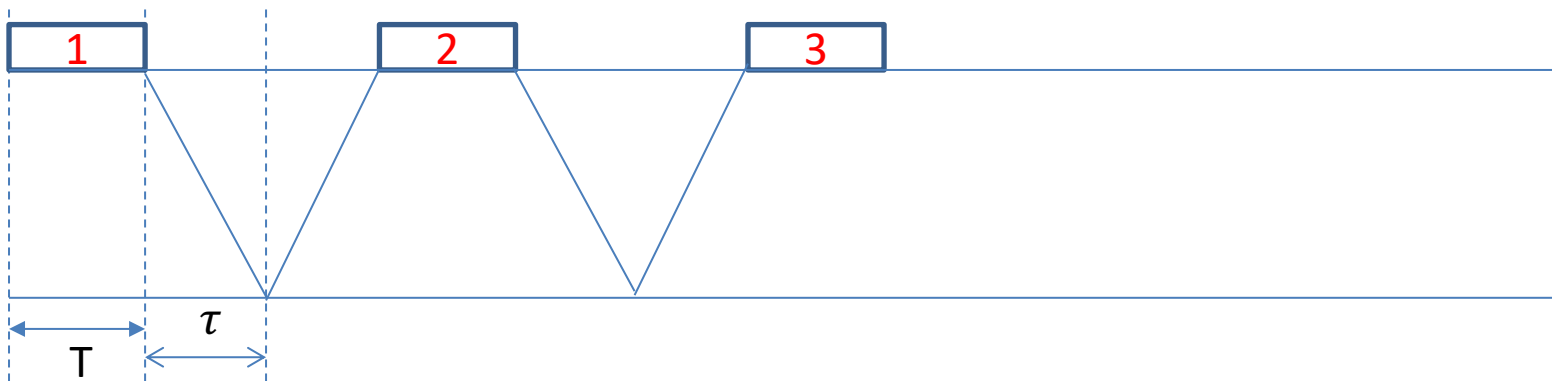
Detti:

- $T = 2 \text{ [ms]} = 2000 \text{ [bit]} / 1 \text{ [Mb/s]}$, il tempo di trasmissione di un pacchetto
- $t = 250 \text{ [ms]}$, il tempo di propagazione

allora deve essere: $NT \geq T + 2t \rightarrow N > 1 + \frac{2t}{T} = 1 + 2 \cdot \frac{250}{2} = 251$

Per finestre $N \geq 251$, la trasmissione risulta continua, dunque l'efficienza del meccanismo è 1.

Nel caso di Stop & Wait abbiamo:



L'efficienza dello Stop & Wait si calcola considerando che il protocollo trasmette 1 pacchetto (durata $T=2 \text{ [ms]}$)

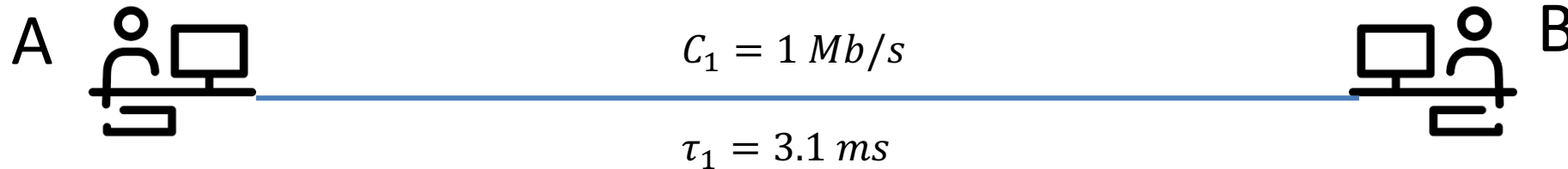
ogni $T+2t$, dunque l'efficienza: $\eta = \frac{T}{(T+2t)} = \frac{1}{251}$



Esercizio 4

Si consideri il collegamento in figura.

L'Host A deve trasferire tramite canale TCP una sequenza di 100 segmenti di lunghezza massima.



QUESITO

Si calcoli il tempo necessario, supponendo che:

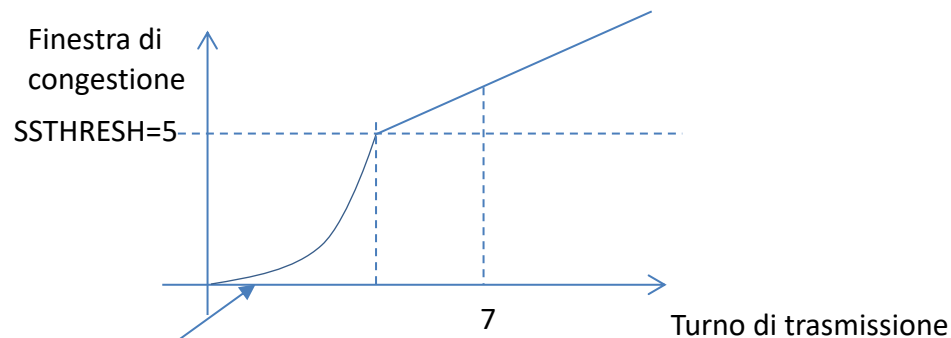
- $MSS = 1000 \text{ [bit]}$
- Lunghezza degli Header di tutti i livelli trascurabile
- La connessione viene aperta da A e la lunghezza dei segmenti di apertura della connessione è trascurabile
- La lunghezza degli ACK è trascurabile
- $SSTHRESH$ è pari a 5 MSS



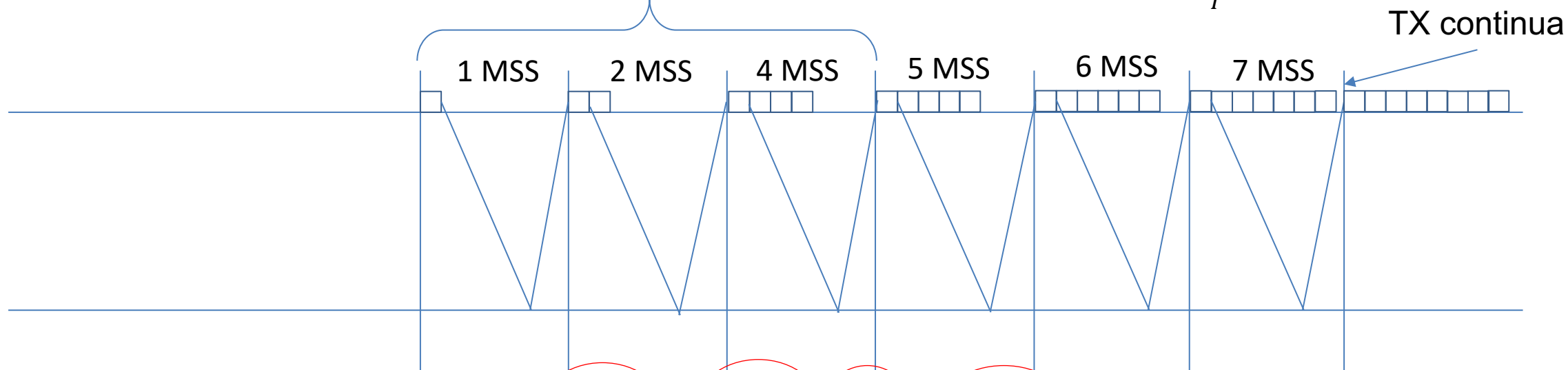
Soluzione Esercizio 4 – Quesito 1

$$T = \frac{1000 [bit]}{1 [\frac{Mb}{s}]} = 1 [ms]$$

$$RTT = 2 \cdot \tau_1 + T = 6.2 [ms] + 1 [ms]$$



La trasmissione è discontinua fino a che $W \cdot T < RTT$, cioè fino a che $W < \frac{RTT}{T} \approx 8$



$$T_{transfer} = 6.2 [ms] + 6 RTT + 75 \cdot T + 6.2 [ms] = 130.6 [ms]$$

Set up
(abbiamo
ipotizzato che
la lunghezza
dei segmenti
di apertura sia
trascurabile)

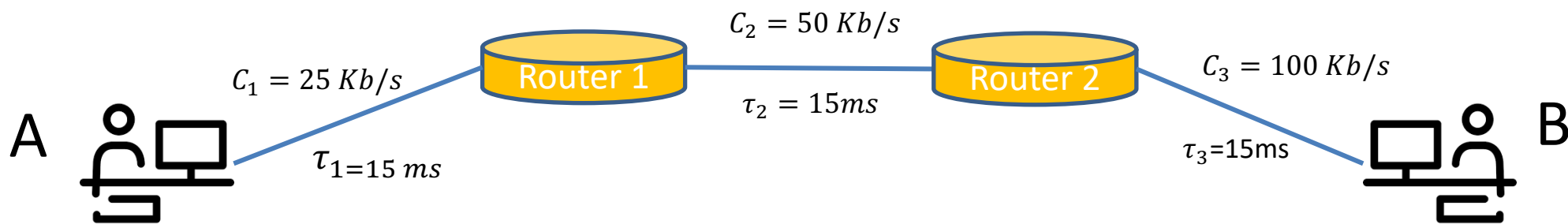
Primi 25 MSS
(ossia i primi 6 turni
 $1+2+4+5+6+7=25$)

Ultimi 75 MSS

Ack ultimo MSS



Esercizio 5



All'istante 0 viene attivata una connessione TCP tra l'host A e l'host B.

QUESITI:

- Si calcoli l'istante di tempo oltre il quale la trasmissione sul link 1 risulta continua, supponendo
 - ☐ *header* trascurabili
 - ☐ *link* bidirezionali e simmetrici
 - ☐ RCWND = 4000 [byte] e Ssthresh = 400 [byte]
 - ☐ dimensione segmenti MSS = 200 [byte]
 - ☐ dimensione ACK=dimensione segmenti per apertura della connessione = 20 [byte]
 - ☐ Connessione aperta dal terminale A
- Quanto tempo occorre per trasferire un file da 2 [kbyte] (dall'istante di trasmissione del primo segmento all'istante di ricezione dell'ACK dell'ultimo segmento)?



Soluzione Esercizio 5 – Quesito 1

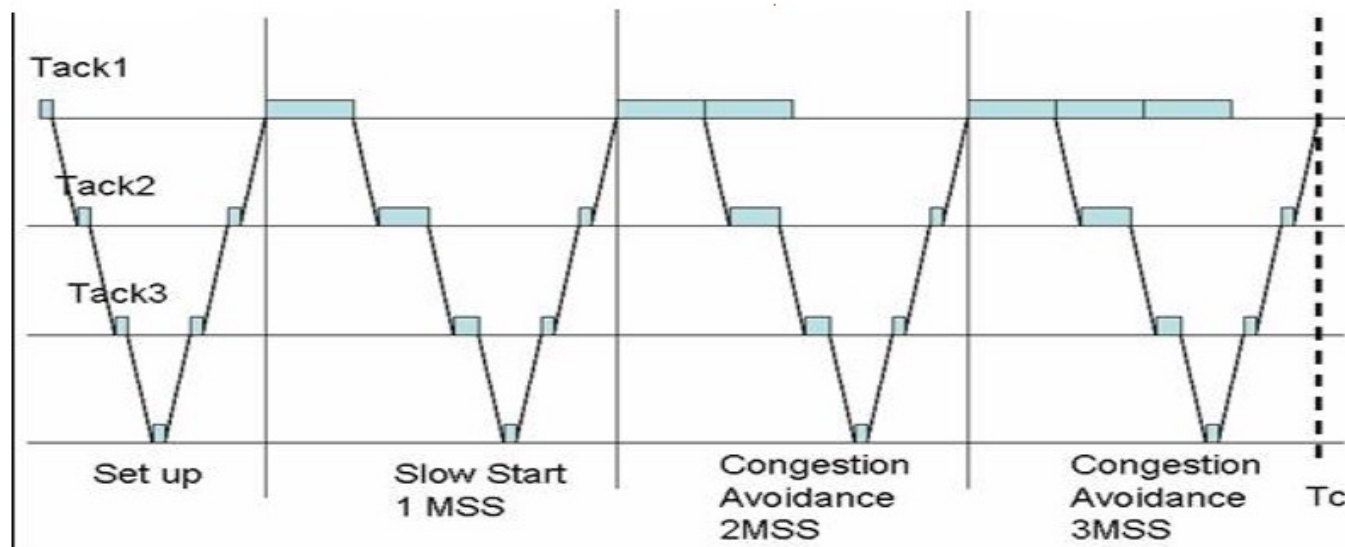
Calcoliamo un po' di valori:

- $T_1 = \frac{200 \cdot 8 \text{ [bit]}}{25 \left[\frac{\text{kbit}}{\text{s}}\right]} = 64 \text{ ms}$

- $T_2 = \frac{1}{2} T_1 = 32 \text{ ms}$

- $T_3 = \frac{1}{2} T_2 = 16 \text{ ms}$

- $RTT = T_1 + T_2 + T_3 + 2 \cdot (\tau_1 + \tau_2 + \tau_3) + (T_{ack_1} + T_{ack_2} + T_{ack_3}) = 64 \text{ [ms]} + 32 \text{ [ms]} + 16 \text{ [ms]} + 2 \cdot (45 \text{ [ms]}) + (6.4 \text{ [ms]} + 3.2 \text{ [ms]} + 1.6 \text{ [ms]}) = 213.2 \text{ ms}$



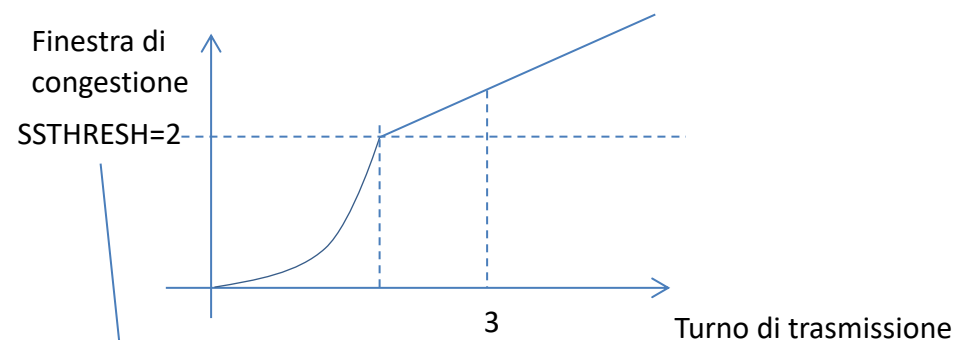
Soluzione Esercizio 5 – Quesito 1

- $T_{setup} = 2 \cdot (T_{ack_1} + T_{ack_2} + T_{ack_3}) + 2 \cdot (\tau_1 + \tau_2 + \tau_3) = 2 \cdot (6.4 [ms] + 3.2 [ms] + 1.6 [ms]) + 2 \cdot (45 [ms]) = 112.4 ms$
- Il link 1 è il collo di bottiglia, trasmissione è continua sul link 1 quando:
 $WT_1 > RTT$
- Quindi $W > \frac{RTT}{T_1} = \frac{213.2 ms}{64 ms} = 3.3$
- L'istante in cui la trasmissione diventa continua è

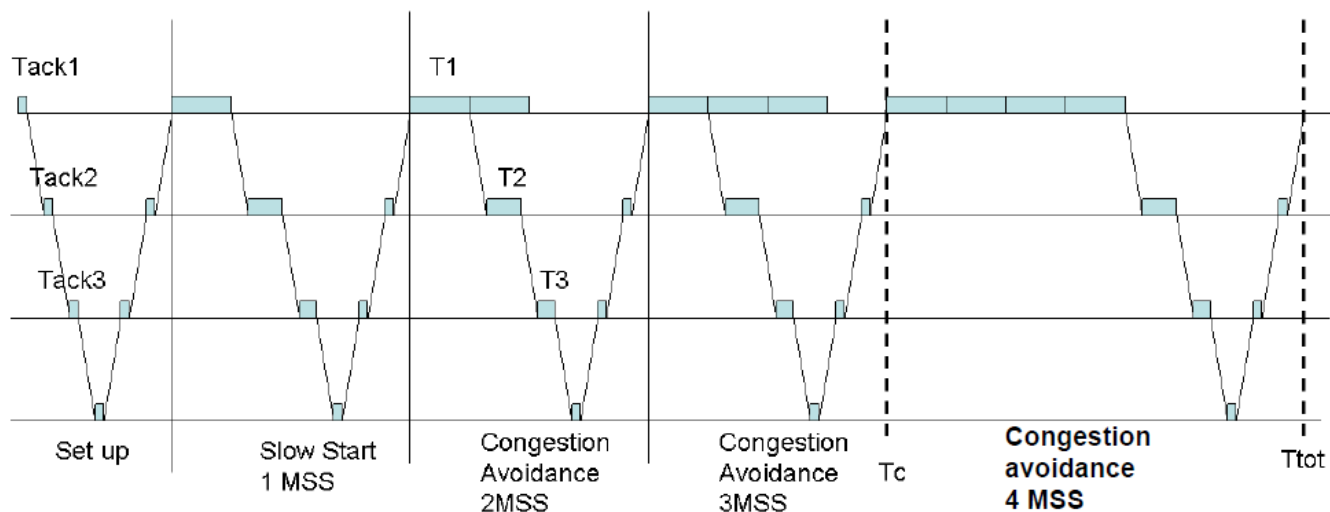
$$T_c = T_{setup} + 3 \cdot RTT = 112.4 [ms] + 3 \cdot 213.2 [ms] = 752 [ms]$$



Soluzione Esercizio 5 – Quesito 2



- Il file da trasferire è di 2 [kbyte], equivalenti a 10 MSS.
- Il tempo per trasferire 10 MSS è: $T_c = T_{setup} + 4 \cdot RTT + 3 \cdot T_1 = 1.15 [s]$



ossia i primi 4 turni ($W > 3.3$),
rimangono poi 3 MSS per completare
l'invio



Esercizio 6

- Una connessione TCP è usata per trasmettere un file da 39.5 [kbyte] utilizzando i seguenti parametri:
 - $MSS=500$ [byte]
 - $RTT = 500$ [ms]
 - $timeout\ T1 = 2*RTT$.
- Si assuma che le condizioni iniziali delle finestre siano:
 - $RCWND = 12$ [kbyte]
 - $SSTHRESH = 8$ [kbyte]
 - $CWND = 500$ [byte]
- E che inoltre:
 - si verifichi un errore sulla connessione all'istante 3 [s] (tutti i segmenti in trasmissione vengano persi)
 - al tempo 4,5 [s] il ricevitore segnali $RCWND = 2$ [kbyte]
- Si tracci l'andamento nel tempo di:
 - $CWND$
 - $SSTHRESH$
 - $RCWND$
- Si calcoli il tempo di trasmissione del file



Soluzione Esercizio 6 – Quesito 1

Conviene ragionare in numero di segmenti trasmessi

$$\text{Dimensione File (in MSS)} = \frac{39.5 \text{ [Kbyte]}}{500 \text{ [byte]}} = 79 \text{ MSS}$$

Dobbiamo trovare il tempo necessario per trasferire 79 MSS

$$\text{RCWND} = \frac{12 \text{ [kbyte]}}{500 \text{ [byte]}} = 24 \text{ MSS}$$

$$\text{SSTHRESH} = \frac{8 \text{ [kbyte]}}{500 \text{ [byte]}} = 16 \text{ MSS}$$

$$\text{Timeout} = 1 \text{ s}$$

Tempo di trasferimento del file:

$$T = 8.5 \text{ [s]}$$

