

# LABORATORIO DI INTERNET

## Report 3: Test performance con off the shelf hardware

Gruppo 21

Maggio 2021



**Politecnico  
di Torino**

Diego Zanfardino s256536,  
Fabio Trovero s258574,  
Lorenzo Ferro s260878

prof. Mellia Marco

# 0 Introduzione

In questo laboratorio si cerca di predirre il goodput di un modello di rete semplificato con l'utilizzo del comando *iperf3*. Per valutare l'effettiva velocità di connessione tra due host *iperf* utilizza:

$$goodput = \frac{dati\ utili\ (a\ livello\ 7)}{\Delta T} = \frac{\eta}{\Delta T} \quad (1)$$

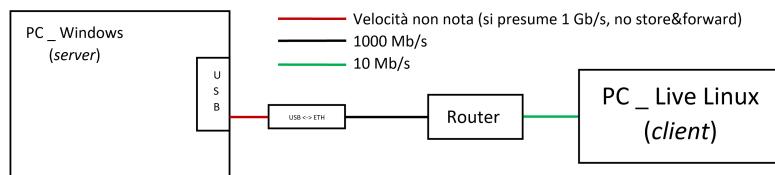
Noi siamo invece in grado di predirre il risultato dei test calcolando l'efficienza attesa della rete, e successivamente la velocità massima come segue:

$$V_{MAX} = \eta \cdot C = \frac{S}{S + header_{TCP|UDP} + header_{IP} + header_{ETH}} \cdot C \quad (2)$$

Dove  $C$  è la capacità del canale e  $S$  sono i dati utili scambiati a livello 7.

## 1 Scenario controllato: lan locale

### 1.1 Test TCP - Singolo flusso, full duplex



In questo primo scenario si collegano due pc, uno con live Linux (Ubuntu) e un altro con Windows, tramite router. L'assegnazione degli indirizzi è lasciata al DHCP del router, si imposta inoltre la velocità dell'interfaccia di rete tra pc Linux e router a 10Mb/s per semplicità di analisi: *"sudo ethtool -s eth0 speed 10 duplex full autoneg on"*. Per questo primo test si esegue il comando *iperf3 -s* per abilitare il server su Windows e *iperf3 -c 192.168.1.196* per aprire una connessione tra pc Linux (client) e server. Sono lasciate le opzioni di default dell'applicazione *iperf*:

- la durata del test è di 10 secondi, così da rendere trascurabile lo *store and forward* del router
- la lunghezza dei dati generati è pari ad un MSS, ovvero 1460 byte a livello applicazione (L7)

Utilizzando (2) si può prevedere l'efficienza del canale e di conseguenza la velocità ottenuta con *iperf3*.

$$V_{MAX} = \frac{MSS}{MSS + 20 + 20 + 38} \cdot C \simeq 9.5Mb/s \quad (3)$$

Prendendo in considerazione anche le opzioni TCP, si ottiene un'efficienza più coerente con i dati ottenuti:

$$V_{MAX} = \frac{MSS}{MSS + 32 + 20 + 38} \cdot C \simeq 9.41Mb/s \quad (4)$$

I problemi a livello fisico sono trascurabili, essendo i due host collegati tramite cavi ethernet di dimensioni ridotte. Non ci aspettiamo collisioni di alcun tipo essendo i canali *full-duplex*. In questo scenario non si dovrebbero generare problemi di congestione poichè il client che comunica con il server ha una velocità di connessione di due ordini di grandezza inferiore, quindi il server sarà in grado di smaltire il traffico senza problemi.

### 1.2 Risultati

Eseguendo il comando si ottiene un riscontro di quanto ipotizzato al punto precedente. In particolare eseguendo 10 test si ottengono i risultati in Figura 1. Si può notare che il bitrate del sender è leggermente sovrastimato rispetto a quello del receiver: ciò accade perché i  $\Delta T$  di sender e receiver sono differenti, il primo calcola il tempo trascorso dall'invio del primo pacchetto all'invio dell'ultimo, mentre il secondo usa tempi più grandi poichè deve attendere la ricezione. Si ottiene quindi una velocità media di 9.58 Mbit/s al sender e 9.39 Mbit/s al receiver.

```
lo@ubu:~/Download$ cat TCPCC.dat | grep "sender\| receiver"
[ 5] 0.00-10.00 sec 11.5 MBytes 9.62 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.41 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.5 MBytes 9.62 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.40 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.5 MBytes 9.62 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.39 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.5 MBytes 9.62 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.40 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.4 MBytes 9.57 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.40 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.4 MBytes 9.57 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.39 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.3 MBytes 9.52 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.1 MBytes 9.35 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.5 MBytes 9.62 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.42 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.3 MBytes 9.47 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.1 MBytes 9.33 Mbits/sec 0 receiver
[ 5] 0.00-10.00 sec 11.4 MBytes 9.57 Mbits/sec 0 sender
[ 5] 0.00-10.00 sec 11.2 MBytes 9.41 Mbits/sec 0 receiver
```

Figura 1

### 1.3 Test TCP - Singolo flusso, half duplex

In questo caso è possibile predire l'efficienza della rete tenendo in considerazione il funzionamento di TCP: il protocollo prevede infatti l'invio di un ACK di dimensioni  $20+20+38$ , una volta che il receiver ha ricevuto il pacchetto.

$$\eta = \frac{MSS}{MSS + 20 + 32 + 38 + (20 + 20 + 38)} \simeq 0.89 \quad (5)$$

Essendo il canale condiviso per trasmissione e ricezione possono verificarsi collisioni a livello fisico che porterebbero a una diminuzione del goodput calcolato. Eseguendo il test si ottengono le seguenti velocità di trasmissione

Come ipotizzato la velocità è più lenta di quella calcolata per via delle collisioni, gestite direttamente dalla scheda di rete, visualizzabili con *ifconfig* prima e dopo il comando *iperf3*

### 1.4 Test TCP - Singolo flusso, full duplex, reverse mode

```
Connecting to host 192.168.1.196, port 5201
Reverse mode, remote host 192.168.1.196 is sending
[ 5] local 192.168.1.210 port 49392 connected to 192.168.1.196 port 5201
[ ID] Interval Transfer Bitrate
[ 5] 0.00-1.00 sec 847 Kbytes 6.94 Mbits/sec
[ 5] 1.00-2.00 sec 1.13 MBytes 9.46 Mbits/sec
[ 5] 2.00-3.00 sec 1.13 MBytes 9.47 Mbits/sec
[ 5] 3.00-4.00 sec 1.13 MBytes 9.48 Mbits/sec
[ 5] 4.00-5.00 sec 1.13 MBytes 9.47 Mbits/sec
[ 5] 5.00-6.00 sec 1.13 MBytes 9.48 Mbits/sec
[ 5] 6.00-7.00 sec 1.13 MBytes 9.48 Mbits/sec
[ 5] 7.00-8.00 sec 1.13 MBytes 9.49 Mbits/sec
[ 5] 8.00-9.00 sec 1.13 MBytes 9.47 Mbits/sec
[ 5] 9.00-10.00 sec 1.13 MBytes 9.49 Mbits/sec
[ 5] 0.00-10.00 sec 11.1 MBytes 9.33 Mbits/sec
[ 5] 0.00-10.00 sec 11.0 MBytes 9.22 Mbits/sec
sender receiver
```

Figura 2: TCP PC - R

Facendo riferimento al setup precedente si esegue il comando *iperf3 -c 192.168.1.196 -R* al client, in questo modo è il client a mandare un primo messaggio al server per richiedere la connessione inversa, scambiando così i ruoli di client e server. In questo scenario la velocità del client è molto maggiore di quella del server, per questo motivo interverranno il controllo di congestione e di flusso per adattare, nei primi istanti della connessione, la velocità di *sender* e *receiver* per evitare congestione e ritrasmissioni che diminuirebbero il goodput della rete.

In figura 2 è possibile notare l'intervento del controllo di congestione guardando la quantità di dati scambiati nel primo secondo rispetto agli istanti successivi. Eseguendo il comando 10 volte è possibile ricavare le velocità medie di sender e receiver, rispettivamente 9.33 e 9.20.

### 1.5 Test UDP - Singolo flusso, full duplex

L'efficienza di UDP è massima se si riescono a scambiare la maggiore quantità di bit senza che avvenga frammentazione del pacchetto, ovvero quando  $S + 8 + 20 = MTU$ , quindi  $S_{MAX} = 1472$

Dalla (2) si ricava che

$$\eta = \frac{1472}{1472 + 8 + 20 + 38} \simeq 0,957 \rightarrow V_{TXMAX} = C \cdot \eta = 9,57 Mb/s \quad (6)$$

Nel caso del singolo flusso full duplex non ci aspettiamo problemi a livello fisico, né problemi di congestione della rete. Poiché UDP è un protocollo inaffidabile e non aspetta conferme dall'interlocutore, l'efficienza per connessioni half duplex rimane la stessa di quella calcolata precedentemente. La velocità ottenuta risulta essere maggiore di quella TCP per via delle dimensioni ridotte dell'header UDP

### 1.6 Risultati

```
Log@Ubuntu:~/Documents/Labint/Lab4$ cat UDP_PC.dat | grep " sender| receiver"
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.044 ms 2533/10708 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.015 ms 2534/10710 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.030 ms 2533/10708 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.000 ms 2533/10707 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.021 ms 2532/10706 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 1.870 ms 2534/10710 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 2.010 ms 2533/10708 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 1.978 ms 2533/10708 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10750 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.54 Mbits/sec 2.118 ms 2531/10703 (24%) receiver
[ 5] 0.00-10.00 sec 15.0 MBytes 12.6 Mbits/sec 0.000 ms 0/10760 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.55 Mbits/sec 1.751 ms 2534/10710 (24%) receiver
```

Figura 3: UDP PC

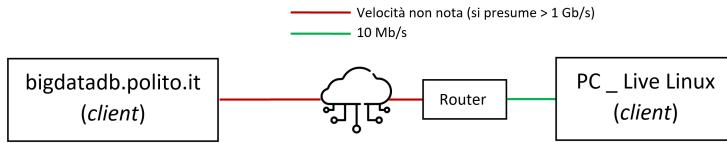
A conferma della maggiore efficienza del protocollo UDP eseguendo i 10 test si ottiene una velocità media di 12.6 al sender e 9.56 al receiver. Poiché non viene limitato il bitrate di generazione dei pacchetti (opzione *-b 0*) e il sender li genera ad una velocità maggiore di 10 Mb/s una parte dei pacchetti creata non può essere smaltita dalla pila protocollare del sender, in questo caso il 24%.

## 1.7 Test UDP - Singolo flusso, full duplex, reverse mode

```
Laboratorio@laboratorio: $ iperf3 -c 192.168.1.196 -u -b 0 -R
Connecting to host 192.168.1.196, port 5201
Reverse mode, remote host 192.168.1.196 is sending
[ 5] local 192.168.1.210 port 36376 connected to 192.168.1.196 port 5201
[ ID] Interval Transfer Bitrate Jitter Lost/Total Datagrams
[ 5] 0.00-1.00 sec 1.13 MBytes 9.50 Mbytes/sec 1.641 ms 63566/64379 (99%)
[ 5] 1.00-2.00 sec 1.13 MBytes 9.50 Mbytes/sec 1.602 ms 78989/79802 (99%)
[ 5] 2.00-3.00 sec 1.13 MBytes 9.52 Mbytes/sec 1.566 ms 74661/75476 (99%)
[ 5] 3.00-4.00 sec 1.13 MBytes 9.55 Mbytes/sec 1.636 ms 73455/74273 (99%)
[ 5] 4.00-5.00 sec 1.13 MBytes 9.52 Mbytes/sec 1.604 ms 77375/78190 (99%)
[ 5] 5.00-6.00 sec 1.13 MBytes 9.55 Mbytes/sec 1.600 ms 76084/76982 (99%)
[ 5] 6.00-7.00 sec 1.14 MBytes 9.55 Mbytes/sec 1.602 ms 76284/77082 (99%)
[ 5] 7.00-8.00 sec 1.14 MBytes 9.55 Mbytes/sec 1.590 ms 76040/76858 (99%)
[ 5] 8.00-9.00 sec 1.14 MBytes 9.55 Mbytes/sec 1.658 ms 77777/78595 (99%)
[ 5] 9.00-10.00 sec 1.14 MBytes 9.54 Mbytes/sec 1.560 ms 74951/75768 (99%)
[ 5] 0.00-10.00 sec 1.04 GBytes 892 Mbytes/sec 0.000 ms 0/759304 (0%) sender
[ 5] 0.00-10.00 sec 11.4 MBytes 9.53 Mbytes/sec 1.560 ms 751141/759304 (99%) receiver
iperf Done.
```

Figura 4: UDP PC - R

## 2 Scenario internet



In questo secondo caso in esame si comunica con un server in rete. Si ipostizza una velocità dopo il default gateway in media maggiore di 1 Gb/s (il test è effettuato con fibra FTTH). In tal modo il collo di bottiglia è rappresentato dalla velocità tra il nostro Pc è l'home router.

### 2.1 Test TCP - Singolo flusso, full duplex

Eseguendo il comando `iperf3 -c bigdata.polito.it` si comunica come client con il server. L'efficienza attesa è la stessa calcolata in (4) e non ci si aspetta congestione di rete.

### 2.2 Risultati

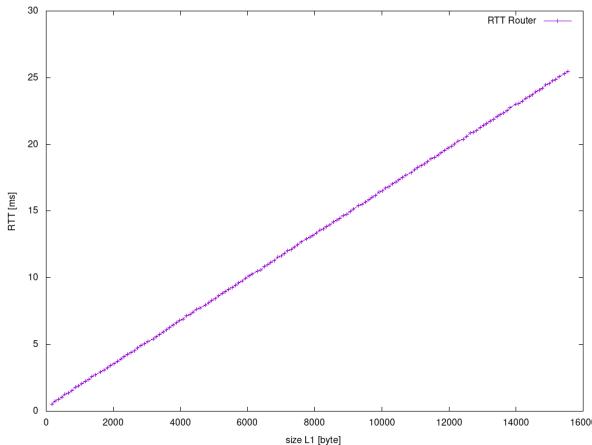


Figura 5: RTT

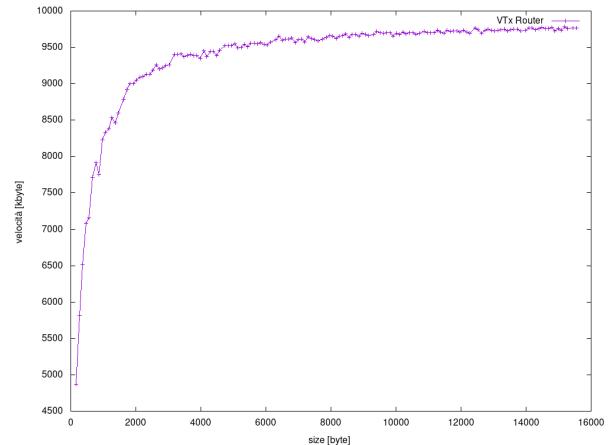


Figura 6: VTx

In questo caso si rielaborano le formule (2) (4) nel modo seguente:

$$RTT = 2T_{TX} + T_\eta \quad (7)$$

$$V_{TX} = \frac{2D(s)}{RTT} \quad (8)$$

Coinvolgendo solamente due dispositivi collegati mediante un cavo ethernet di modesta lunghezza il comportamento riscontrato è soggetto a meno variazioni ed è molto simile al comportamento teorizzato.

Collisioni prima del test	Collisioni dopo il test	Velocità sender	Velocità receiver
44706	46985	7.13	6.94
46985	50133	7.03	6.82
50133	52652	7.13	7.03
52652	54812	7.24	7.08
54812	56999	7.24	7.06
56999	60774	6.71	6.55
60774	63026	7.13	6.96
63026	65215	7.24	7.05
65215	67387	7.03	6.89
67387	69572	7.13	6.93

## A Script per collezionare i dati

```

for((s=100; s<15000; s+=100))
do
    echo -n $s >> dataset.dat
    sudo ping 192.168.1.XX -s $s -c 5 -i 0,05 | grep min
    | cut -d '=' -f2 | cut -d '/' -f1 >> dataset.dat
done

```

## B Script configurazione 1

```

set xlabel "size L1 [byte]"
set ylabel "RTT [ms]

set terminal png size 1024, 768

D(x)=(x+8)+(20+38)*(1+floor((x+8-1)/1480))

set output "RTTp.png"

plot 'rtt_min_h2.dat' using (D($1)):2 title "RTT powerline" with linespoint

set xlabel "size [byte]"
set ylabel "velocità [kbyte]"
set output "VTx.png"

plot 'rtt_min_h2.dat' using (D($1)):(\$1<1472 ? (8*4*(D($1))/(\$2)) :
((8*2*(D($1))/(\$2)) + (8*2*(D(1538)/(\$2))))) :
title "VTx H2 powerline" with linespoint

```

## C Script configurazione 2

```

set xlabel "size L1 [byte]"
set ylabel "RTT [ms]

set terminal png size 1024, 768

D(x)=(x+8)+(20+38)*(1+floor((x+8-1)/1480))

set output "RTTp.png"

plot 'rtt_min_h2.dat' using (D($1)):2 title "RTT powerline" with linespoint

set xlabel "size [byte]"

```

```
set ylabel "velocità [kbyte]"
set output "VTx.png"

plot 'rtt_min_h2.dat' using (D($1)):(8*2*(D($1)))/($2))
    title "VTx H2 powerline" with linespoint
```

## D Script configurazione 3

```
set xlabel "size L1 [Byte]
set ylabel "RTT [ms]

set terminal png size 1024, 768

D(x)=(x+8)+(20+38)*(1+floor((x+8-1)/1480))

set output "RTTW.png"

plot 'rtt_min_wifi.dat' using (D($1)):2 title "RTT" with linespoint

set xlabel "size [Byte]
set ylabel "velocità [KBit]

set output "VTxW.png"

plot 'rtt_min_wifi.dat' using (D($1)):( D($1)<1500 ?
    (((2*8*10000000*D($1))/(( 10000)*($2) - 2*8*D($1 )))*0.001) :
    (((2*8*10000000*1538)/(( 10000)*($2) - 2*8*D($1 )))*0.001 ) )
    title "VTx" with linespoint
```

## E Script configurazione 4

```
set xlabel "size L1 [byte]
set ylabel "RTT [ms]

set terminal png size 1024, 768

D(x)=(x+8)+(20+38)*(1+floor((x+8-1)/1480))

set output "RTTPow.png"

plot 'rtt_min_h2_con_powerline.dat' using (D($1)):2 title "RTT" with linespoint

set xlabel "size [byte]
set ylabel "velocità [kbyte]

set output "VTxPow.png"

plot 'rtt_min_h2_con_powerline.dat' using (D($1)):( D($1)<1500 ?
    (4*8*D($1)*10**7*10**8/(( $2)*10**4*10**8 - (2*8*10**8*D($1)) -
    (6*8*10**7*D($1))) )*0.001 : (4*8*1538*10**7*10**8/(( $2)*10**4*10**8 -
    (2*8*10**8*D($1)) - (6*8*10**7*D($1))) )*0.001 ) with linespoint
```