Sara Oliveri (S5578081), Fabio Benjamine Roncallo (S5601246), Marco Zoratti (S5562866).

Relazione Laboratorio “Ping-Pong”

# Sviluppo e Codice

Tutto il codice, i file .dat ed eseguibili sono disponibili online su un repository GitHub pubblico: [Uni-Informatica/SecondoAnno/SETI/pingpong at master · marchacio/Uni-Informatica (github.com)](https://github.com/marchacio/Uni-Informatica/tree/master/SecondoAnno/SETI/pingpong)

#### tcp\_ping.c

Il file tcp\_ping.c implementa la parte client del progetto che sfrutta il protocollo TCP.

##### Difficoltà e problemi riscontrati

La difficoltà maggiore riscontrata è stato comprendere il codice già presente e implementarne di nuovo in linguaggio C.

Difficoltà superata consultando numerose volte il MAN (sia online sia da terminale).

#### udp\_ping.c

Il file udp\_ping.c implementa la parte client del progetto che sfrutta il protocollo UDP .

##### Difficoltà e problemi riscontrati

La difficoltà maggiore è stata l’implementazione di chiamate di sistema non bloccanti.

In particolare all’interno della funzione *do\_ping* quando bisogna ricevere la risposta del server attraverso il socket ([righe 79-100](https://github.com/marchacio/Uni-Informatica/blob/2163fc9c88b3d9eb348fe9e341a14731cfed42cc/SecondoAnno/SETI/pingpong/udp_ping/udp_ping.c#L79C3-L100C4)):

La strategia adottata prevede di creare loop che esegue la chiamata di sistema *recv* ogni *RIDUZIONE* millisecondi; quest’ultima variabile è un intero costante che decrementa un timer che una volta arrivato a zero, termina il loop in quanto il client non ha ricevuto risposta dal server.

#### pong\_server.c

Il file pong\_server.c implementa la parte server del progetto che sfrutta sia il protocollo UDP sia il TCP.

##### Difficoltà e problemi riscontrati

La difficoltà maggiore è stato il completamento della funzione *server\_loop*, in particolare l’implementazione della chiamata di sistema *accept*.

Infatti capita spesso che la *syscall accept* ritorni un errore di tipo *EINTR*, ovvero *Interrupted-System-Call.*

Per ovviare a questo problema è bastato creare un loop tramite un’etichetta chiamata *again* che in caso il codice di errore sia uguale a *EINTR*, riesegue la chiamata.

Aggiungendo una *printf* di debug, si può notare che mediamente la syscall *accept* fallisce due volte su tre per un errore di tipo *Interrupted-System-Call.*

Non è chiaro se questo problema sia limitato alle macchine e ai sistemi operativi utilizzati dai componenti del gruppo.

# Script Bash

###### Dato che Bash non supporta i numeri in virgola mobile, nell’implementazione del codice è stato utilizzato il programma “bc” impostato su risultati con al massimo 9 cifre decimali.

I grafici del modello banda-latenza sono stati creati utilizzando il file *graph.bash* che prende in input un file .*dat* contenente dati divisi in 3 colonne:

Per ogni riga, il primo valore contiene la dimensione del messaggio, il secondo e il terzo contengono la mediana e la media del throughput.

Il file graph.bash prende dal file .*dat* la dimensione minima (N1) e massima (N2) dei pacchetti e il loro throughput medio (T1 e T2).

Una volta ottenuti questi dati, il programma calcola il Delay (D1 e D2) dei due messaggi tramite la seguente formula:

*dati il throughput T e la dimensione N, il delay D è: D=N/T.*

Per poter calcolare la Banda (B) e la Latenza (L) bisogna applicare le seguenti formule:

*B= (N2-N1)/(D2-D1)*

*L = (D1\*N2 - D2\*N1)/(N2-N1)*

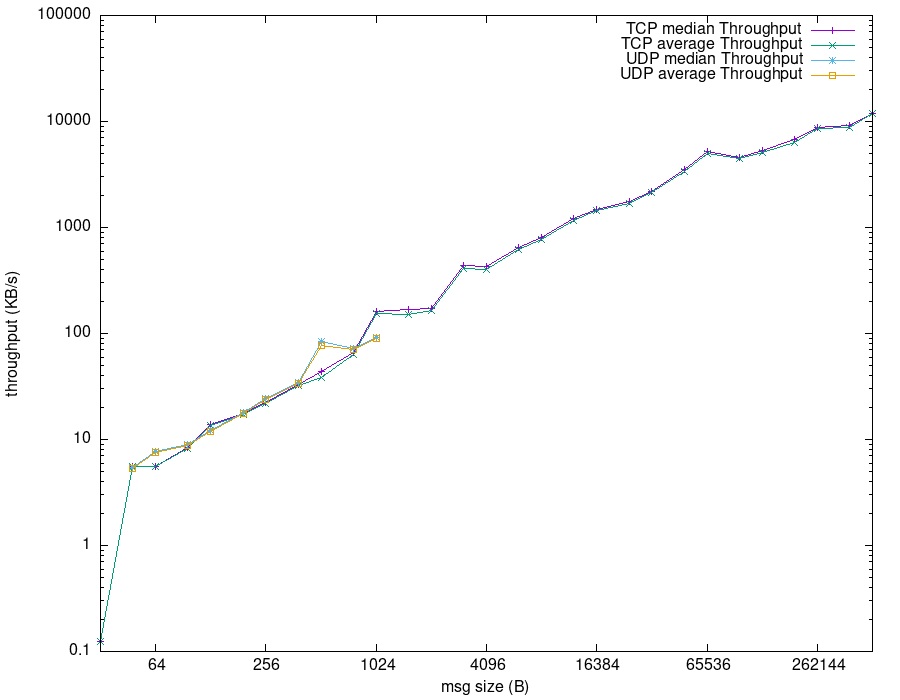
Una volta ottenuti B, L e il nome del file, il programma genera il grafico del modello banda-latenza tramite l’esecuzione di “gnuplot” su una funzione f(x) = x / (L + x/B).

Una volta generato il file .*png* contenente il grafico, termina.

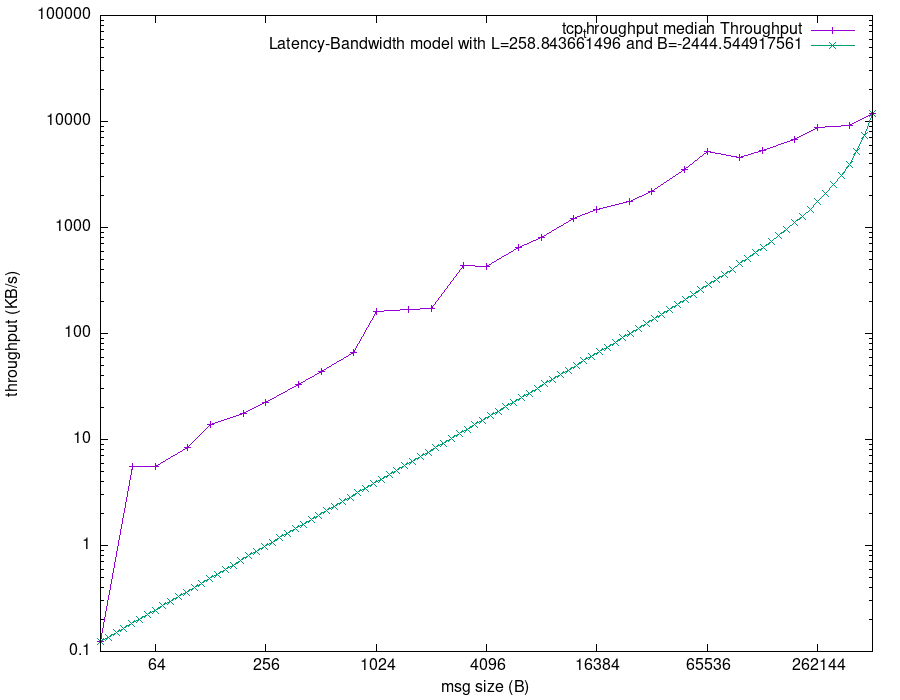
# Grafici

### Grafici creati sulla base dei test eseguiti usando il server *seti.dibris.unige.it*:

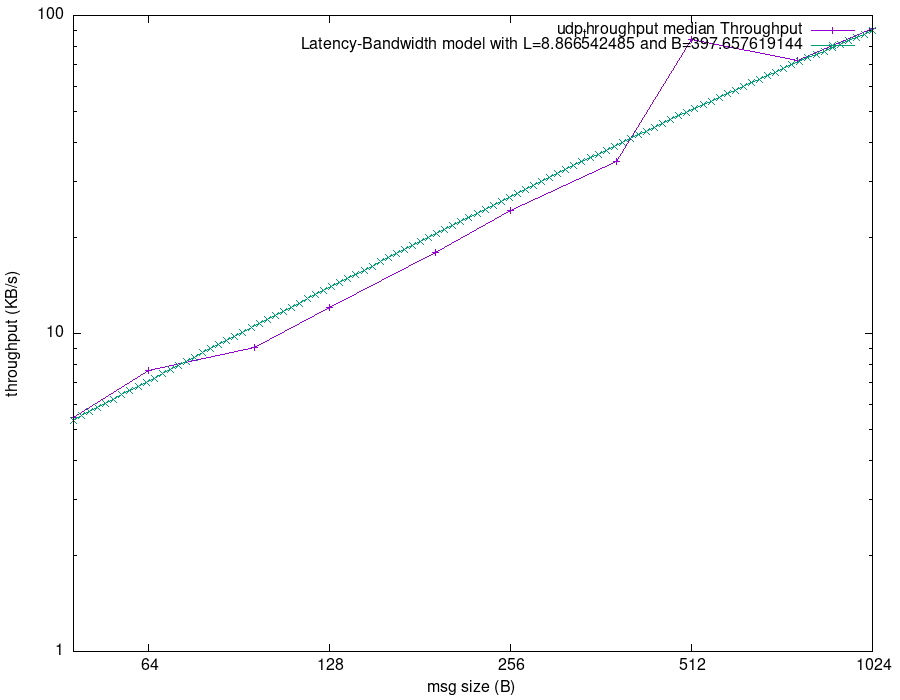
1. Throughput:



1. Modello banda-latenza con TCP:



1. Modello banda-latenza con UDP:



Dai grafici si può notare la principale differenza tra il protocollo TCP e UDP:

Il primo protocollo mantiene una connessione stabile anche quando la dimensione dei pacchetti è elevata, cosa che non avviene con il secondo.

Non garantendo la ricezione dei datagrammi, la connessione UDP risulta più veloce ma meno affidabile infatti una volta superata la dimensione di 1024 byte per datagramma, il dialogo tra le due macchine viene interrotto.

##### 

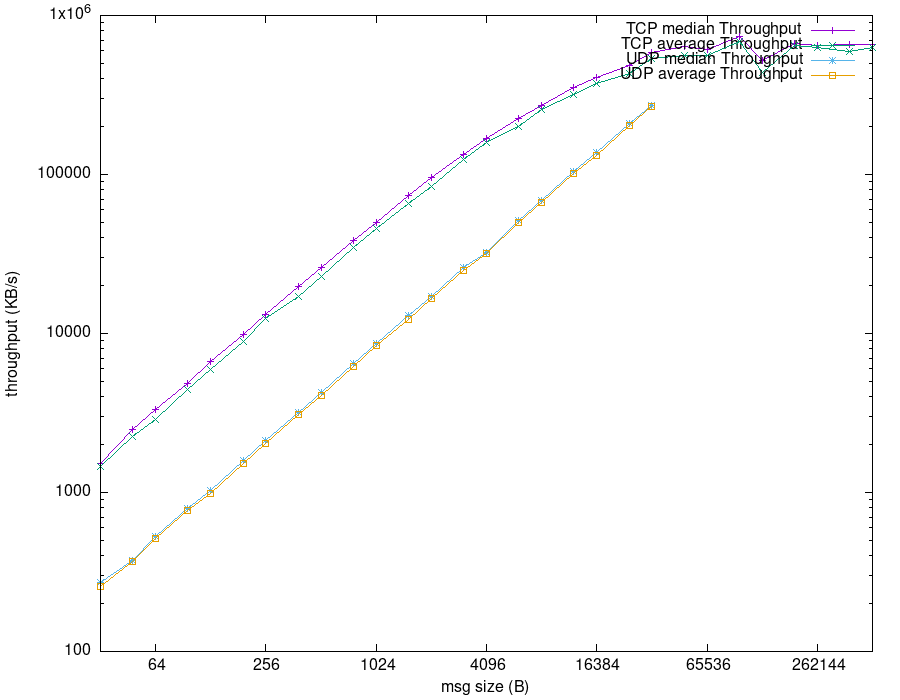
##### 

##### 

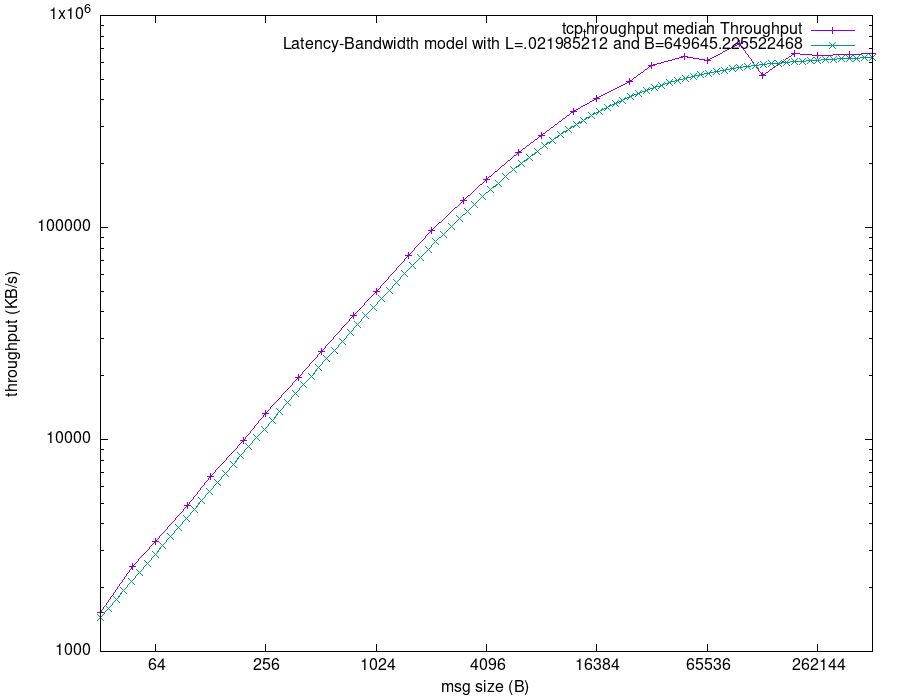
##### 

### Grafici creati sulla base dei test eseguiti usando il server *localhost*:

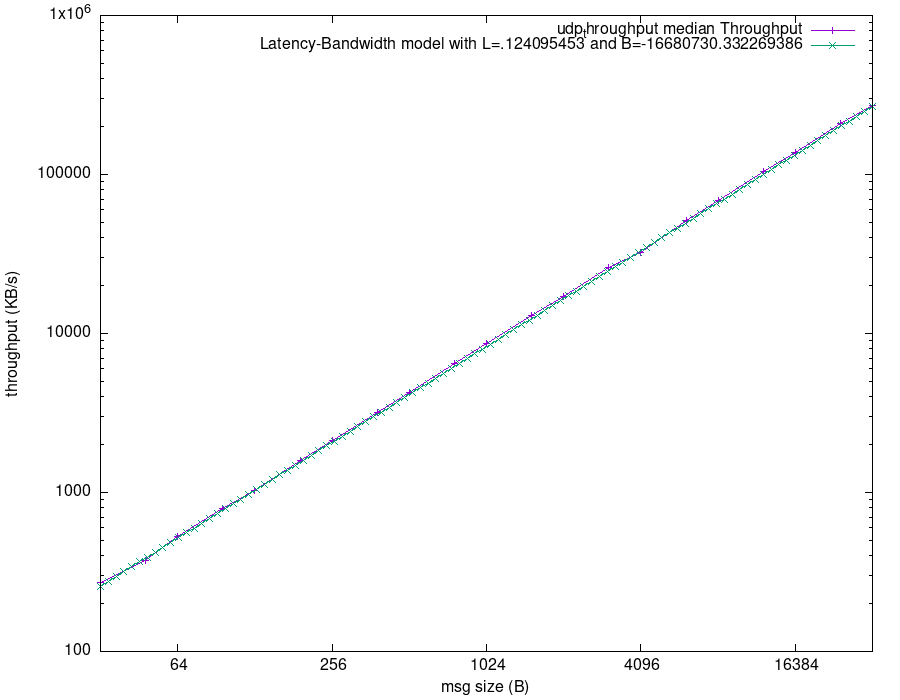
1. Throughput:



1. Modello banda-latenza con TCP:



1. Modello banda-latenza con UDP:



Come si può notare, in localhost il protocollo UDP riesce a dialogare col server anche con datagrammi di dimensione elevata in quanto la rete locale è senza vincoli, traffico e perdite di dati.