Costa Davide, Forneris Luca, Ivaldi Emiliano

Laboratorio 1

Relazione Scritta

Lo scopo di questa relazione è discutere e approfondire i valori e i risultati ottenuti dalla compilazione e dall'esecuzione dei file messi a disposizione dai professori per il primo laboratorio del corso di SET, riguardante l'interazione client-server tra computer (ping-pong).

A nostra disposizione le seguenti directories, con all'interno i file su cui abbiamo lavorato (che descriveremo più approfonditamente in seguito):

1) *bin* - creata dal Makefile e contenente gli eseguibili
2) *data* – creata dal Makefile e utilizzata dagli script come deposito per i dati
3) *pingpong_lib* - contenente il file header *pingpong.h* ed il sorgente *statistics.c*
4) *pong_server* (parte facoltativa) – che contiene la parte relativa all'implementazione del lato server
5) *udp_ping* – contenente i file per l'implementazione del lato client (protocollo UDP)
6) *tcp_ping* – contenente i file per l'implementazione del lato client (protocollo TCP)
7) *scripts* – contenente file in linguaggio script bash che serviranno per visualizzare i grafici

TCP_PING.C

Il compito del sorgete *tcp_ping.c* è quello di stabilire una connessione ad un server (per esempio: *webdev.dibris.unige.it*) mediante protocollo di trasporto TCP, che garantisce l'affidabilità della connessione stabilita. Di conseguenza il tipo di socket usato in questo file .c è il socket di tipo stream. Per completare il file *tcp_ping.c* abbiamo dovuto inizializzare alcuni campi della struct *gai_hints* (ai_family, ai_socktype e ai_protocol) ci siamo occupati, poi, delle funzioni:

main():

- **getaddrinfo() -** si occupa dell'inizializzazione della struct *addrinfo*: è definita nel file di sistema *netdb.h* e può essere usata facendo un include di questo file. Serve alle funzioni per la gestione dei socket per sapere quale protocollo a livello trasporto usare (TCP o UDP) quale protocollo a livello Network usare (IPv4 o IPv6), su quale indirizzo IP connettersi e su quale porta. Poiché si tratta di una struct, abbiamo dovuto inizializzare alcuni dei suoi campi: *int ai_family*, *int ai_socktype* e *int ai_protocol*. In particolare il primo a AF_INET, il secondo a SOCK_STREAM e il terzo a 0.

- **socket() -** funzione che inizializza un socket e restituisce un file descriptor. Per vedere i valori che tale funzione prende in ingresso abbiamo consultato il manuale tramite il comando *man socket*, e abbiamo passato a tale funzione come parametri i campi della struct sopra descritti.

- **connect() -** funzione che stabilisce una connessione tra il client e il server: così come per la funzione socket, una volta consultato il manuale abbiamo deciso di passare a tale funzione gli adeguati parametri necessari per stabilire effettivamente la connessione.

- **send()** - funzione che si impegna a inviare un certo numero di byte: la lunghezza di tali byte sono passati come argomento.

do_ping():

- **sprintf() -** utilizzata per scrivere il messaggio nel buffer che verrà poi spedito al server tramite la funzione send().

- **clock_gettime() -** funzione che salva l'ora all'interno di uno struct, utilizzata una volta con send_time e poi nuovamente con recv_time.

- **send() -** uguale alla *send()* del *main()* ma con un messaggio differente.

UDP_PING

Nel caso del sorgente *udp_ping.c* I passaggi che abbiamo svolto risultano simili a quelli fatti in precedenza nel file *tcp_ping.c*. Le funzioni di cui ci siamo occupati sono:

main():

- **getaddrinfo() -** uguale alla *getaddrinfo()* del file *tcp_ping.c*

- **socket() -** uguale alla *socket()* del *tcp_ping.c*

- **connect() -** uguale alla *connect()* del *tcp_ping.c*

- **send() -** uguale alla *send()* del *tcp_ping.c*

prepare_udp_socket():

Le differenze più evidenti si trovano all'interno di questa funzione (assente nel file tcp_ping.c), ad esempio il tipo di socket utilizzato: se il protocollo di trasporto TCP si appoggia al socket di tipo di tipo stream, il protocollo UDP si appoggia invece al socket di tipo datagram. Dunque abbiamo inizializzato I campi della struct *addrinfo*, ponendo pero' il parametro *ai_socktype* uguale a SOCK_DGRAM, in quanto il protocollo usato non e' piu' TCP.

- **fcntl()** - funzione che trasforma il socket file descriptor da BLOCKING a NONBLOCKING.

- **getaddrinfo()** - uguale alla *getaddrinfo()* del file *tcp_ping.c*

- **connect() -** uguale alla *connect()* del *tcp_ping.c*

do_ping():

- **sprintf() -** uguale alla sprintf*()* del *tcp_ping.c*

- **clock_gettime() -** uguale alla clock_gettime*()* del *tcp_ping.c*

- **send() -** uguale alla *send()* del *tcp_ping.c*

- **recv()** - funzione che si occupa di ricevere la risposta dal server, che sia locale o no. Nel file *tcp_ping.c* questa funzione era presente, ma già definita. In *udp_ping.c*, invece, abbiamo scritto questa funzione consultando il man e passandole gli adeguati argomenti.

PONG_SERVER

In quanto indietro con il lavoro, abbiamo scelto di non implementare il lato server del ping-pong.

STATISTICS.C

Questo file non implementa nessuna funzione relativa al lato client-server, bensì si occupa del trattamento in maniera statistica dei dati che si possono ricavare dall'esecuzione del programma.

I valori ricavati che verranno poi trattati statisticamente sono:

- **RTT** (Round Trip Time), ossia il tempo che impiega un pacchetto per essere inviato da un

computer a un altro e tornare indietro

Le funzioni di cui ci siamo occupati sono:

- **delta2milliseconds()** - Funzione che si occupa della conversione della differenza di due tempi (*previous* e *last*) in millisecondi.

- **print_statistics()** - Funzione che si occupa della stampa dei dati relativi all'istogramma e del median throughput

SCRIPTS

La parte inerente agli scripts è stata la parte più difficile del laboratorio, in quanto nessuno fra i membri del gruppo aveva mai avuto esperienza con gli script bash, ma seguendo le istruzioni trovate su AulaWeb siamo riusciti a capire l'esercizio.

Questi ultimi sono i grafici che raffigurano l'andamento del throughput, sono stati effettuati su un Raspberry Pi 3 Model B collegato al server webdev.dibris.unige.it alla porta 1491:

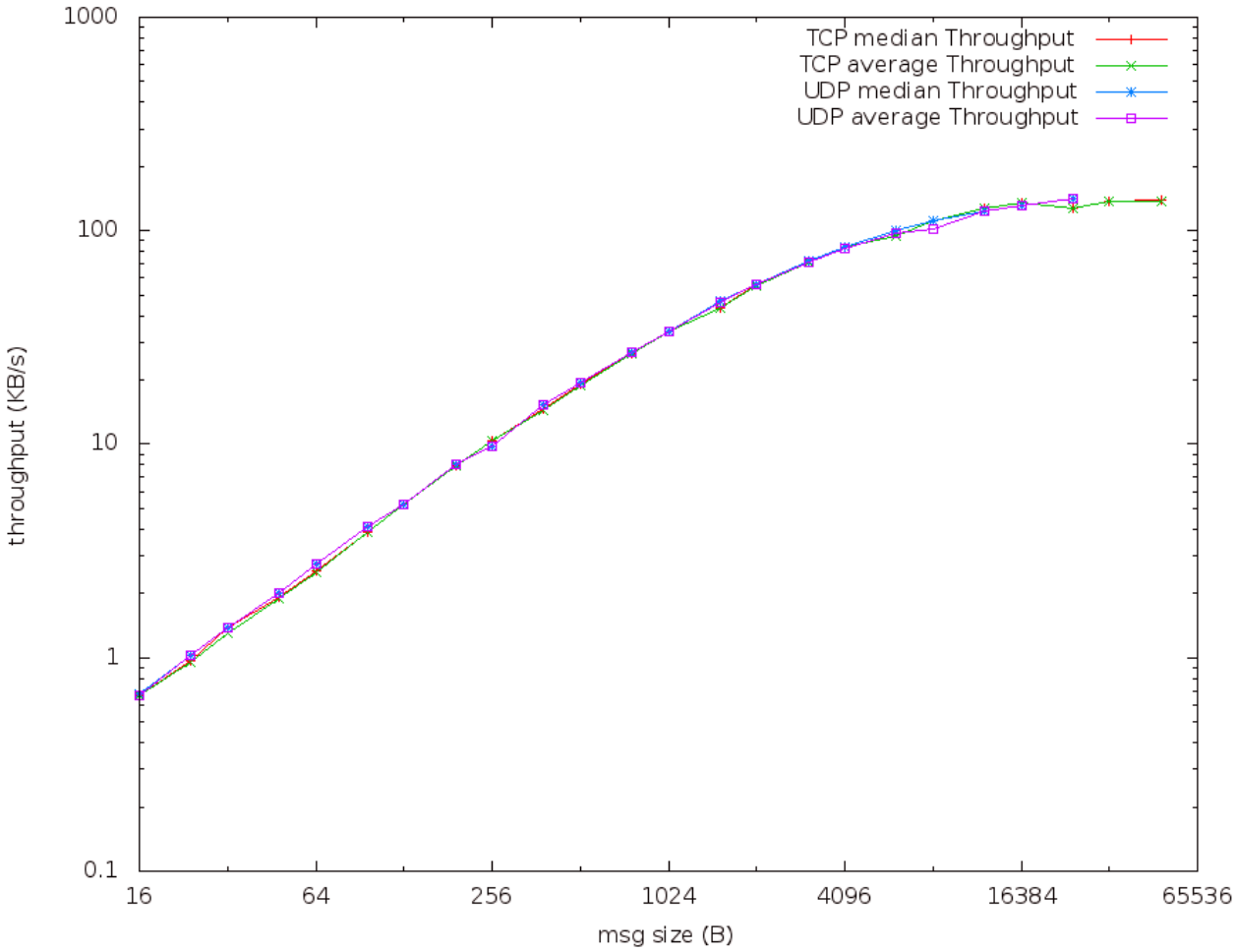

Illustrazione 1: Throughput

L'immagine in questione descrive l'andamento del throughput (asse delle ordinate) al crescere della dimensione, in byte, del messaggio inviato (asse delle ascisse), in base al protocollo di trasporto utilizzato (TCP o UDP) e in base alla scelta fra media o mediana.

Vengono raffigurati qui sotto i casi in base al protocollo scelto.

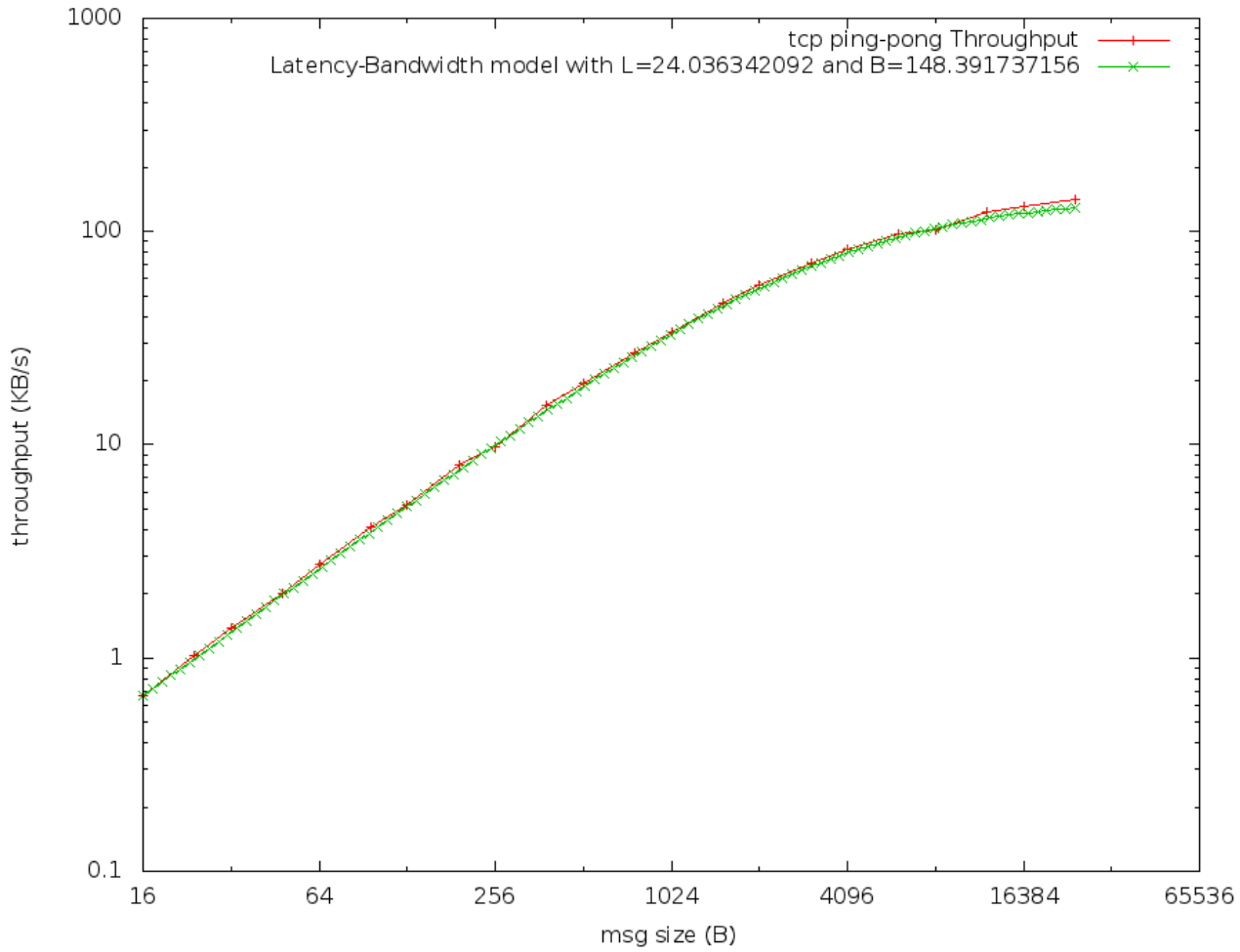

Illustrazione 2: Throughput TCP

Si osserva che la curva del throughput calcolato è, da circa 8192B in poi, al di sopra di quella relativa al throughput dedotto: si può dire di conseguenza che la deduzione è stata fatta per difetto, ma che in generale le due curve si muovono insieme e che, di conseguenza, il throughput calcolato e quello dedotto sono simili.

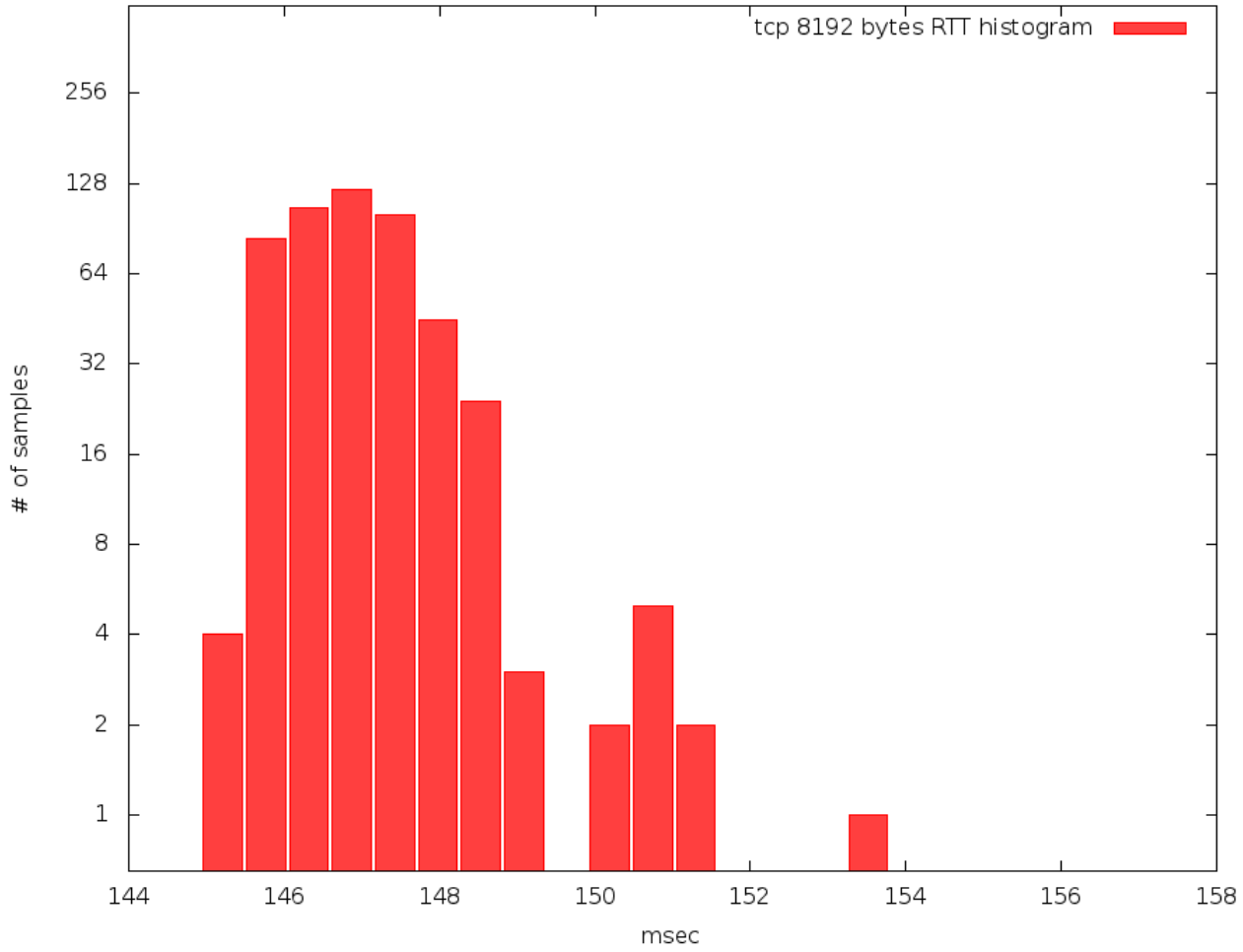

Illustrazione 3: Istogramma relativo al protocollo TCP

Qui sopra invece viene raffigurato, tramite istogramma, l'adamento del Round Trip Time all'aumento dei millisecondi nel caso relativo al protocollo TCP con 8192 bytes.

Vediamo ora, per trarre le necessarie conclusioni, lo stesso modello grafico/istogramma nel caso del protocollo UDP:

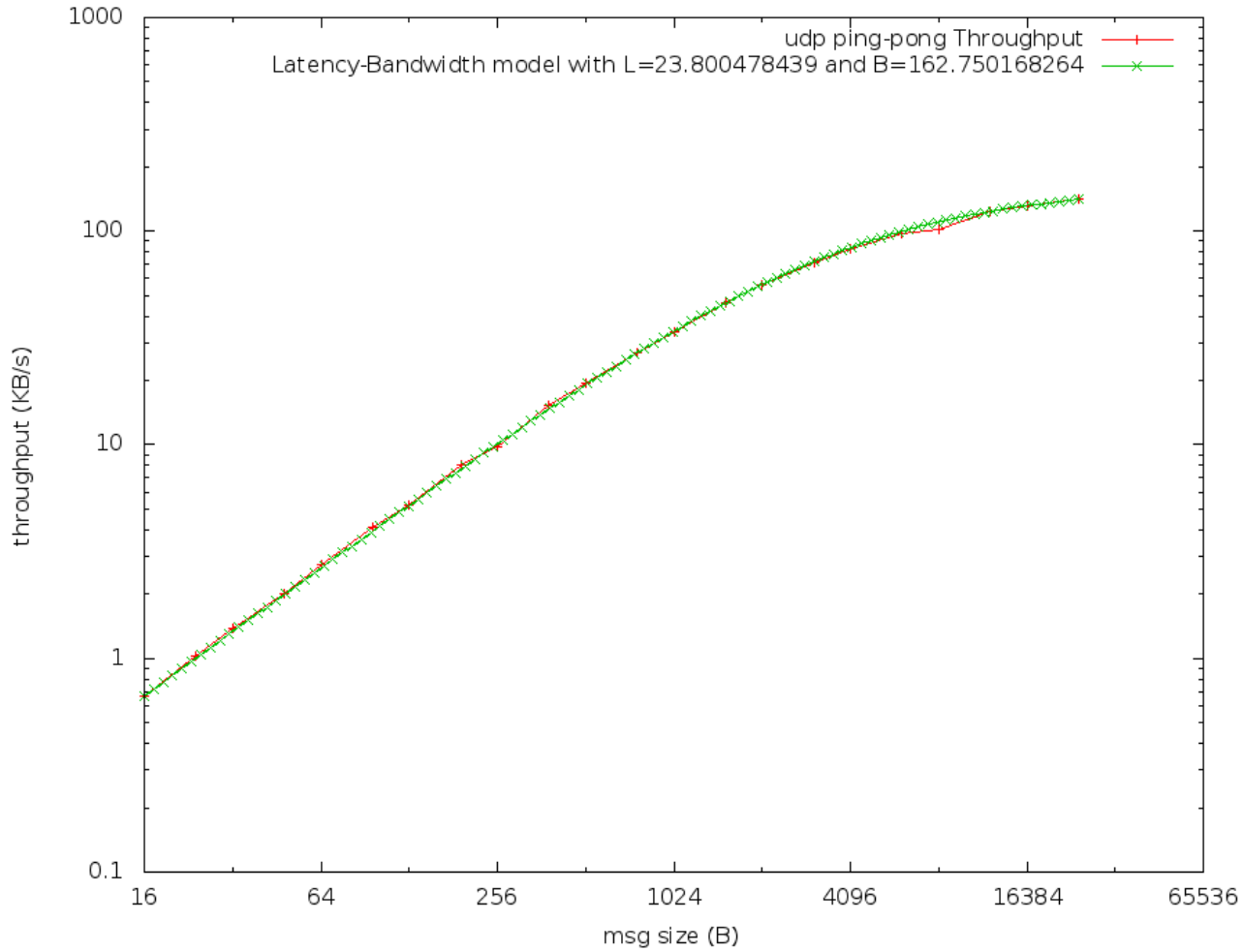

Illustrazione 4: Throughput UDP

Notiamo immediatamente che, rispetto al grafico del protocollo TCP, la curva relativa al throughput calcolato e quella relativa al throughput dedotto combaciano anche oltre i 16384B, ma quando la dimensione del messaggio è pari a 8192B il throughput calcolato è poco al di sotto del throughput dedotto: l'andamento è quasi uguale a quello previsto.

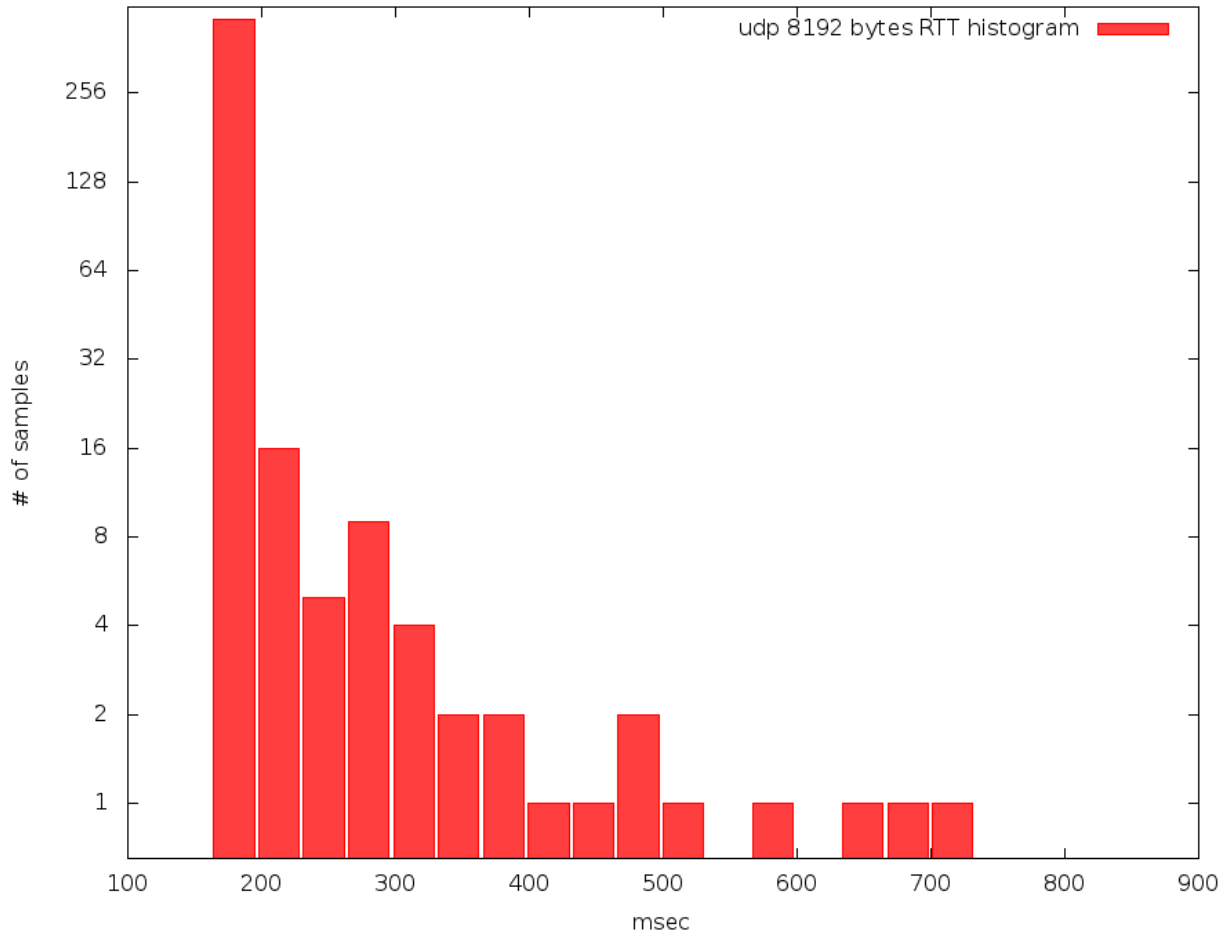

Illustrazione 5: Istogramma relativo al protocollo UDP

L'ultimo istogramma rappresenta l'adamento del Round Trip Time all'aumento dei millisecondi nel caso relativo al protocollo UDP con 8192 bytes.

CONCLUSIONI

Il laboratorio inerente al ping-pong si è rivelato molto complicato, ma nonostante ciò, in parte grazie alle informazioni trovate su AulaWeb, la guida di Beej e il manuale (comando *man*), in parte all'aiuto fornitoci dai professori, siamo riusciti a capire e commentare le funzioni all'interno dei file sorgenti, i comandi relativi alla creazione degli script bash e i grafici qui sopra riportati, purtroppo non siamo riusciti a finire il secondo e ultimo script e abbiamo deciso di non implementarlo.