Relazione laboratorio ping-pong

Relazione a cura di:
Brandi Eugenio S4064172
Nolberto Felix Cruces Hidalgo S4056658
Pellaco Francesco S4228743

Client UDP

L'UDP ping, per comunicare col server remoto, deve effettuare due tipi di connessione:
-> una connessione TCP per l'autenticazione;
-> una connessione UDP per lo scambio effettivo di messaggi.
Per essere lanciato, l'UDP ping richiede il passaggio di alcuni parametri:
-> indirizzo ip del server;
-> porta;
-> #byte da inviare;
-> ripetizioni (opzionale, se omesso sarà impostato ad un valore di default).

Per effettuare la prima connessione, si è dovuto dunque creare un socket TCP. Per fare ciò, si è settata una struttura base `gai_hints`. Per creare il socket si è impostata la struttura con i seguenti parametri:
-> `gai_hints.ai_family = AF_INET;` (indica la famiglia del protocollo, in questo caso ipv4)
-> `gai_hints.ai_socktype = SOCK_STREAM;` (indica la tipologia del socket)
-> `gai_hints.ai_protocol = IPPROTO_TCP.` (indica il protocollo usato)

Attraverso la syscall `getaddrinfo()` si creerà la struttura usata per creare il socket e per la connessione. A differenza di quella sopra, questa conterrà anche i parametri specifici del server da contattare.
La creazione del socket, viene fatta attraverso la syscall `socket()` la quale in base ai parametri passati creerà il socket necessario. Il valore ritornato, se diverso da -1, sarà il file descriptor che rappresenterà il nostro socket. La connessione viene gestita dalla syscall `connect`(). Se tutto è andato a buon fine ora, l'udp_ping è pronto a spedire il primo messaggio al server e fa ciò tramite la syscall `write`(). Una volta spedita la richiesta, il client si metterà in attesa della risposta e tramite la syscall `read`() leggerà la risposta. Questi primi due messaggi, fungono da autenticazione e sono messaggi pre accordati. Se il client o il server non dovessero riconoscere il messaggio ricevuto ci sarebbe un errore. La prima stringa spedita dal client conterrà il numero di byte da inviare e il numero di ripetizioni. Ora che l'autenticazione è avvenuta con successo, si può chiudere la prima connessione ed aprire la seconda.
Per effettuare la seconda connessione, si sono dovuti rifare alcuni passaggi elencati sopra, e ad occuparsi di ciò è la funzione `prepare_udp_socket()` che ritornerà il socket da usare per questa connessione o un errore.
All'interno della funzione, viene settata la struttura con i seguenti parametri:
-> `gai_hints.ai_family = AF_INET;` (indica la famiglia del protocollo, in questo caso ipv4)
-> `gai_hints.ai_socktype = SOCK_DGRAM;` (indica la tipologia del socket)
-> `gai_hints.ai_protocol = IPPROTO_UDP.` (indica il protocollo usato)

Anche in questo caso, vengono usate le syscall `getaddrinfo`() per indicare l'ip e la porta del server, la `socket`() per creare il socket e la `connect`() per collegarsi al server. Tramite la syscall `fcntl`() abbiamo cambiato un parametro del socket rendendolo non bloccante. La gestione dell'invio dei messaggi, viene delegato alla funzione `do_ping`() alla quale bisogna specificare la dimensione dei messaggi, il numero delle ripetizioni, un array di caratteri (il messaggio da inviare), il socket e un timeout.

Lo scopo di questa funzione è quella di spedire e ricevere i messaggi e di calcolare, tramite alcune funzioni i dati relativi alla trasmissione. Il messaggio viene "formato" scrivendo su di esso il numero delle ripetizioni più un "\n". Prima di inviare il messaggio, l'udp_ping si salva il tempo di invio utilizzando la funzione `clock_gettime`() e attraverso la syscall `send`() invia il messaggio a server.
Con la syscall `recv`(), l'udp_ping va a leggere la risposta del server, e dato che il socket non è autobloccante, si è dovuto specificare alla `send`() il flag MSG_DONTWAIT. Dopo aver "ricevuto" la risposta dal server, l'udp_ping si salva il tempo di arrivo tramite un'altra `clock_gettime`(). A questo punto viene calcolato il RTT. Nel caso il S.O. avesse bloccato la ricezione, si proverebbe a rileggere il messaggio del server sino allo scadere di un timeout prestabilito, e ad ogni tentativo si andrebbe a modificare il tempo di ricezione e di conseguenza RTT. Invece se a fallire fosse la `recv()` si interromperebbe l'esecuzione. La funzione ritornerà per ogni pacchetto spedito il quantitativo di tempo impiegato a spedire e ricevere il messaggio. Alla fine delle ripetizioni, verrà stampata una statistica con i tempi ottenuti.

Client TCP

Il TCP ping, per comunicare col server remoto, a differenza dell'UDP, deve effettuare una sola connessione:

-> una connessione TCP per l'autenticazione e per lo scambio effettivo dei messaggi;

Per essere eseguito, il TCP ping richiede il passaggio di alcuni parametri:

-> indirizzo ip del server;
-> porta;
-> #byte da inviare;
-> ripetizioni (opzionale, se omesso sarà impostato ad un valore di default)

Il TCP ping, per creare la connessione e scambiare i messaggi dovrà fare le stesse operazione l'udp_ping. Il client ping TCP crea un socket usando configurazioni specifiche e definite in questo caso dal programmatore. Queste specifiche riguardano per esempio l'uso di un socket di tipo IPv4, che, essendo TCP usa il protocollo SOCK_STREAM il quale garantisce stream affidabili e connection-based. Questo avviene tramite l'utilizzo della struttura `gai_hints`:

-> `gai_hints.ai_family = AF_INET;` <- IPv4
-> `gai_hints.ai_socktype = SOCK_STREAM;`
-> `gai_hints.ai_protocol = IPPROTO_TCP;`

Il client cerca di ottenere l'indirizzo IP e la porta al quale deve connettersi, informazioni precedentemente inserite da riga di comando, utilizzando la syscall `getadrrinfo`() e passa queste informazioni alla struttura `server_addrinfo`, utile alla creazione del socket. A questo punto il programma genera il socket tramite la syscall `socket`(), prova ad effettuare la connessione usando la syscall `connect`(). Attraverso la syscall `write`(), il TCP ping scrive una richiesta sul socket. In caso positivo, il pong server risponde positivamente e accetta la connessione; si procederà quindi al ping con conseguente restituzione dei pacchetti e tempo di invio-ricezione. La funzione che si occuperà di effettuare il ping ed inviare il messaggio, che avrà la stessa formattazione dell'UDP ping, è la funzione `do_ping`(), la quale richiede la grandezza del messaggio, il numero di ripetizioni, il messaggio e il socket. La funzione salva all'inizio del buffer il tempo corrente, successivamente invia il messaggio attraverso il socket, aspetta la risposta, salva il tempo di risposta e per finire lo restituisce.

Server

Il compito del server_pong e' quello di restare in attesa di possibili richieste di connessioni. Per la struttura del nostro server, accetta solo due tipi di connessione:

-> TCP;
-> UDP.

Il server, in ambo i casi, richiede una connessione di autenticazione basata su protocollo TCP; dopo questa fase di autenticazione, differenzierà le richieste.

Per la prima connessione è necessario creare un socket passivo in grado di attendere una richiesta: per fare ciò, si è settata una struttura base `gai_hints` con i seguenti parametri base:

-> `gai_hints.ai_family = AF_INET;`
-> `gai_hints.ai_socktype = SOCK_STREAM;`
-> `gai_hints.ai_flags = AI_PASSIVE;`
-> `gai_hints.ai_protocol = IPPROTO_TCP;`

Attraverso la syscall `getaddrinfo()` si creerà la struttura usata per creare il socket e per la connessione. A differenza di quella sopra, questa conterrà anche i parametri necessari per essere contattati dai client.
La creazione del socket, viene fatta attraverso la syscall `socket()` la quale in base ai parametri passati creerà il socket necessario. Il valore ritornato, se diverso da -1, sarà il file descriptor che rappresenterà il nostro socket. L'associazione tra la porta e il socket verrà eseguita attraverso la syscall `bind()`. Attraverso la syscall `listen()` definiamo le specifiche di attesa del socket. Dopo aver settato i vari parametri del socket, possiamo procedere con la syscall `accept()`, la quale aspetterà una connessione per procedere e accetterà il numero massimo di connessioni definite dalla `listen()`.
Ad autenticazione riuscita, il server attraverso l'analisi della richiesta differenzierà le due tipologie di connessione differenziando il caso TCP dall'UDP.

Caso TCP
A questo punto il server invierà un messaggio di conferma sulla connessione TCP creata prima; per lo scambio dei messaggi tra client e server, verrà usata la stessa connessione. Il server, leggerà il messaggio attraverso la syscall `getc()`, dopo aver verificato la correttezza del messaggio, risponderà al client attraverso una `write_all()`. Alla fine dello scambio dei messaggi, il server chiuderà la connessione.

Caso UDP
In questo caso, il server preparerà un nuovo socket per supportare la connessione UDP. Per creare questa tipologia di connessione, si è usata la struttura `gai_hints` impostata come sopra cambiando soltanto il socket e il protocollo usato :

-> `gai_hints.ai_protocol = IPPROTO_UDP;`
-> `gai_hints.ai_socktype = SOCK_DGRAM;`

Anche in questo caso, vengono usate le syscall `getaddrinfo()` per indicare la porta d'ascolto, la `socket()` per creare il socket e la `bind()` per associare la porta al socket.
A questo punto il server invierà un messaggio di conferma sulla connessione TCP specificando il numero della porta e dopodiché la chiuderà. La connessione appena creata servirà per l'effettivo scambio di messaggi. Il server leggerà la richiesta del client tramite la syscall `recvfrom()` e risponderà tramite la syscall `sendto()`. Attraverso un'analisi della richiesta, il server riconoscerà la corretta sequenza dei pacchetti e la correttezza del messaggio.

Scripts

Dato che gli scripts non permettono di effettuare i calcoli in float, è stato necessario usare il comando bc per effettuare tutti i calcoli necessari.

Throughput.bash

throughput.bash è un programma che dato un protocollo (TCP o UDP) stampa il grafico del throughput in relazione al numero dei pacchetti inviati e lo mette in relazione con il modello banda latenza.

Questo script si aspetta in input un parametro (TCP o UDP) da linea di comando.
Per la corretta esecuzione, esso necessita anche dell'esistenza della cartella data e del file throughput.dat da analizzare. In base al parametro passato sceglierà tcp_throughtput.dat o udp_throughtput.dat.
In entrambi i casi il procedimento è analogo:
lo script estrae la prima (con il comando head) e l'ultima riga (con il comando tail) del file per poi andare ad estrarre il primo e l'ultimo valore delle due righe estratte (DimMinByte e DimMaxByte dimensione dei pacchetti, ThroughputMin e ThroughputMax relativi throughput).
Successivamente, lo script esegue una serie di calcoli per ricavarsi i valori base per la costruzione del modello banda latenza:
`->DelayMin=$(echo ${DimMinByte}/${ThroughputMin} | bc -l)` rappresenta il delay della prima misura
`->DelayMax=$(echo ${DimMaxByte}/${ThroughputMax} | bc -l)` rappresenta il delay della seconda misura
`->differenceN=$(echo ${DimMaxByte}-${DimMinByte} | bc -l)`
`->differenceD=$(echo ${DelayMax}-${DelayMin} | bc -l)`
`-> B=$(echo "scale=5; ${differenceN}/${differenceD}" | bc -l)` rappresenta la banda
`-> L=$(echo "scale=9; ((-${DimMinByte})/${B}+${DelayMin})/1" | bc -l)` rappresenta la latenza

Lo script, se assente, crea una cartella dentro la cartella data, e cancella il file che sta creando se presente.
Per la creazione del grafico, utilizza la funzione `gnuplot` che impostando i parametri in maniera adeguata, ci permette di osservare e di capire i dati riportati. Per il calcolo del modello banda-latenza si è specificato la funzione da calcolare: f(x) = x / ($L + x / $B), mentre per la rappresentazione del ping-pong Throughput si è specificato il file e le colonne da importare. Alla fine viene visualizzato il grafico creato.

Rtt.bash

rtt.bash è un programma che dato un protocollo e una dimensione genera due file:
-> un file.dat contenente i dati calcolati;
-> un file.png che rappresenta i dati calcolati in un grafico.

Questo script si aspetta due parametri da linea di comando: il nome del protocollo e il numero di pacchetti.
Per la corretta esecuzione, questo script necessita dell'esistenza della cartella data e del file .out da analizzare. In base ai parametri passati sceglierà il file.out da analizzare. Il file restituito sarà composto da 4 colonne:

1) N delle ripetizioni;
2) Simple RTT;
3) estimatedRTT;
4) variabilityRTT.

Dato che si conosceva la formattazione delle stringhe, per recuperare i dati richiesti abbiamo potuto usare comandi specifici per questo scopo. Sapendo di dover usare una cut, abbiamo usato il comando tr per essere sicuri che le parole all'interno del file fossero divise esattamente da uno spazio.
Per il numero delle ripetizioni ci serviva recuperare solo il valore massimo, e per fare ciò, abbiamo filtrato il file tramite una grep, dopo di che attraverso una tail abbiamo estratto la riga interessata e con una cut abbiamo estratto l'ultimo valore; per estrarre gli sampleRTT si è proceduto in una maniera del tutto analoga: con una grep si è filtrato il contenuto e poi con la cut si sono estratti i dati voluti.

Gli atri due valori, li abbiamo dovuti calcolare applicando in un ciclo le seguenti formule:
-> estimatedRTTi = 0.875*estimatedRTT(i-1) + 0.125*sampleRTTi;
-> variabilityRTTi = 0.75*variabilityRTT(i-1) + 0.25*|sampleRTTi - estimatedRTTi|.

Essendo che per calcolare un valore, in entrambi i casi, ci si basa sul valore precedente si è provveduto ad inizializzare separatamente i primi valori :
-> estimatedRTT=$(echo ${SimpleRtt[0]} | bc -l);
-> variabilityRTT=$(echo 0 | bc -l).

Lo script, crea una cartella dentro la cartella data, se assente, dopo di che cancellerà se presente la cartella NomeProtocollo_Dimesione con il suo contenuto per poi ricrearla vuota. Al suo interno verranno salvati il file dei dati e il grafico. Per la creazione del grafico, si utilizza la funzione `gnuplot` che, impostando i parametri in maniera adeguata, ci permette di osservare e di capire i dati riportati.
Per la rappresentazione dei 3 grafici richiesti si è specificato il file da prendere, specificando ogni volta le colonne da utilizzare.

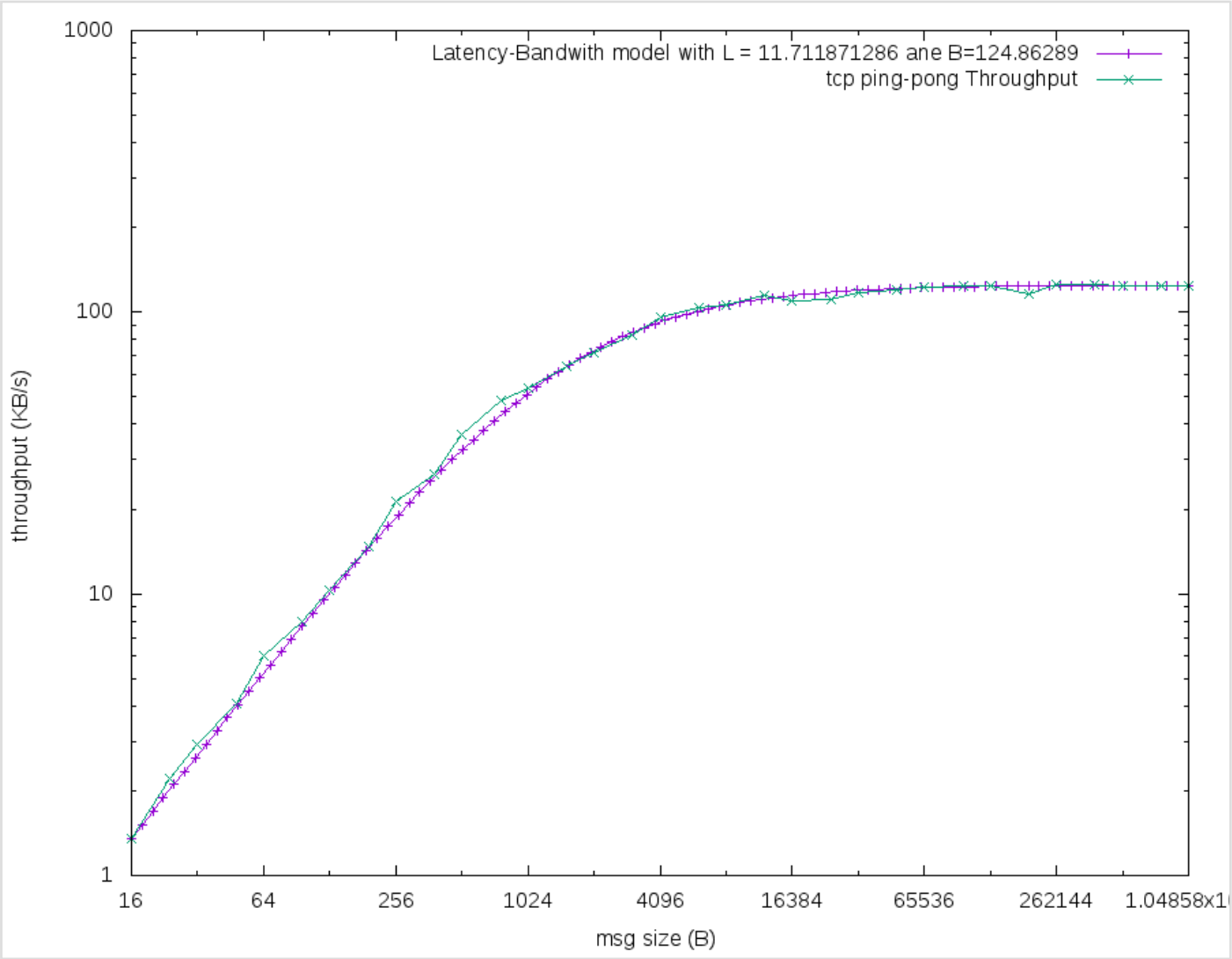
Grafici

In questi grafici, throughput TCP e UDP, sono riportati due tipologie di campioni :

-> il campione teorico rappresentato dal modello banda-latenza viene calcolato in seguito al reperimento di due valori sperimentali, rispettivamente il più piccolo ed il più grande e rispettivi throughput (linea viola);
-> il campione sperimentale "raccolto" durante la fase di ping del server remoto (linea verde).

Il modello teorico, approssima l'andamento della rete nell'intervallo dei test fatti e più i dati sperimentali si avvicinano al modello più la rete è stabile.
Dai grafici, ci aspettiamo che per dimensioni dei pacchetti piccoli, l'invio dei messaggi non sfrutti a pieno la nostra banda disponibile, mentre per pacchetti sempre più grandi, sfrutti sempre in modo migliore la banda, fino a tendere alla banda massima rilevata nel periodo di campionamento.

Throughput TCP

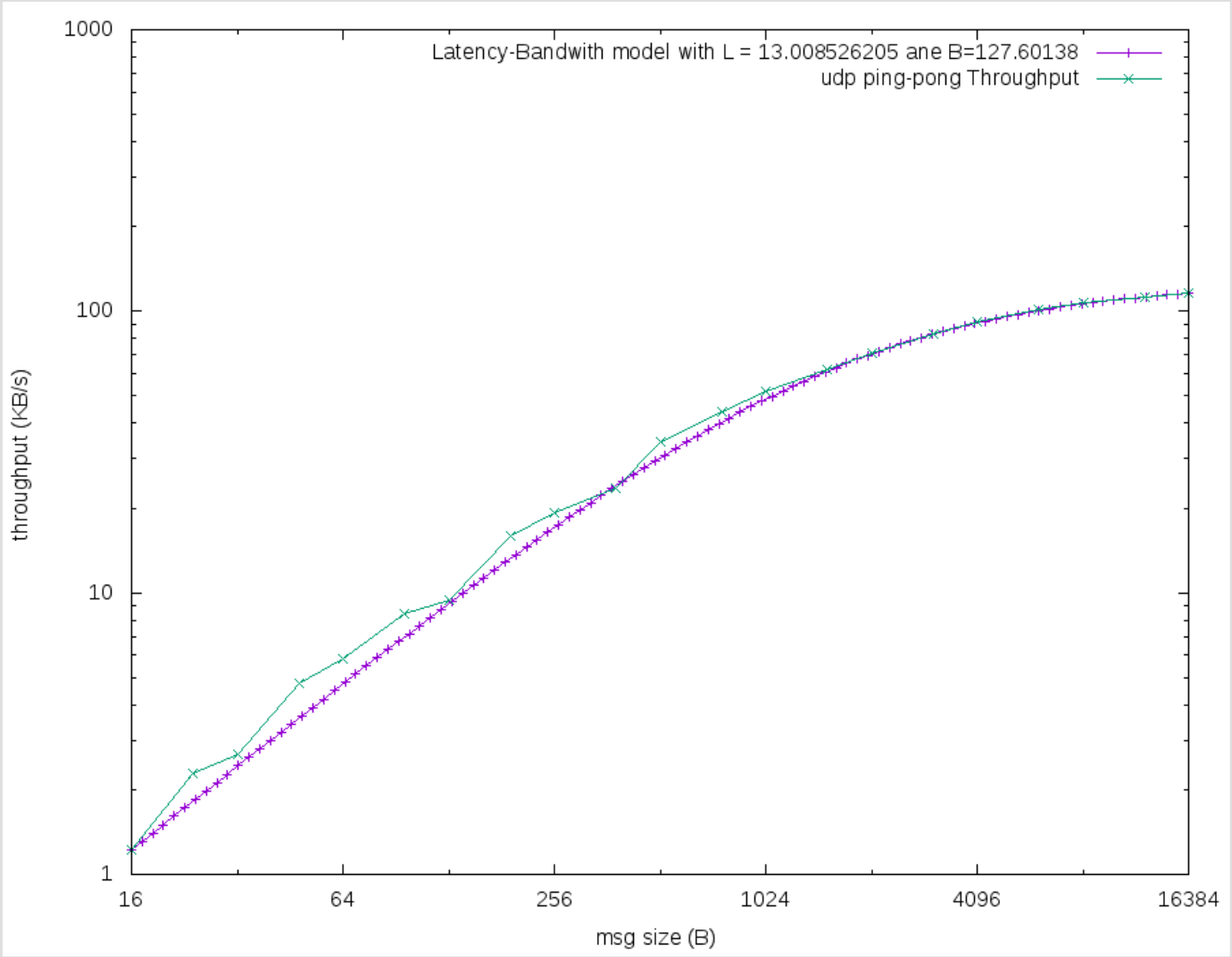


Con una prima analisi del grafico, possiamo osservare che, come da noi atteso, per pacchetti piccoli, la banda disponibile non viene sfruttata al massimo, e all'aumentare della dimensione dei pacchetti aumenta anche l'utilizzo della banda disponibile che in questo caso era 124.86289.
Come possiamo notare, nonostante i campioni sperimentali oscillino intorno a quelli teorici, le due linee rimangono sempre simili, ciò significa che durante la fase di ping la rete usata è stata abbastanza stabile.

Analizzando in modo più accurato i dati sperimentali, si può notare come la Banda (B), non rappresenti il valore massimo.
93216 126.213 125.914
524288 126.196 124.129
786432 125.516 124.714
1048576 125.776 124.689 (ultimo valore)

Questo è dovuto ad un errore della stima della banda massima, probabilmente alla fine della rilevazione dei dati sperimentali la banda disponibile è calata poichè richiesta anche da un altro client. Siccome è impossibile spedire ad una velocità maggiore della massima, per correggere questo errore, si potrebbe ricalcolare il modello banda-latenza impostando alla banda il valore massimo rilevato.

Throughput UDP



Per quanto riguarda il grafico dell'UDP, l'analisi è analoga :
con una prima analisi del grafico, possiamo osservare che, come da noi atteso, per pacchetti piccoli, la quantità di banda sfruttata sia minima, e all'aumentare della stessa aumenta anche l'utilizzo di quella disponibile che in questo caso era 127.60138.
Come possiamo osservare, nonostante i campioni sperimentali oscillino intorno a quelli teorici, le due linee rimangono sempre simili, ciò significa che durante la fase di ping la rete usata è stata abbastanza stabile. A differenza del TCP, la banda massima non viene mai superata quindi in questo caso, non c'è stato alcun errore nella stima della banda.

RTT

In questi grafici, RTT TCP e UDP, sono riportati tre tipologie di campioni:

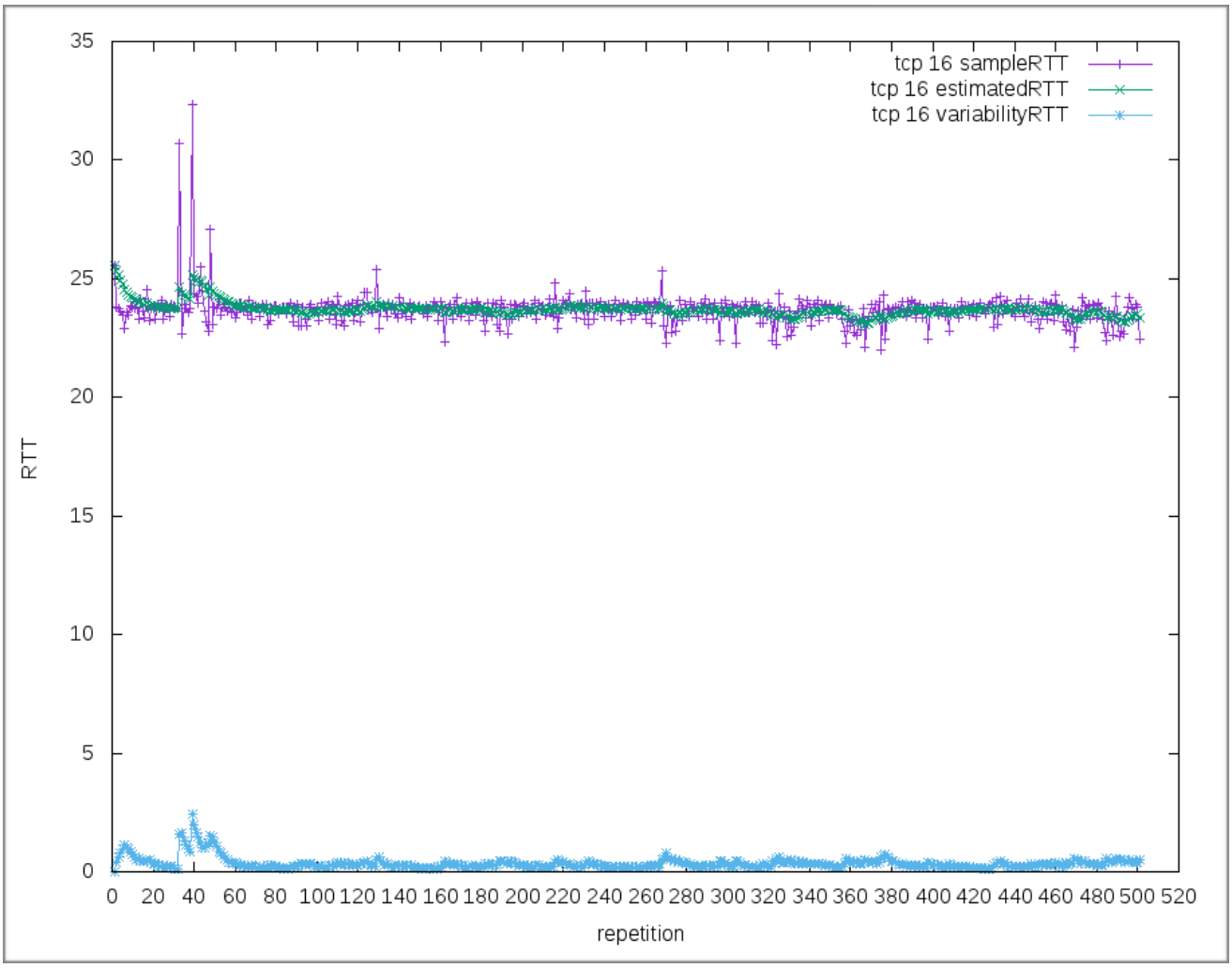
-> il campione teorico rappresentato dall'estimated RTT (linea verde);
-> il campione teorico rappresentato dal variability RTT (linea azzurra);
-> il campione sperimentale sampleRTT "raccolto" durante la fase di ping del server remoto (linea verde).

Dall'analisi di questi grafici si può dedurre che i pacchetti più piccoli impiegano meno tempo a tornare al client rispetto quelli grandi, inoltre i dati UDP dovrebbero risultare più veloci rispetto a quelli TCP. Dato che il RTT non dipende dal numero di ripetizioni, ci si aspetterebbe dei campioni sperimentali tutti uguali e di conseguenza anche quelli teorici.
Quello che probabilmente si otterrà, saranno dei campioni sperimentali che oscilleranno tra un minimo ed un massimo (ci aspettiamo oscillazioni piccole). Queste oscillazioni sono dovute al congestionamento della rete. EstimatedRTT ci permette di stimare il campione sperimentale ammortizzando le oscillazioni, ossia ammortizzando i cambi di congestione della rete. Il campione sperimentale variability RTT ci da una stima di quanto i due campioni descritti sopra si discostano tra di loro.

RTT TCP 16 - 1048576

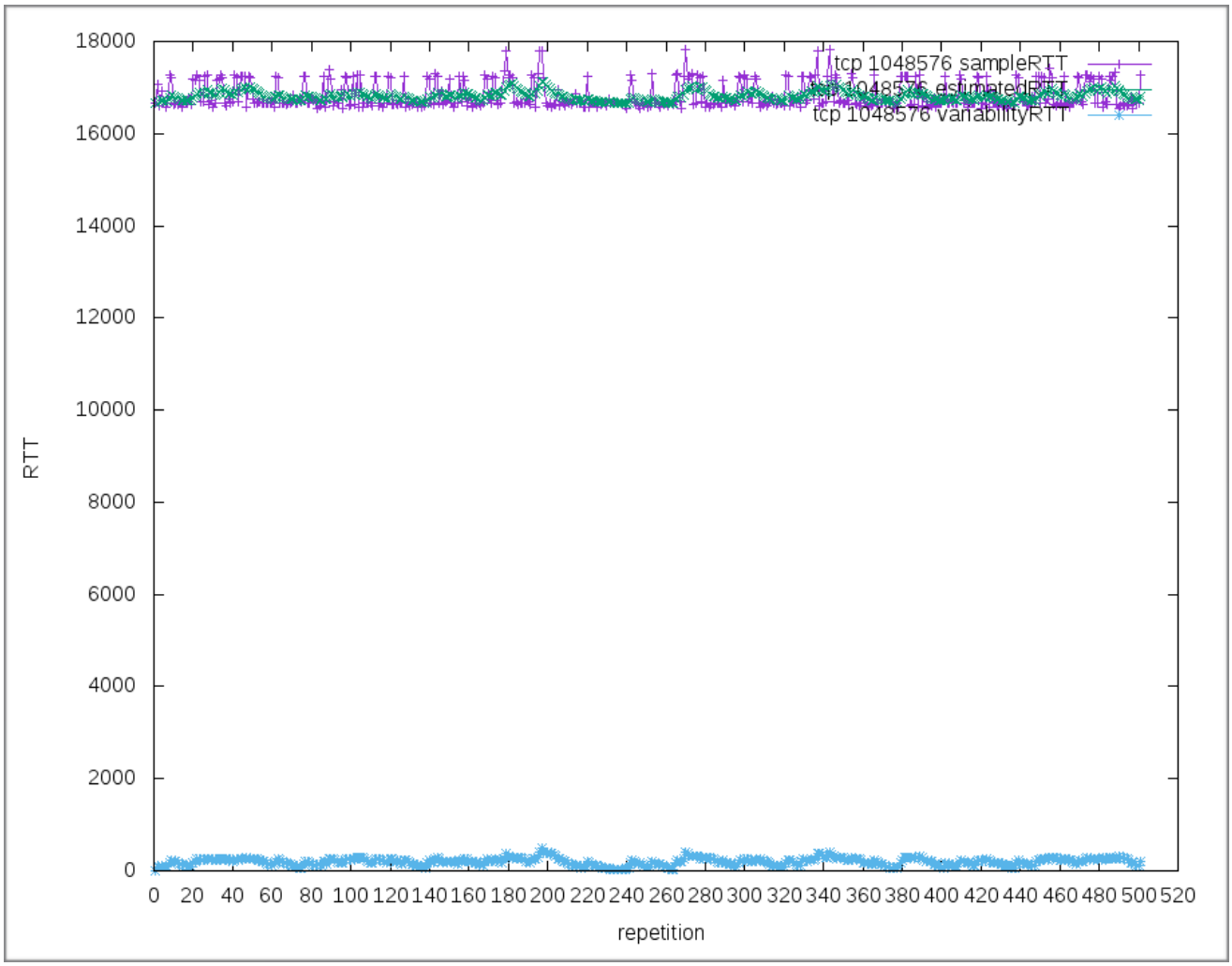
Osservando questi due grafici, possiamo osservare che, come da noi atteso, i pacchetti più grandi impiegano più tempo a tornare al mittente.

RTT TCP 16



Osservando il grafico, possiamo notare che i campioni sperimentali assumono valori diversi nell'intervallo di campionamento. Inoltre, possiamo osservare come la stima estimatedRTT non risenta più di tanto della congestione della rete. Infatti, a grandi picchi del campione sampleRTT corrispondono piccoli picchi del campione estimatedRTT. Infine, la stima della loro differenza è spesso vicino a zero, solo in pochi punti i campioni di distaccano molto, da questi campioni, possiamo osservare che durante il campionamento dei dati sperimentali, la congestione della rete ha avuto alcuni cambiamenti significativi.

RTT TCP 1048576

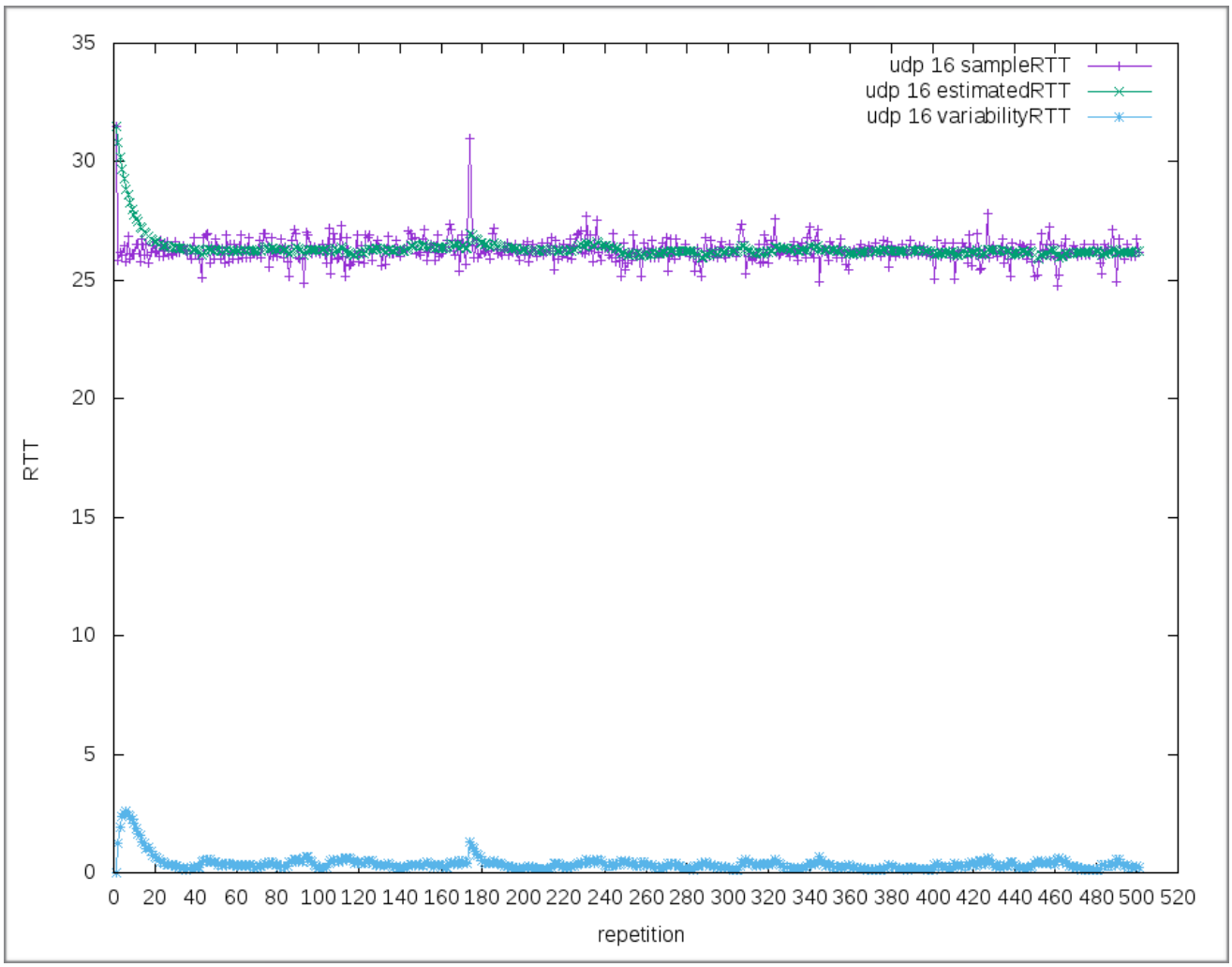


Possiamo osservare che i campioni sperimentali assumono valori diversi nell'intervallo di campionamento. Come in precedenza possiamo di nuovo notare come la stima estimatedRTT non risenta più di tanto della congestione della rete, a grandi picchi del campione sampleRTT corrispondono piccoli picchi del campione estimatedRTT. Infine, la stima della loro differenza è spesso vicino a zero e ciò significa che durante il campionamento dei dati sperimentali, la congestione della rete non ha avuto molti cambiamenti.

RTT UDP 16-16384

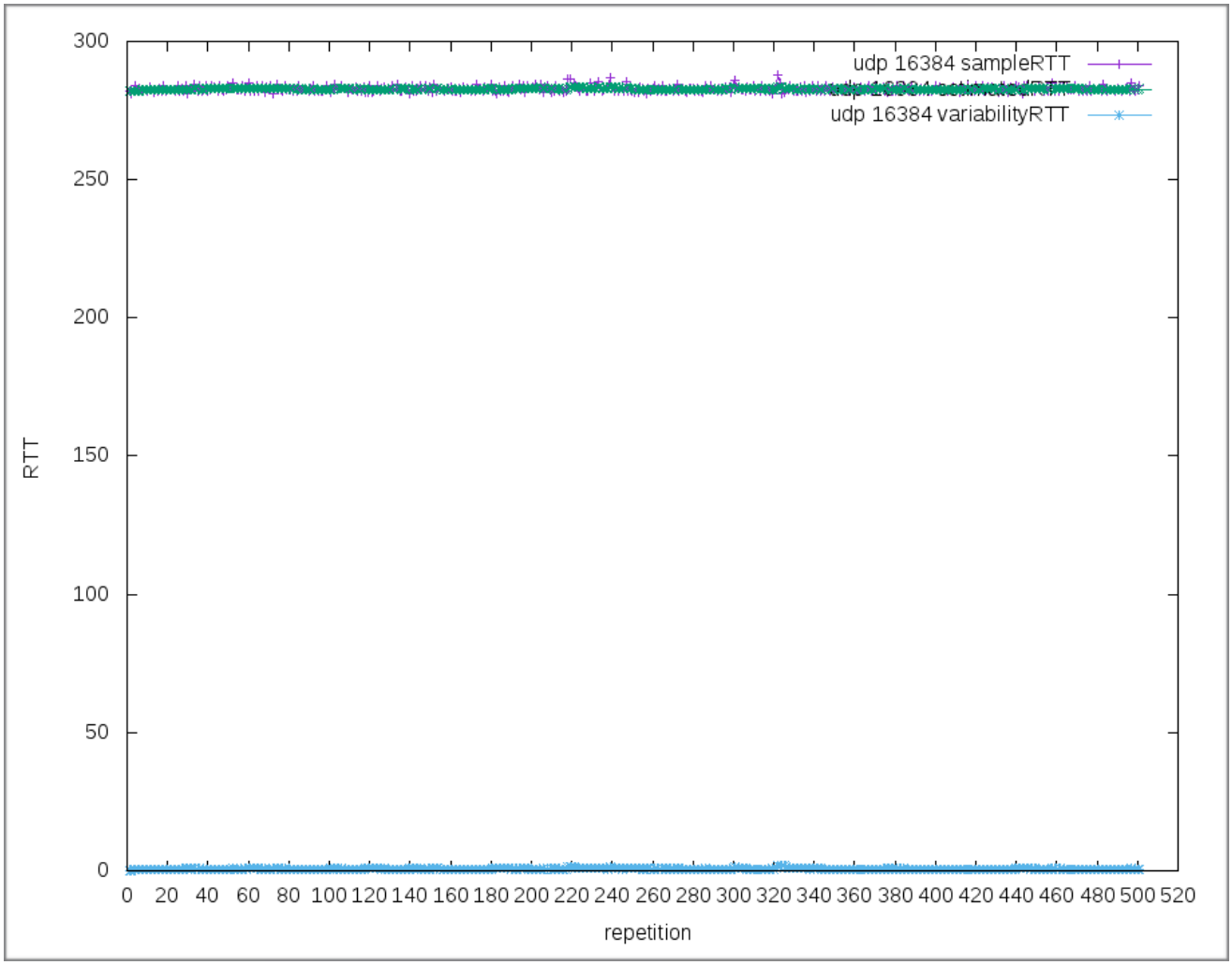
Per quanto riguarda il grafico dell'UDP, l'analisi è analoga:
osservando questi due grafici, possiamo notare che, come da noi atteso, i pacchetti più grandi impiegano più tempo a tornare al mittente.

RTT UDP 16



Possiamo notare che i campioni sperimentali assumono valori diversi nell'intervallo di campionamento. Inoltre, possiamo osservare come la stima estimatedRTT non risenta più di tanto della congestione della rete. Infatti, a grandi picchi del campione sampleRTT corrispondono piccoli picchi del campione estimatedRTT.

RTT UDP 16384



Analizzando il grafico, possiamo notare che i campioni sperimentali assumono valori diversi nell'intervallo di campionamento (anche se poco percettibili dal grafico). Possiamo anche osservare come la stima estimatedRTT non abbia grosse variazioni. Infatti, nei campioni sampleRTT non ci sono picchi abbastanza alti da influenzare i campioni estimatedRTT.
Infine, la stima della loro differenza è pressoché zero, e ciò è dovuto dal fatto che nei campioni stimati non siano presenti oscillazioni significative.
Per concludere, da questi campioni, possiamo osservare che durante il campionamento dei dati sperimentali, la congestione della rete non ha avuto cambiamenti significativi.

Conclusioni:

Sono state riscontrate alcune difficoltà:

-> Nella comunicazione col server specificato, e di conseguenza una difficoltà nel prendere i dati voluti, si è deciso quindi di creare un altro server per eseguire le operazioni richieste;

-> Nel completare il primo codice richiesto, e negli script poiché linguaggi pressoché nuovi;

-> Nel capire il corretto funzionamento di alcune nuove funzioni.