Reti di calcolatori - Modulo Multimedia

Homework 2

Francesco Roncolato

7 Giugno 2024

Indice

1 Codice

Per lo svolgimento dell'esperimento è stato scritto del codice `python` compreso nella cartella `/code_and_results`.
Per eseguire il codice su macchina Windows:

1. accertarsi che sia installato `python` oppure `python3` (ad esempio eseguendo `python -V` o `python3 -V` su un terminale)
2. accertarsi che le librerie `matplotlib` e `numpy` siano installate (ad esempio eseguendo `pip install matplotlib` e `pip install numpy` su un terminale)
3. eseguire il programma con il comando `python hw2_roncolato.py`

Note:

- ◇ L'esecuzione di `hw2_roncolato.py` potrebbe richiedere all'incirca 2 minuti per la prima parte (calcolo del numero di nodi) mentre per seconda parte (esecuzione dei `ping` con raccolta e analisi dei dati) con i valori predefiniti di `K`, `L_byte_step` e `max_threads` (si veda la sezione parametri) anche mezz'ora. Per testare singolarmente le parti è sufficiente commentare le righe non volute nel codice di `hw2_roncolato.py`
- ◇ Il programma compone dei comandi e li manda alla shell sulla quale viene eseguito il programma python. Questo implica che, se i programmi installati hanno una sintassi diversa (in input o in output) rispetto a quella prevista su Windows impostato in lingua italiana, l'esecuzione dei comandi o il parsing dei risultati potrebbe non funzionare. Ad esempio il comando `ping` in Windows permette di ricavare il RTT individuandolo come il numero che viene dopo la stringa "`durata=`" (e questo ovviamente dipende dalla lingua del sistema), mentre in Linux sarebbe necessario andare a cercare la stringa "`time=`".
  Per eseguire il programma su una macchina Linux o con diverse impostazioni per la lingua, sarebbe quindi necessario riscrivere le funzioni contenute nel file `commands.py` e i parser contenuti nel file `parsers.py`.
- ◇ I dati riportati sono stati ottenuti su macchina dotata di Windows11 utilizzando il comando `ping` per la prima parte in quanto permette di scegliere il valore del TTL (time to live) e `psping` per la seconda parte perché permette maggiore precisione.

2 Parametri dell'esperimento

Eseguendo la prima parte attraverso la funzione `number_of_nodes_with_ping_and_traceroute` (calcolo del numero di nodi attraverso i comandi di `ping` e `traceroute`), vengono chiamate le funzioni e `find_nodes_with_traceroute` e `find_nodes_with_ping`: entrambe richiedono un `target_name` (nome o indirizzo del server a cui mandare i pacchetti) che ragionevolmente deve essere lo stesso, di default è impostato `lyon.testdebit.info`.
Le due funzioni sono chiamate in serie ma, al suo interno, `find_nodes_with_ping` parallelizza diverse chiamate al comando `ping`, ciascuna composta da un solo pacchetto di un solo byte e con un determinato valore del TTL (time to live). `number_of_nodes_with_ping_and_traceroute` accetta i parametri:

- ◇ `min_ttl`: minimo valore del parametro TTL ad essere testato, di default è impostato a 1
- ◇ `max_ttl`: massimo valore del parametro TTL ad essere testato, di default è impostato a 40 (valori più alti rendono l'esecuzione della prima parte più lunga)
- ◇ `max_workers`: numero massimo di thread lanciati parallelamente per eseguire i vari `ping`, di default è impostato a 40, in modo da poter fare contemporaneamente tutte le chiamate
- ◇ `target_name` e `source_name`: server a cui inviare i pacchetti e posizione da cui si esegue l'esperimento (quest'ultimo usato solamente per il nome del file di output)

Successivamente viene eseguita la funzione per eseguire i comandi di ping e salvare i risultati in un file locale `perform_pings_and_save_into_file` che accetta i seguenti parametri:

- ◇ `K`: numero di pacchetti mandati da ogni esecuzione del comando ping
- ◇ `min_L_byte`: minima dimensione dei pacchetti inviata, il valore di default è 10 bytes
- ◇ `max_L_byte`: massima dimensione dei pacchetti inviata, il valore di default è 1472 bytes
- ◇ `L_byte_step`: passo (misurato in byte) con cui viene incrementata la dimensione dei pacchetti (tra `min_L_byte` e `max_L_byte`) tra un'esecuzione del comando ping e la successiva
- ◇ `max_threads`: con lo stesso significato di quanto detto per la prima parte ma con valore di default 10
- ◇ `target_name`: con lo stesso significato e lo stesso valore di default usato per `number_of_nodes_with_ping_and_traceroute`

◇ `output_file`: nome del file dove vengono salvati i risultati delle oeprazioni di ping

◇ `function`: funzione definita in `commands.py`, indica quale comando usare per le operazioni di ping, di default è impostato `win_psping`

Nell'esecuzione di questa seconda parte è importante capire l'effetto che i vari parametri hanno sul tempo di esecuzione e sull'accuratezza dei risultati. Aumentare `K` rende più probabile che almeno una volta il pacchetto inviato trovi tutte le code libere e quindi migliora l'accuratezza dell'RTT minimo. Diminuire `L_byte_step` (e quindi aumentare il numero di comandi `ping` eseguiti) permette di avere più dati su cui fare l'analisi, rendendola più significativa. Queste due azioni quindi potrebbero migliorare il risultato dell'esperimento ma a costo di un'esecuzione molto più impegnativa in termini di tempo. Si può infine pensare di eseguire più comandi allo stesso momento, aumentando `max_threads`, questo permette di velocizzare l'esperimento ma incrementando il rischio di creare delle code di pacchetti in uscita che compromettano i risultati. Per questo si è deciso di impostare un valore di default per il numero di thread piuttosto basso. Il problema delle code non si pone nel caso della parte 1 ed è per questo motivo che in quel caso si è impostato un valore più alto di `max_threads`.

Infine viene chiamata `parse_ping_result_data` che legge il file generato dalla fase precedente e ne ricava una lista di `Result` (un tipo di dato definito in `utilities.py` che contiene un valore per `L_byte` e la lista `rtt_list` di valori misurati per quella particolare dimensione del pacchetto). Questa lista permette di proseguire con l'elaborazione (interpolazione lineare e calcolo del throughput) in modo molto semplice.

Le funzioni definite in `commands.py` compongono i seguenti comandi per il terminale e li eseguono:

◇ `ping` con i parametri:

- *`target_name`* per indicare a quale server inviare i pacchetti
- `-i` *`ttl`* per impostare il time to live
- `-n` *`K`* per impostare il numero di pacchetti inviati dal comando
- `-l` *`L`* per impostare la dimensione dei pacchetti inviati
- `-f` che se impostato (come di default), dice di non frammentare il pacchetto
- `>` *`result_file`* per reindirizzare l'output del comando ad un file testuale

◇ `psping` con i parametri:

- `-n` *`K`* per impostare il numero di pacchetti inviati dal comando
- `-l` *`L`* per impostare la dimensione dei pacchetti inviati

- `-nobanner` per semplificare il parsing dell'output
- *target_name* per indicare a quale server inviare i pacchetti
- `>` *result_file* per reindirizzare l'output del comando ad un file testuale

◇ `tracert` con i parametri:

- *target_name* per indicare a quale server inviare i pacchetti
- `>` *result_file* per reindirizzare l'output del comando ad un file testuale

3 Stima del numero di link attraversati

Utilizzando solamente i comandi forniti dai sistemi operativi più noti, è possibile calcolare il numero di link attraversati da un pacchetto per arrivare ad un determinato server. In particolare in questo esperimento (prima parte di `hw2_roncolato.py`) si sono confrontati i risultati ottenuti attraverso `ping` e `tracert` in Windows.

`tracert` è pensato appositamente per identificare il percorso intrapreso da un pacchetto dal client al server. Permette infatti di identificare e contare i nodi attraversati dal pacchetto. Per ricavare il valore è quindi sufficiente fare un parsing dell'output di tale programma.

`ping` invece ha uno scopo, e quindi un output, molto diverso. Non fa infatti vedere i nodi attraversati dal pacchetto. Per poter ricavare il numero di nodi è quindi necessario inviare diversi `ping` (anche parallelamente) variando il valore di `ttl`. In particolare sono stati inviati (parallelamente) dei `ping` con valori di TTL da 1 a 40 e successivamente analizzati i risultati per capire fino a quale valore del parametro l'invio fallisce.

I risultati ottenuti raggiungendo `lyon.testdebit.info` dalla rete `eduroam` del DEI sono:

```
  Command   | # Nodes  | Ex. time
    ping    |    14    |  3.919   s
 traceroute |    14    |  77.515  s
```

mentre per raggiungere lo stesso server dalla mia rete domestica:

```
Could take around 2 minutes
  Command   | # Nodes  | Ex. time
    ping    |    18    |  4.007   s
 traceroute |    18    | 115.455  s
```

Si noti che questi valori potrebbero cambiare con il tempo.

4 Andamento dell'RTT

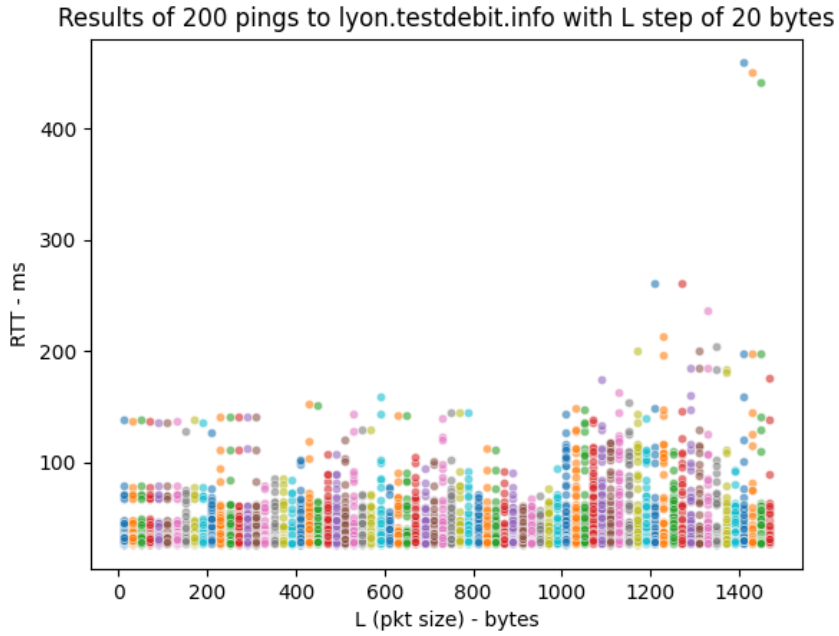
La seconda parte di `hw2_roncolato.py` esegue parallelamente numerose chiamate `ping`, variando la dimensione del pacchetto inviato. Tutti i valori dell'RTT vengono salvati in un file e successivamente analizzati.

Per ogni valore di `L` vengono calcolati i valori dell'RTT minimo, massimo, medio e la deviazione standard. Per i minimi si è inoltre calcolata la regressione lineare[1] e l'errore in valore assoluto tra ciascun minimo e tale retta. I risultati sono quelli mostrati nei grafici sotto.

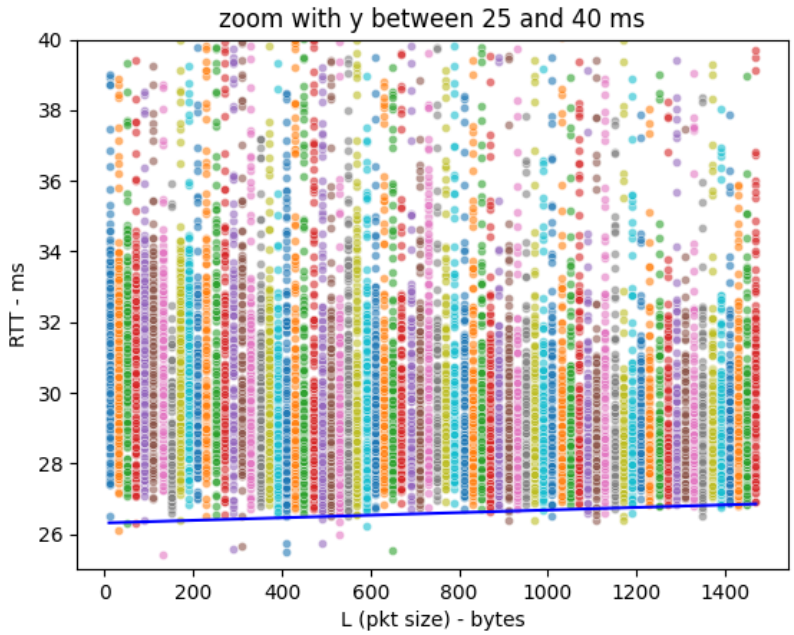
Si può notare come la maggior parte dei pacchetti abbiano compiuto il percorso di andata e ritorno in meno di 100 ms con qualche outlyer oltre i 400 ms.

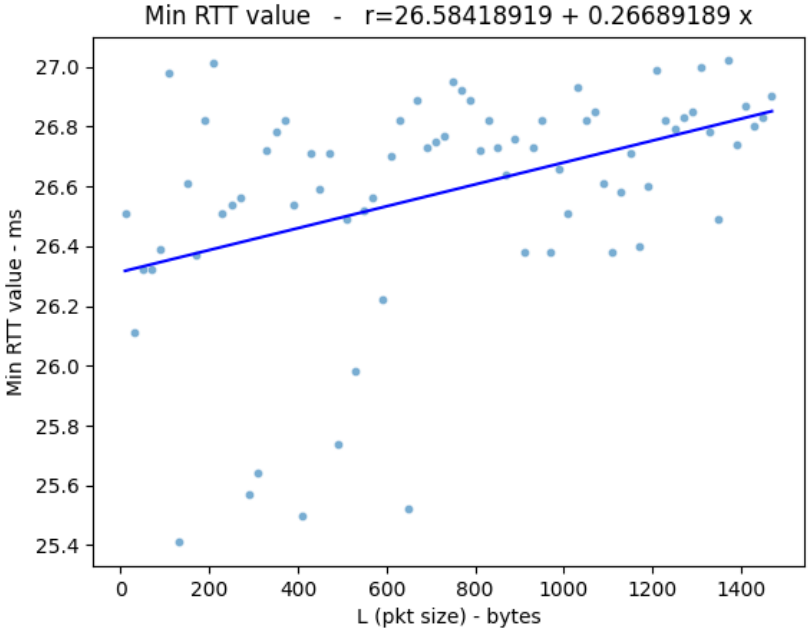
I minimi, come evidenziato dalla retta di regressione, hanno un andamento crescente con il valore di `L`.

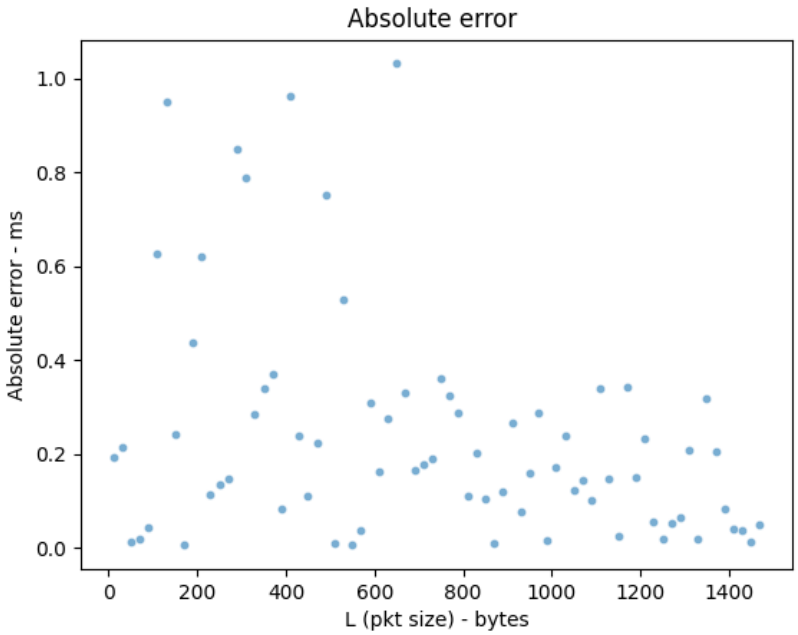
4.1 Risultati DEI-lyon.testdebit.info

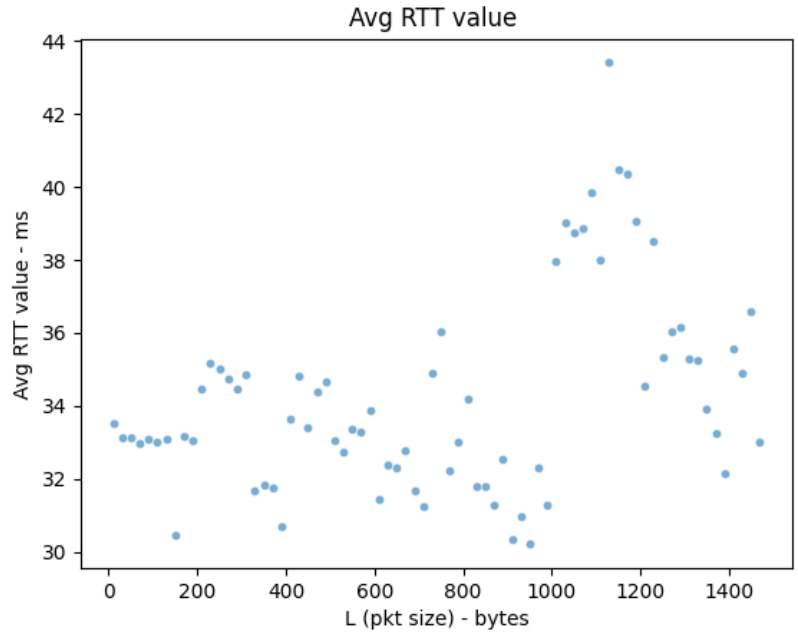


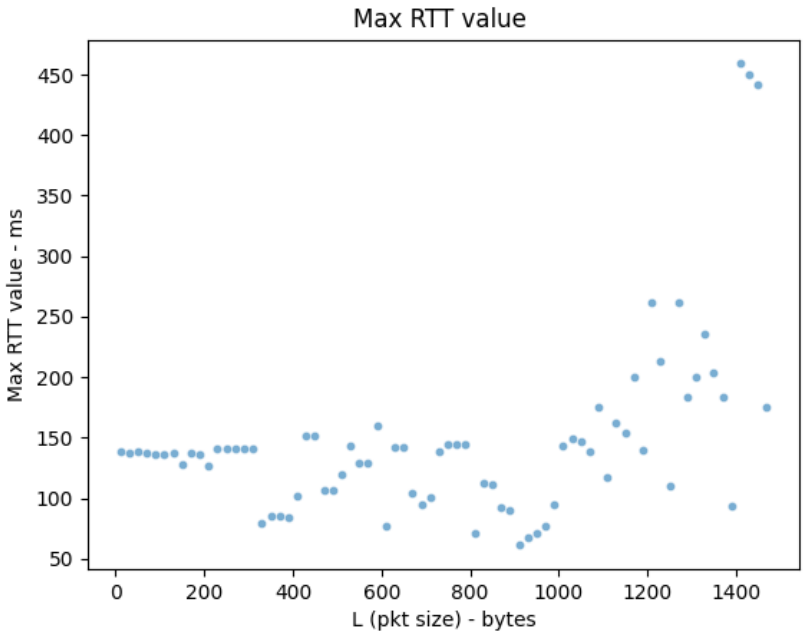
[1] attraverso la funzione `Polynomial.fit` del submodule `numpy.polynomial` che usa il metodo dei minimi quadrati

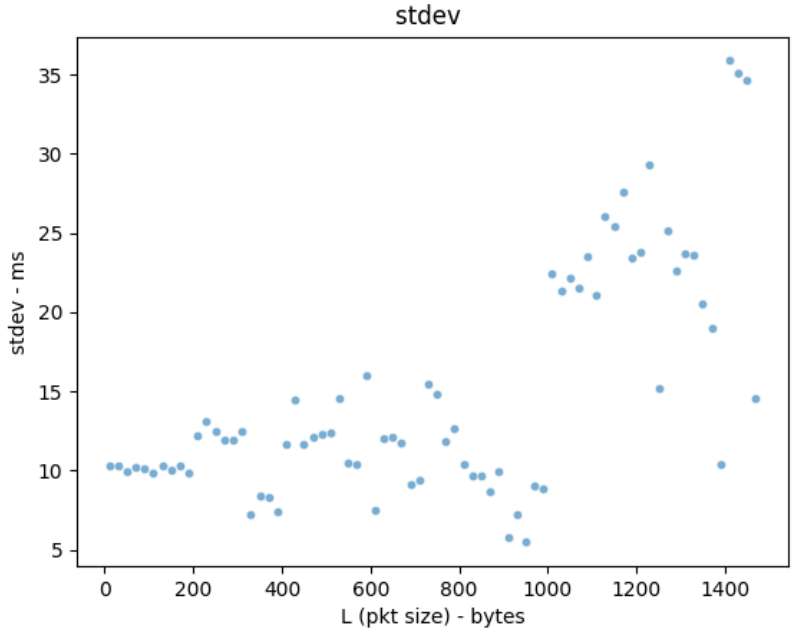

zoom with y between 25 and 40 ms
RTT - ms
L (pkt size) - bytes
40
38
36
34
32
30
28
26
0
200
400
600
800
1000
1200
1400


Min RTT value - r=26.58418919 + 0.26689189 x
Min RTT value - ms
L (pkt size) - bytes
27.0
26.8
26.6
26.4
26.2
26.0
25.8
25.6
25.4
0
200
400
600
800
1000
1200
1400

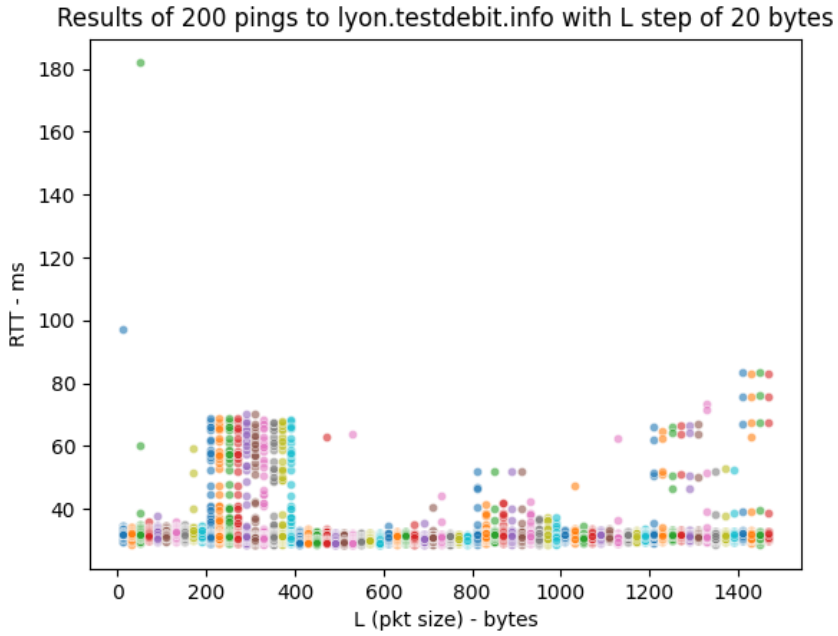

Absolute error
Absolute error - ms
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400
L (pkt size) - bytes

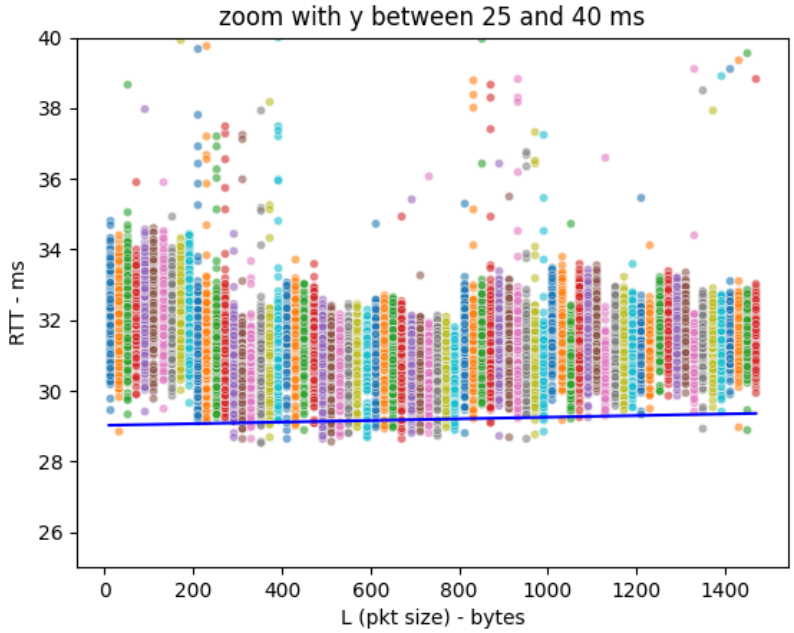

Avg RTT value
Avg RTT value - ms
44
42
40
38
36
34
32
30
0
200
400
600
800
1000
1200
1400
L (pkt size) - bytes

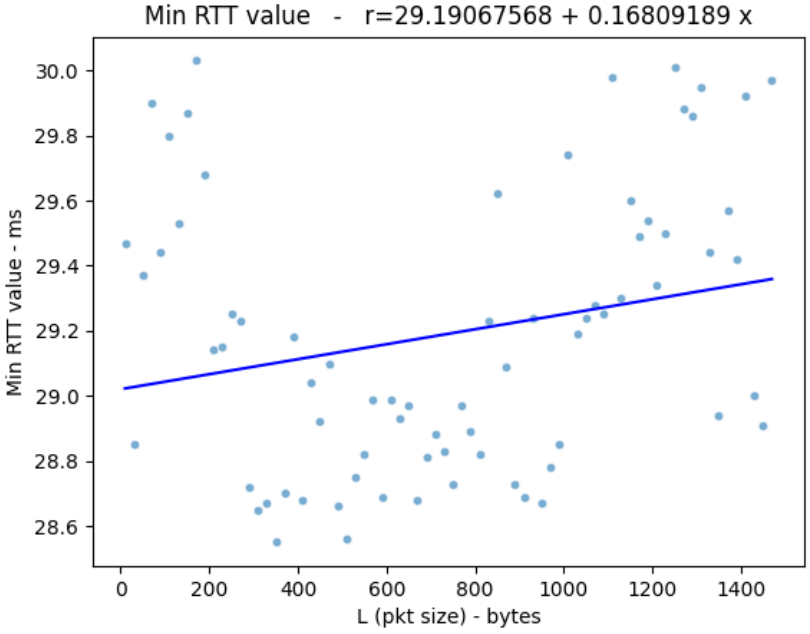

Max RTT value
Max RTT value - ms
L (pkt size) - bytes
50
100
150
200
250
300
350
400
450
0
200
400
600
800
1000
1200
1400

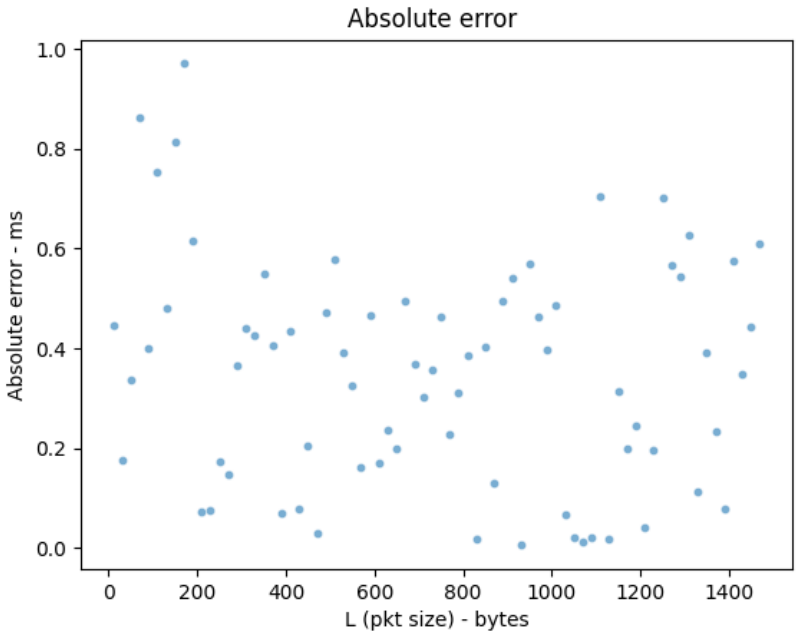

stdev
stdev - ms
L (pkt size) - bytes
5
10
15
20
25
30
35
0
200
400
600
800
1000
1200
1400

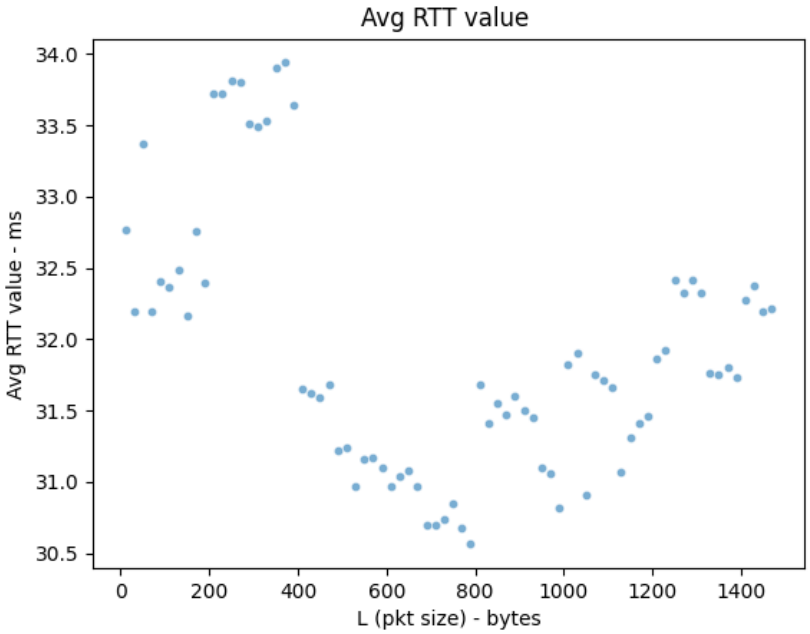
4.2 Risultati home-lyon.testdebit.info

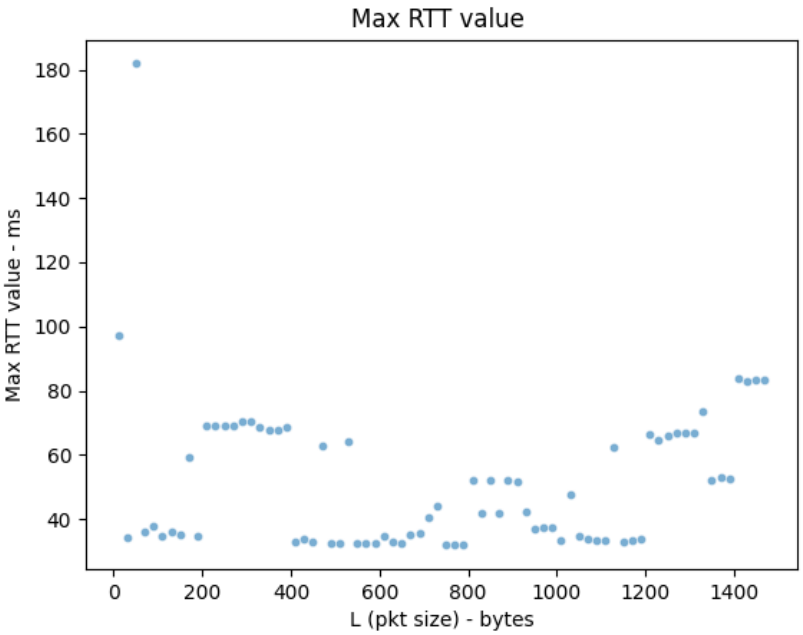


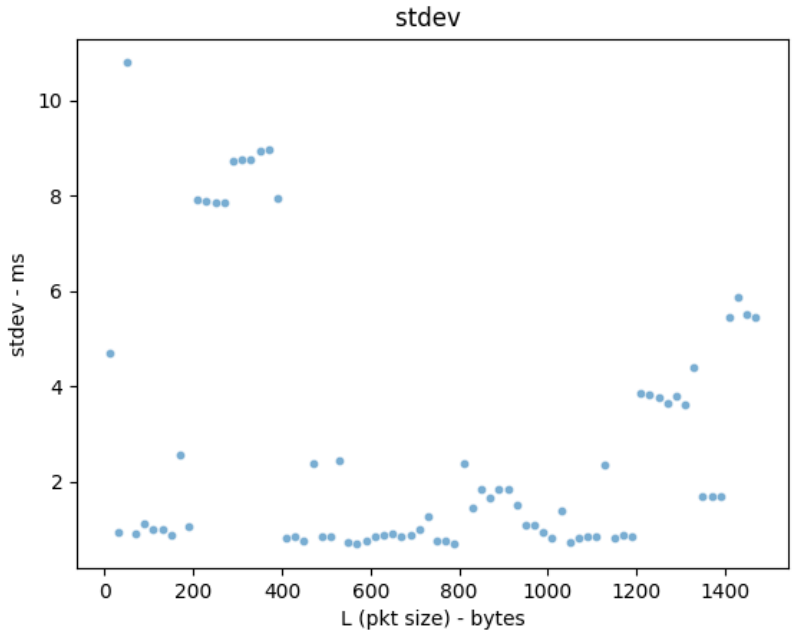

zoom with y between 25 and 40 ms
RTT - ms
L (pkt size) - bytes
40
38
36
34
32
30
28
26
0
200
400
600
800
1000
1200
1400


Min RTT value - r=29.19067568 + 0.16809189 x
Min RTT value - ms
L (pkt size) - bytes
30.0
29.8
29.6
29.4
29.2
29.0
28.8
28.6
0
200
400
600
800
1000
1200
1400


Absolute error
Absolute error - ms
L (pkt size) - bytes
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000
1200
1400


Avg RTT value
Avg RTT value - ms
L (pkt size) - bytes
34.0
33.5
33.0
32.5
32.0
31.5
31.0
30.5
0
200
400
600
800
1000
1200
1400


Max RTT value
Max RTT value - ms
180
160
140
120
100
80
60
40
0
200
400
600
800
1000
1200
1400
L (pkt size) - bytes


stdev
stdev - ms
10
8
6
4
2
0
200
400
600
800
1000
1200
1400
L (pkt size) - bytes

5 Stima di `S` e `S_bottleneck`

Partendo dai soli tempi RTT misurati per i diversi valori di `L` è possibile stimare

- ◇ il valore del throughput `S`, ipotizzando che sia lo stesso per ogni nodo
- ◇ il valore `S_bottleneck`, ipotizzando che nel percorso di andata e ritorno ci sia un nodo particolarmente più lento degli altri

Partiamo dal modello matematico che rappresenta il tempo di attraversamento di un nodo:

$$d_i = d_{i,queue} + d_{i,proc} + d_{i,prop} + d_{i,trans}$$

dove

- ◇ $d_{i,queue} = q_i(k)$ è il tempo che il pacchetto spende nel nodo i-esimo
- ◇ $d_{i,proc}$ è il tempo per l'esecuzione di algoritmi di routing, trascurabile
- ◇ $d_{i,prop} = \tau_i$ è il tempo di propagazione che dipende dalla distanza e dalla velocità della luce nel mezzo (grandezze costanti nel tempo)
- ◇ $d_{i,trans} = \frac{L}{S_i}$ durata del segnale che rappresenta il pacchetto (L dimensione del pacchetto e S_i throughput al nodo i-esimo)

Indicando con n il numero di nodi attraversati in andata e ritorno (quindi ipotizzando che il pacchetto compia lo stesso tragitto nei due versi, n è il doppio del numero calcolato nella prima parte dell'esperimento) si ha

$$RTT(L,k) = \left(\sum_{i=1}^{n} \frac{1}{S_i}\right) L + \sum_{i=1}^{n} q_i(k) + \sum_{i=1}^{n} \tau_i$$

Ora possiamo chiamare

$$a = \sum_{i=1}^{n} \frac{1}{S_i} \qquad Q(k) = \sum_{i=1}^{n} q_i(k) \qquad T = \sum_{i=1}^{n} \tau_i$$

quindi

$$RTT(L,k) = aL + Q(k) + T$$

ed eseguendo molte volte la trasmissione del pacchetto per lo stesso valore di L possiamo ipotizzare che nel caso migliore trovi le code vuote per cui $Q(k) \approx 0$, in tal caso

$$RTT_{min}(L) \approx aL + T$$

A questo punto possiamo ipotizzare due diversi scenari che permettono di calcolare il throughput in casi diametralmente opposti:

- ◇ il throughput è uguale in tutti i nodi attraversati dal pacchetto, in tal caso $a = \sum_{i=1}^{n} \frac{1}{S_i} = \frac{n}{S}$ e quindi $S = \frac{n}{a}$
- ◇ c'è un nodo nel percorso (attraversato sia all'andata che al ritorno nell'ipotesi di prima) con un throughput molto minore degli altri, in tal caso $a = \sum_{i=1}^{n} \frac{1}{S_i} \approx \frac{2}{S_{bottleneck}}$ e quindi $S_{bottleneck} = \frac{2}{a}$

Con i dati ricavati (e ricordando che n comprende sia andata che ritorno) si possono stimare quindi:

- ◇ per la rete `eduroam` del DEI

$$S = \frac{n}{a} = \frac{2 \cdot 14}{0.26689189 \frac{\text{ms}}{\text{byte}}} \approx 104911 \frac{\text{byte}}{\text{s}} = 0.839\text{Mbps}$$

$$S_{bottleneck} = \frac{2}{a} = \frac{2}{0.26689189 \frac{\text{ms}}{\text{byte}}} \approx 7494 \frac{\text{byte}}{\text{s}} = 0.060\text{Mbps}$$

- ◇ per la mia rete domestica

$$S = \frac{n}{a} = \frac{2 \cdot 18}{0.16809189 \frac{\text{ms}}{\text{byte}}} \approx 214168 \frac{\text{byte}}{\text{s}} = 1.713\text{Mbps}$$

$$S_{bottleneck} = \frac{2}{a} = \frac{2}{0.16809189 \frac{\text{ms}}{\text{byte}}} \approx 11898 \frac{\text{byte}}{\text{s}} = 0.095\text{Mbps}$$

6 Discussione dei risultati ottenuti

Il calcolo del numero di nodi, sebbene sia la parte più semplice e meno onerosa dell'esperimento, non è priva di insidie e possibili errori. Oltre a quanto già detto per la lingua degli applicativi utilizzati e parsati che limita l'utilizzabilità, si potrebbero verificare errori come `Request timeout` che nella versione italiana viene mostrato con il messaggio `Richiesta scaduta`. Questo succede quando la richiesta inviata non riceve risposta entro un tempo prefissato (che per `ping` in Windows è di 4000 ms) e ciò può accadere per problemi di rete come ad esempio una eccessiva congestione.
Un altro possibile (sebbene improbabile) errore può derivare dal fatto che, come già detto, non vi è nessuna garanzia che due pacchetti inviati in diversi istanti di tempo seguano lo stesso percorso quindi vi è la possibilità che i pacchetti inviati da `ping` e quelli inviati da `tracert` attraversino un numero diverso di nodi. Questa evenienza non si è mai verificata negli esperimenti svolti ed infatti il numero di nodi calcolato con le due modalità è sempre coinciso.

Per quanto riguarda il calcolo dei throughput medio e "bottleneck" possiamo vedere come l'errore che si commette assumendo che i tempi minimi abbiano un andamento lineare sia sempre minore di 1 ms su valori dei minimi nell'ordine dei 30 ms e quindi possiamo consideare l'approssimazione mediocre. Si noti che questi valori di minimo sono stati ottenuti con `K=200` in modo da avere una probabilità più alta di trovare le code vuote. Risultano infatti la deviazione standard piuttosto alta e i valore medio e del massimo che non seguono un andamento lineare, tutti indici della variabilità che può avere il RTT a causa di eventi non controllabili.
In altre esecuzioni di prova, è capitato di trovare una pendenza per la regressione dei minimi negativa. Secondo il ragionamento della sezione precedente si avrebbe quindi un throughput negativo, che non ha alcun significato. I risultati ottenuti in quei casi potrebbero essere dovuti a particolari condizioni di congestione della rete che hanno rallentato i primi pacchetti inviati (quelli con dimensione minore).