Computer Networks: PA #1

Started on Feb 16th : Took 1 Day

Dr. Flavio Esposito .

Rahul K Chauhan

Part 1

Writing an Echo Client-Server Application.

Explanation

For Part 1 the code the code is written in *echo_server_t.py* and *echo_client_t.py*. The purpose of these two files is to create a Client-Server echo application using TCP.
Server Side

```
$ python echo_server.py --port=5000
```

Client Side

```
$ python echo_client_t.py --port=5000 --host={Server host name}
```

These 2 files shows the basic implementation of Client-Server communication. The advanced form of the code with full functionality(RTT & Throughput) is implemented in Part 2 of the assignment.

Part 2

Performing RTT and Throughput Measurements
Explanation

The code is written in Python and checked on version 3.7.5

Requirements —
Client Side

- Python 3.3+
- numpy
- matplotlib

Server Side

- Python 3.3+

Library used for Socket Programming : socket

Server side code is in file *server.py*. Similarly, client side code is written in *client.py*. There are some classes that are overridden from the python socket library.

Usage —

Client Side

```
$ python client.py [-h] [--client CLIENT] OUTPUT MODE TYPE HOST PORT DELAY
```

Positional Arguments:

OUTPUT	Output graphs to given directory
MODE	Select mode of operation (rtt or tput).
TYPE	To define TCP protocol.
HOST	Set host to connect to.
PORT	Set port to use.
DELAY	Set a Server Delay.

Optional Arguments:

-h, –help	show this help message and exit
–client CLIENT	Name of client, for use in plot titles.

Server Side

```
$ python server.py [-h] TYPE PORT
```

Positional Arguments:

TYPE	To define TCP protocol.
PORT	Set port to use.

Part 2 continued on next page. . .

Optional Arguments:

-h, –help	show this help message and exit

Example —

First run from server *ubuntu@ec2.aws.com* (I am using AWS EC2 for testing)

```
$ python3 server.py TCP 3000
```

After that run client

```
$ python3 client.py results rtt TCP 18.191.126.51 3000 0.2 --client=rahul_pc
```

Description —

Server waits for an initialization message from the client. Client sends a 2-byte message, where the first byte represents the testing mode (either round-trip or throughput), and the second byte is a parameter to the test mode (representing either message size or message count, as a power of 2). The server receives the message, prepares to receive from the client, and sends an ACK to the client.

20 PROBES per packet size

For round-trip, the client simply sends a message to the server, and the server echoes it back as soon as possible. The client records the time elapsed.

For throughput, the server also echoes a message back to the client.

The result of each of these tests is output to results as a box-and-whisker plot.

Part 2 continued on next page. . .

Results —
Figure 1 shows the RTT without any delay. This the reason all the RTT are in between 16 - 18 seconds which is quite low when compared to Figure 3

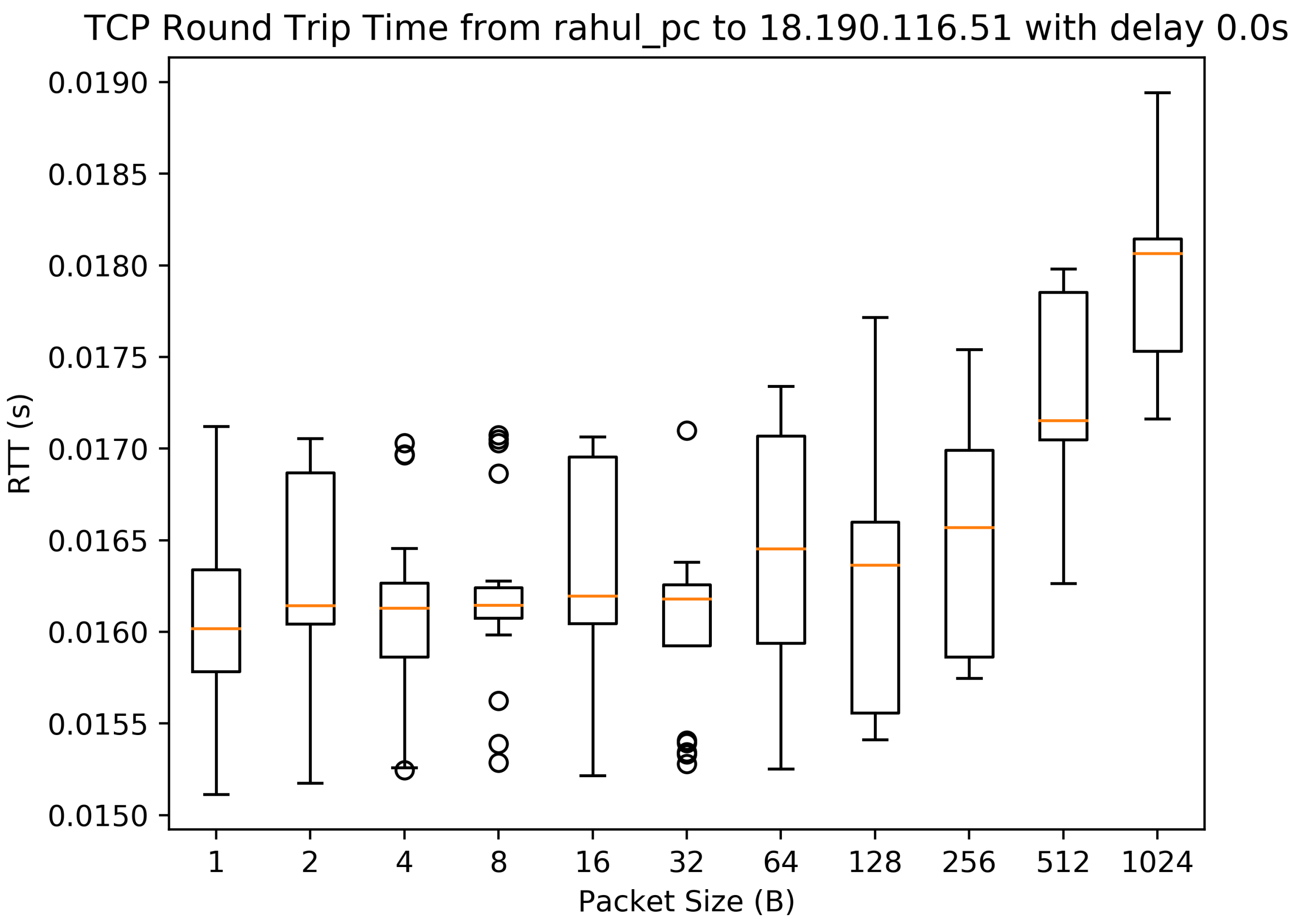

Figure 1: RTT with 0 delay

Figure 2 shows the Throughput with 0 delay. The throughput is between 1000 kbps to 4000 kbps which is quite low when compared to Figure 6

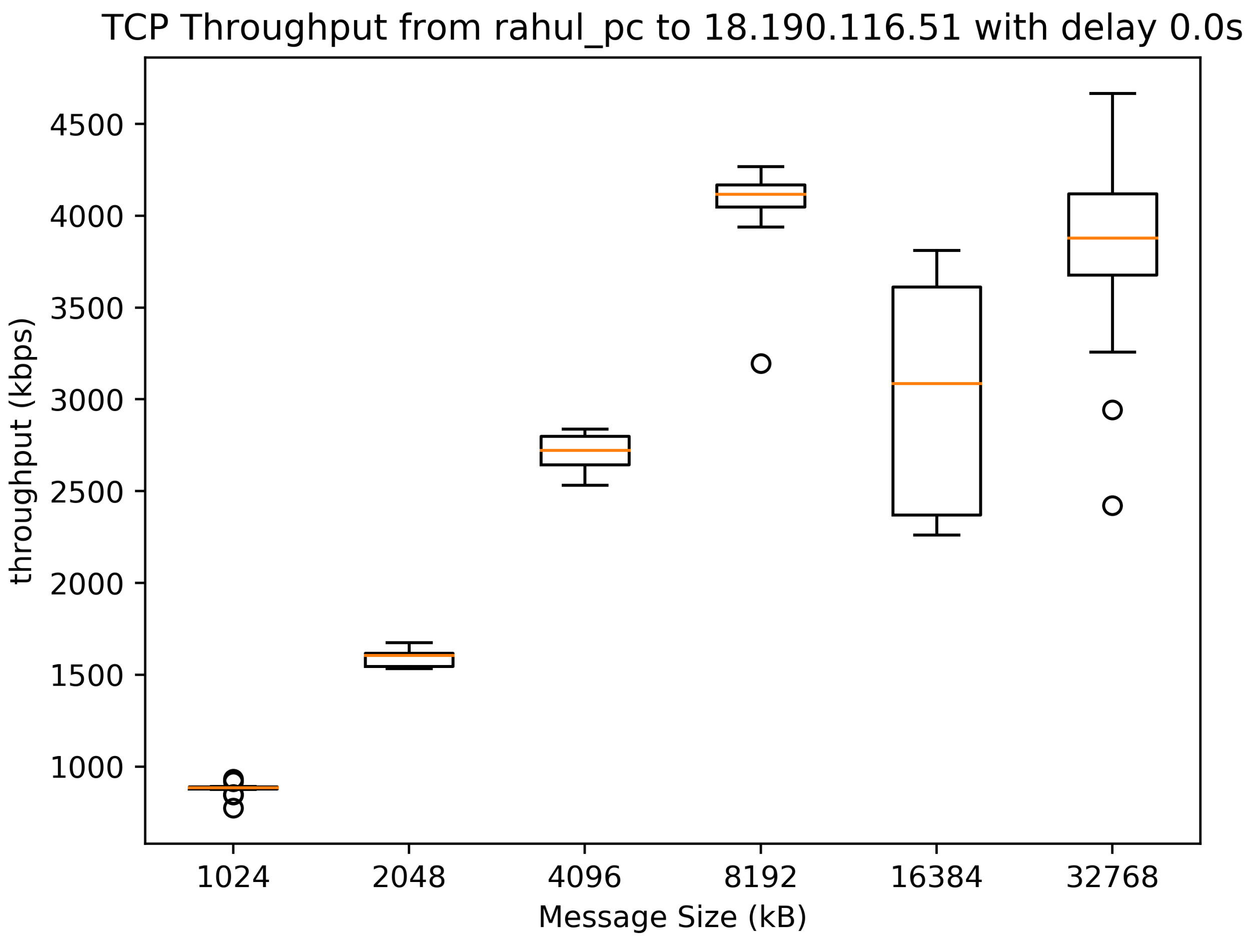

Figure 2: Throughput with 0 delay

Figure 3 shows the RTT with 200ms delay. When compared to Figure 1, the RTT increased due to server delay.

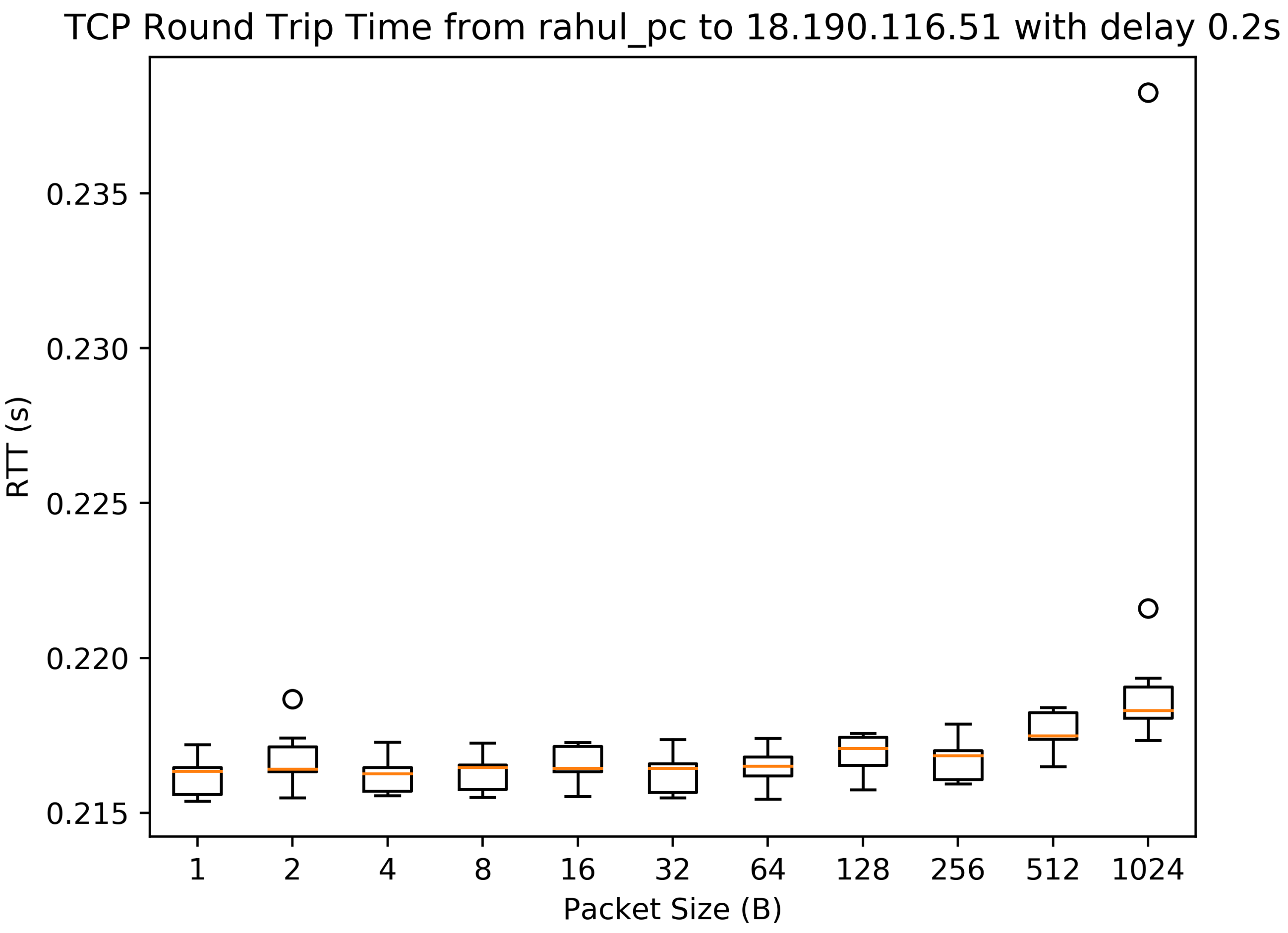

Figure 3: RTT with 200ms delay

Part 2 continued on next page. . .

Figure 6 shows the Throughput with 200ms delay. When compared to Figure 2, the Throughput increased due to server delay.

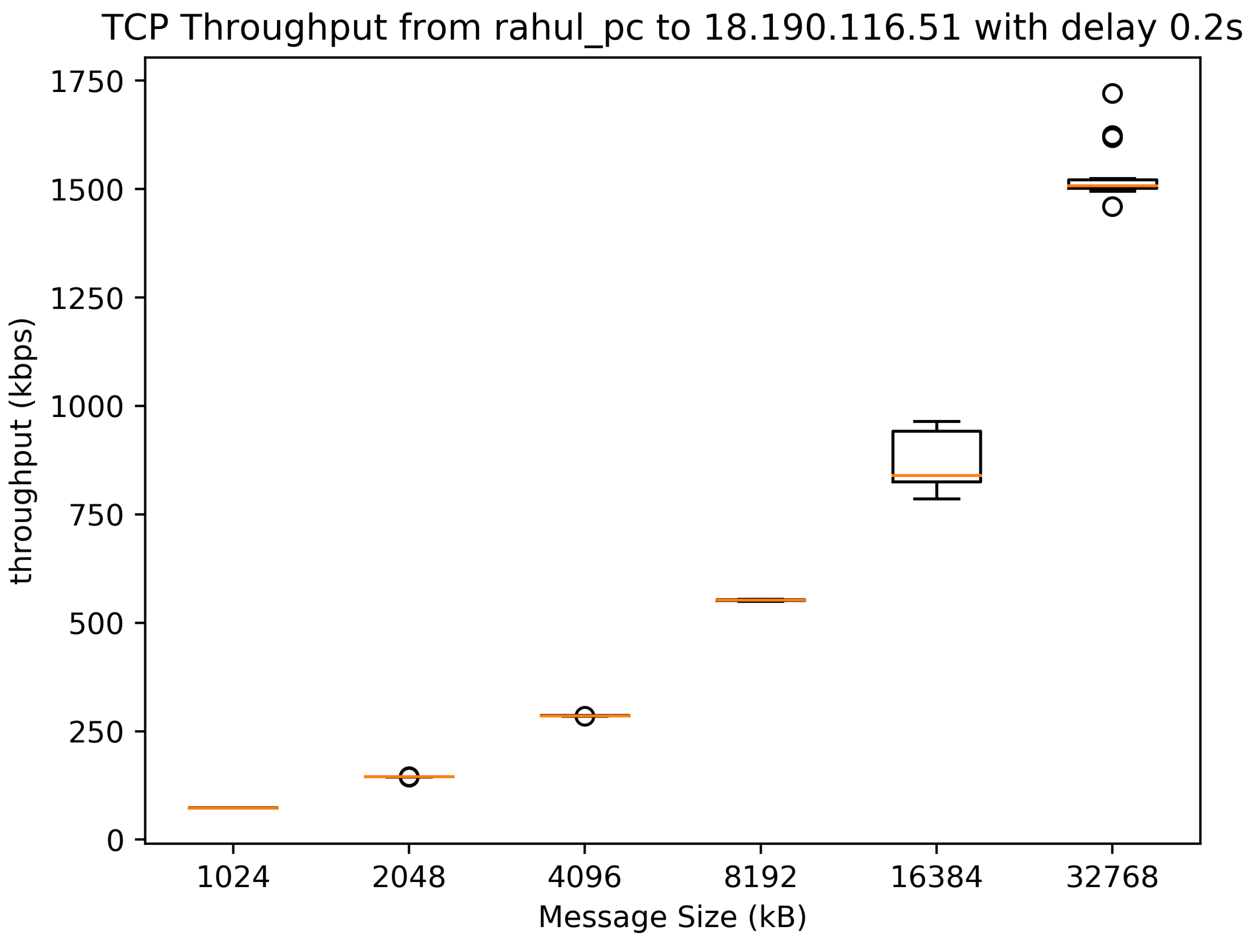

Figure 4: Throughput with 200ms delay

Part 2 continued on next page. . .

Figure 6 shows the RTT with 0ms delay at GENI.

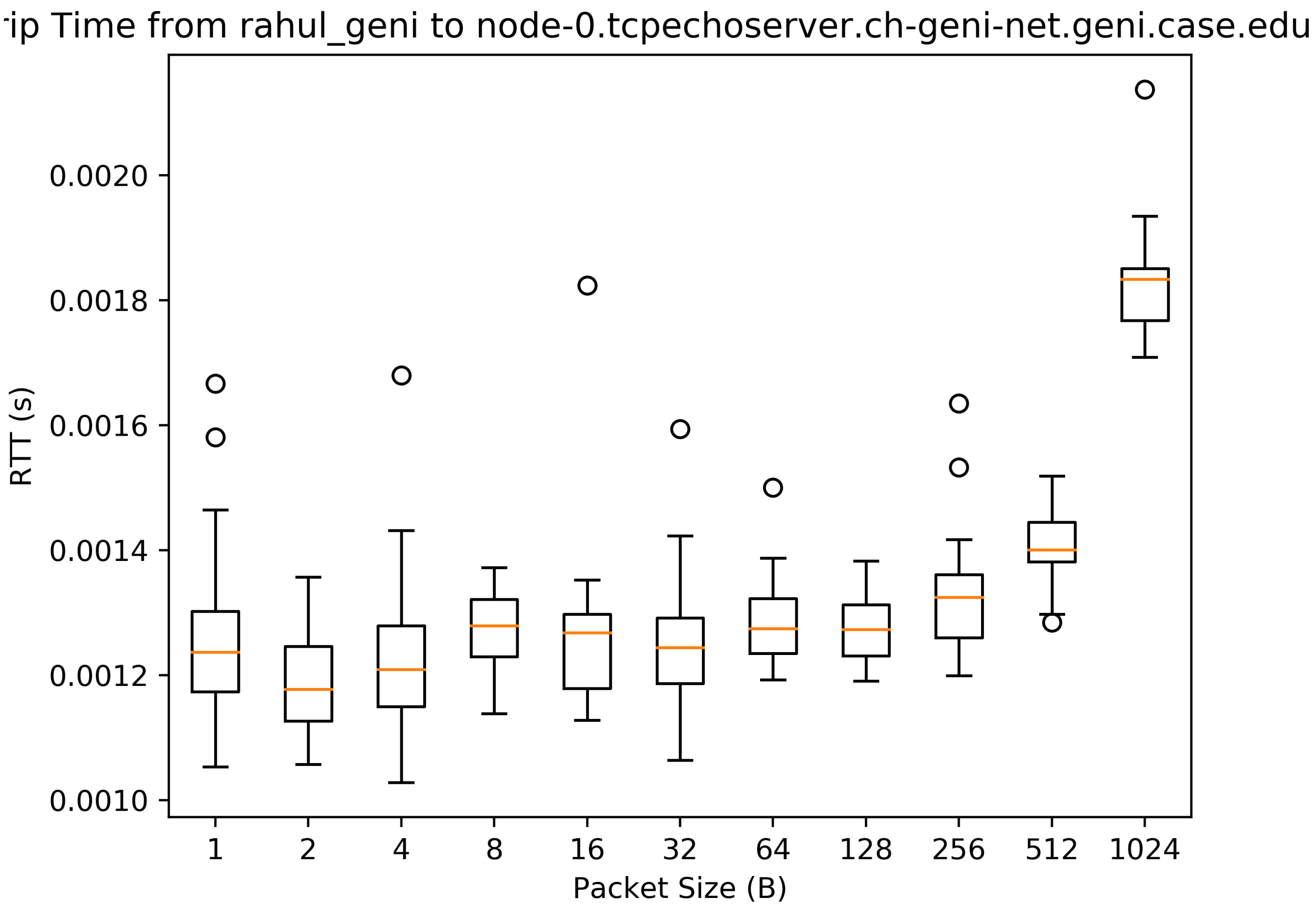

Figure 5: RTT with 0ms delay at GENI

Figure 6 shows the Throughput with 0ms delay at GENI.

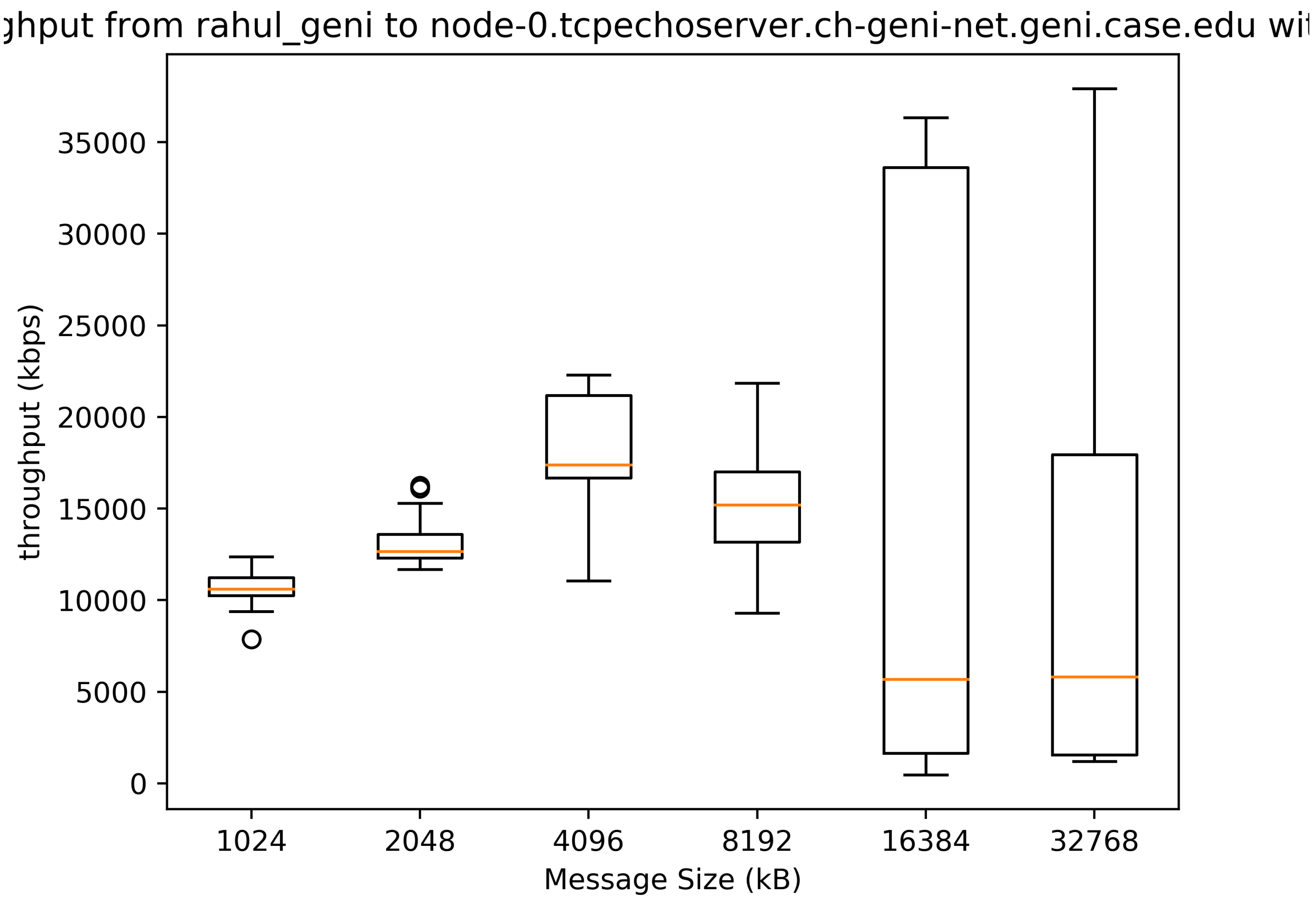

Figure 6: Throughput with 0ms delayat GENI