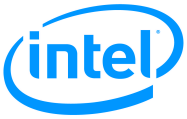
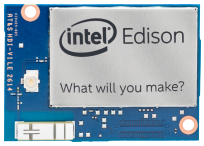
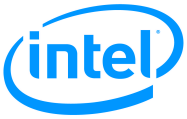




Intel® Edison Tutorial: Timing Analysis and Synchronization



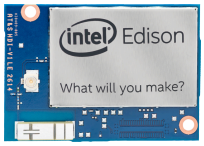
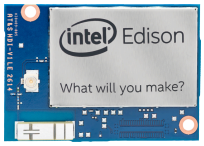


Table of Contents



Introduction

Timing is a very critical component for all engineering projects, and even more so for computers and Internet of Things (IoT) applications. For example, autonomous vehicular coordination systems rely on multiple devices making decisions that require synchronization at the millisecond scale. The potential worst case scenario of a drift in the clock of these devices could lead to fatalities.

As such, this tutorial will examine the following topics.

1. Learn how to experimentally determine how long your code took to execute
2. Manually edit the software and hardware clocks on your Intel Edison
3. Naively compare timestamps between different IoT nodes
4. Think about how to compare timestamps in a more robust manner

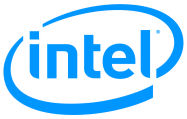
Prerequisite Tutorials

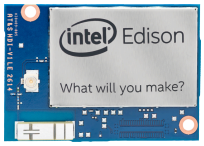
Users should ensure they are familiar with the documents below before proceeding.

1. Intel Edison Tutorial – Introduction, Linux Operating System Shell, and SFTP
2. Intel Edison Tutorial – Introduction to Linux
3. Intel Edison Tutorial – Introduction to Vim
4. Intel Edison Tutorial – Introduction to TCP Socket Communications

Things Needed

- 2xIntel Edison Kit
- 4x USB 2.0 A-Male to Micro B Cable (micro USB cable)
- 2x powered USB hub **OR** an external power supply
- A personal computer
- Access to a network with an internet connection





Timestamps on Computers – Unix/Epoch/POSIX Time

Timing is a very critical concept in computing. To process timing information, computers use numbers to represent time rather than strings.

The method most widely used on computing devices is **Epoch time**. This representation of time is defined as the number of seconds that have elapsed since 00:00:00, Thursday January 1st 1970 Coordinated Universal Time (UTC) **not counting leap seconds**. Since it does not account for leap seconds, this is not a **truly linear** or **true representation** of UTC.

For example, the date January 1, 2016 00:00:00 UTC will have the **Epoch** timestamp of 1451606400.

Follow the steps below to generate an **Epoch timestamp** using C code.

1. Access the shell program on the Intel Edison.
2. Open a new C code source file.

   $ vi get_time.c

3. Enter the code from Figure 1 into the Vim editor.

```c
#include <stdio.h> /* for printing info to the screen */
#include <time.h> /* for timing information */

int main()
{
        time_t now;
        now = time(NULL);
        printf("%d\n", now);
}
```

Figure 1: C code source file get_time.c

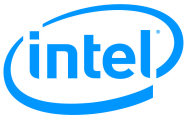
4. Save and quit the file.
5. Compile the source code into an executable binary file.

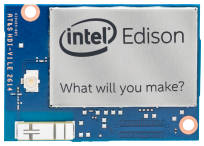
   $ gcc –o get_time get_time.c

6. Execute the binary file.

   $ ./get_time

7. Examine the output. Record it, and convert it to a human readable timestamp.





Code Execution Time

By calling the **time** function, developers can determine how long certain portions of code take to execute. Follow the below steps to learn how this is possible.

1. Open a new C code source file.

   $ vi simple_timing_analysis.c

2. Enter the code from Figure 2 into the Vim editor.

   NOTE: Any code can be placed between the comments /* **A** */ and /* **B** */. The function **usleep()** is used because it should take approximately the same amount of time to return as the indicated by the input argument. For additional information, refer to the link below.

   http://man7.org/linux/man-pages/man3/usleep.3.html

```c
#include <stdio.h> /* for printing info to the screen */
#include <time.h> /* for timing information */
#include <unistd.h> /* for sleep() definition */

int main()
{
        time_t start, end;
        start = time(NULL);
        /*
         * Place code that you would like to time between the markers A and B
         */
        /* A */
        usleep(2000000); /* how long to sleep for in microseconds */
        /* B */
        end = time(NULL);
        printf("Time taken for code segment between A and B to execute: %d\n",
                        end - start);
}
```

Figure 2: C code source file simple_timing_analysis.c

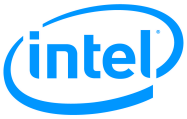
3. Save and quit the file.
4. Compile the source code into an executable binary file.

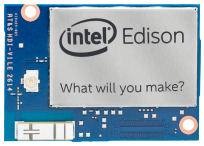
   $ gcc –o simple_timing_analysis simple_timing_analysis.c

5. Execute the binary file.

   $ ./simple_timing_analysis

6. Edit the source code file such that **usleep** takes 2,100,000 microseconds to return.
7. Compile the code and execute the resulting binary file.





Adding Precision

The **time** function returns the current time as an integer to the nearest second. For some applications, this resolution may not be sufficient. As such, this section will discuss using **gettimeofday** to obtain the current timestamp to the nearest microsecond. While the function provides up to microsecond resolution, the hardware may not be able to support this, and the microsecond level data may be meaningless. Check the documentation for the device in use before using the **gettimeofday** for critical segments of code.

1. Open a new C code source file.

   $ vi precise_timing_analysis.c

2. Enter the code from Figure 3 into the Vim editor.

```
#include <stdio.h> /* for printing info to the screen */
#include <sys/time.h> /* for gettimeofday() and timeval */
#include <unistd.h> /* for sleep() definition */

#define MILLION 1000000.0

int main()
{
        struct timeval start, end;
        double start_epoch, end_epoch;
        gettimeofday(&start, NULL);
        /*
         * Place code that you would like to time between the markers A and B
         */
        /* A */
        usleep(2000000);
        /* B */
        gettimeofday(&end, NULL);
        start_epoch = start.tv_sec + start.tv_usec/MILLION;
        end_epoch = end.tv_sec + end.tv_usec/MILLION;
        printf("Time taken for code segment between A and B to execute: %lf\n",
                        end_epoch - start_epoch);
}
```

Figure 3: C code source file precise_timing_analysis.c

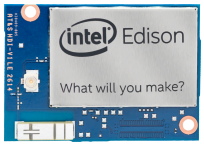
3. Save and quit the file.
4. Compile the source code into an executable binary file.

   $ gcc –o precise_timing_analysis precise_timing_analysis.c

5. Execute the binary file.

   $ ./precise_timing_analysis

6. Edit the source code file such that **usleep** takes 2,100,000 microseconds to return.
7. Compile the code and execute the resulting binary file.



Manual Clock Realignment

One method of rectifying drift is to manually align the clock to an accurate time keeper. Follow the steps below to adjust the clock on the Intel Edison manually.

1. Ensure the Intel Edison has access to a network with an internet connection.
2. Issue the following command.

   $ timedatectl set-time "2015-03-05 13:37:00"

   Notice the error message.

```
root@edison:~# timedatectl set-time "2015-03-05 13:37:00"
Failed to set time: Automatic time synchronization is enabled
root@edison:~#
```

Figure 4: Notice the error message generated when attempting to manually set the clock on the Intel Edison while NTP is enabled

   This error occurs when the Intel Edison is running a service to automatically synchronize its clock to a more reliable clock.

   NOTE: The input string format is "**YYYY-MM-DD hh:mm:ss**".

3. To change the clock manually, the automatic time synchronization must be disabled.

   $ timedatectl set-ntp false

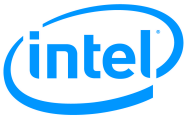
4. Verify that the automatic synchronization has been disabled.

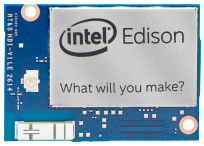
   $ timedatectl

   The output should show the field labelled **NTP enabled** set to **no**.

```
root@edison:~# timedatectl set-time "2015-03-05 13:37:00"
Failed to set time: Automatic time synchronization is enabled
root@edison:~# timedatectl set-ntp false
root@edison:~# timedatectl
      Local time: Thu 2016-09-22 17:54:00 UTC
  Universal time: Thu 2016-09-22 17:54:00 UTC
        RTC time: Thu 2016-09-22 17:54:00
       Time zone: Universal (UTC, +0000)
     NTP enabled: no
NTP synchronized: yes
 RTC in local TZ: no
      DST active: n/a
```

Figure 5: Output from timedatectl when NTP is successfully disabled





5. Adjust the time now that Network Time Protocol (NTP) daemon is not enabled.

   $ timedatectl set-time "2015-03-05 13:37:00"
   $ timedatectl

   The output should match the output below

```
root@edison:~# timedatectl set-time "2015-03-05 13:37:00"
root@edison:~# timedatectl
      Local time: Thu 2015-03-05 13:37:03 UTC
  Universal time: Thu 2015-03-05 13:37:03 UTC
        RTC time: Thu 2015-03-05 13:37:04
       Time zone: Universal (UTC, +0000)
     NTP enabled: no
NTP synchronized: no
 RTC in local TZ: no
      DST active: n/a
```

Figure 6: Output from timedatectl when the time has been manually set

6. Enable the **NTP daemon** by issuing the command below.

   $ timedatectl set-ntp true

   Wait about one minute and examine the output from **timedatectl**.

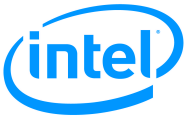
   If the clock is prematurely queried, it may appear to be misaligned with the real time. This is because the operating system may not have updated the clock.

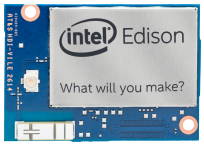
   Even if the Local and Universal time are aligned and correct, the RTC time may not have been updated.

   $ timedatectl

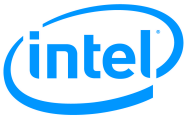
```
root@edison:~# timedatectl
      Local time: Thu 2016-09-22 18:03:32 UTC
  Universal time: Thu 2016-09-22 18:03:32 UTC
        RTC time: Thu 2016-09-22 18:03:32
       Time zone: Universal (UTC, +0000)
     NTP enabled: yes
NTP synchronized: yes
 RTC in local TZ: no
      DST active: n/a
```

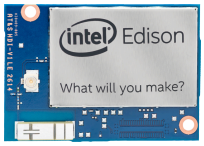
Figure 7: Output from timedatectl when the NTP service is successfully enabled





Note: Check the **Local**, **Universal** and **RTC** time. They should all be the same, but they will not agree with the above figure, since the screenshot was taken at the time of writing this tutorial.





Difference in Timestamps

Follow the steps below to see the differences in timestamps from as measured from two discrete Intel Edison nodes using the TCP/IP protocol to enable data transfer.

1. Download the **zip archive** labelled **FILES.zip** from the link below.

   https://drive.google.com/drive/folders/0B4NGslzPqDhvdUdObnAzdGZUb2M

2. Extract the contents of the zip archive. The files listed below should be present.

   server.c
   client.c

3. Establish a connection between each Intel Edison and a Wi-Fi network with internet access.
4. Designate one Intel Edison as the server. This will be referred to as **the server**.
5. Discover the IP address of the server Intel Edison

   $ configure_edison --showWiFiIP

6. Transfer the file labelled **server.c** to the server Intel Edison via SFTP.
7. Compile the file

   $ gcc –o server server.c

8. Designate the other Intel Edison as the client. This will be referred to as **the client**.
9. Transfer the file labelled **client.c** to the client Intel Edison via SFTP
10. Compile the file

    $ gcc –o client client.c

11. Execute the binary file labelled **server** on the server Intel Edison.

    $ ./server <PORT_NO>

12. Execute the binary file labelled **client** on the client Intel Edison.

    $ ./client <IP_ADDR_SERVER> <PORT_NO>

13. Verify the functionality of the client and server.

Figure 8: Output from client when the system is operational

Figure 9: Output from server when the system is operational

Examine the difference in timestamps by following the steps below.

Generate the current timestamp on the Intel Edison. Send this as a string to the server Intel Edison. Generate the current timestamp on the server Intel Edison when the server Intel Edison successfully reads the message received from the client Intel Edison.

The server Intel Edison will then compute the difference in timestamps.

$$timestamp\ difference = server\ time\ stamp - client\ time\ stamp$$

Follow the below steps to generate the current timestamp with microsecond precision on the client and send this timestamp formatted as a string to the server.

14. Open the **client.c** file on the client Intel Edison.

    $ vi client.c

15. Include the **sys/time.h** library and define **MILLION**.

    #include <sys/time.h>
    #define MILLION 1000000.0 //1,000,000.0

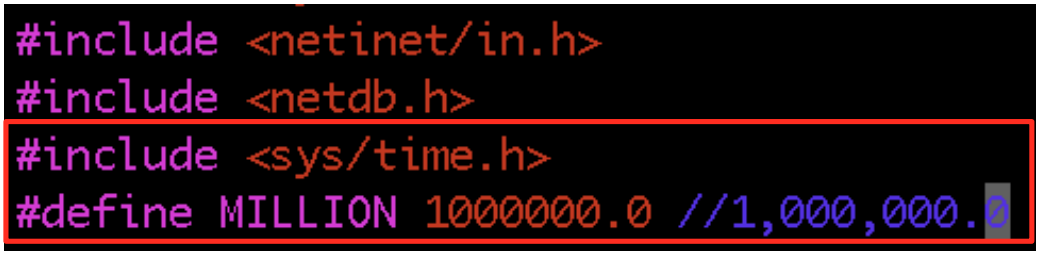


Figure 10: Including the "sys/time.h" library and defining MILLION

16. Declare the variables **now** and **now_epoch**.

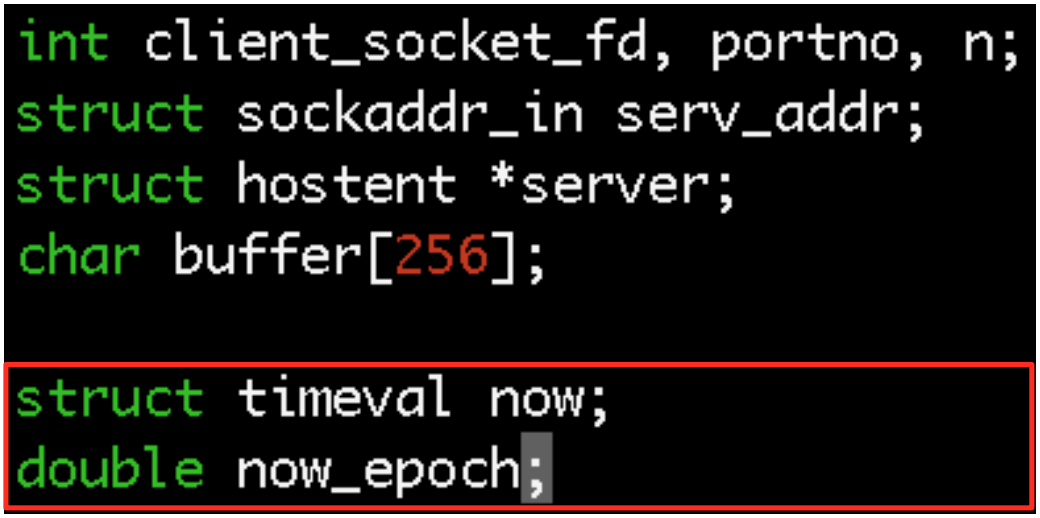


Figure 11: Declaring now and now_epoch

17. Comment out the section that says **get user input**.

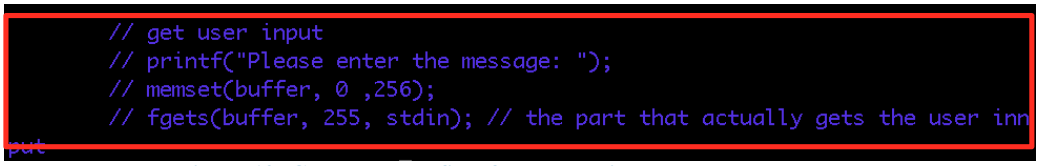


Figure 12: C code source file after converting some code to comments

18. Add the following lines directly below.

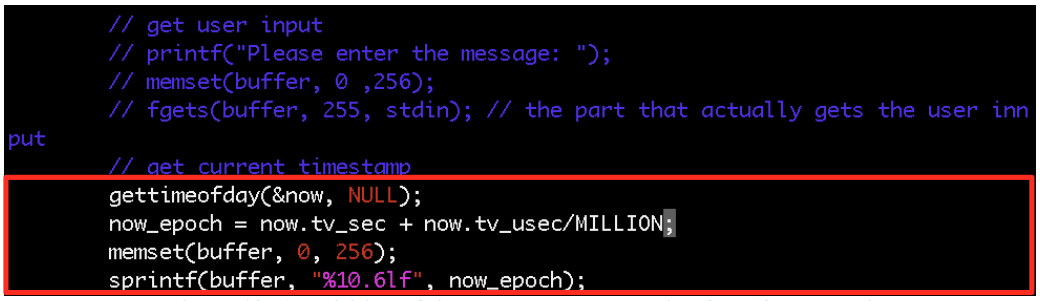


Figure 13: Acquisition of timestamp and conversion from float to string

19. Save and quit the file.
20. Compile the C code source file into a binary executable file.

    $ gcc –o client client.c

21. Execute the binary file.

    $ ./client <SERVER_IP_ADDR> <PORT_NO>

22. Repeat steps 11 and 12 above.

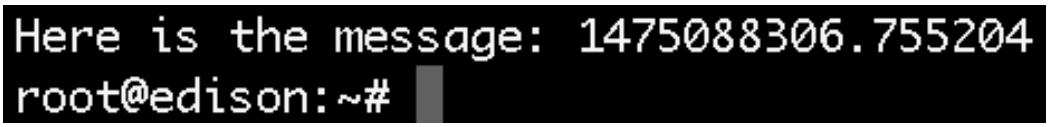


Figure 14: Output from client when the system is operational

Modify the server in a similar fashion to generate the timestamp directly after successfully reading a message from the client.

23. Open the **server.c** file.

    $ vi server.c

24. Include the **sys/time.h** library and define **MILLION**.

    #include <sys/time.h>
    #define MILLION 1000000.0 //1,000,000.0

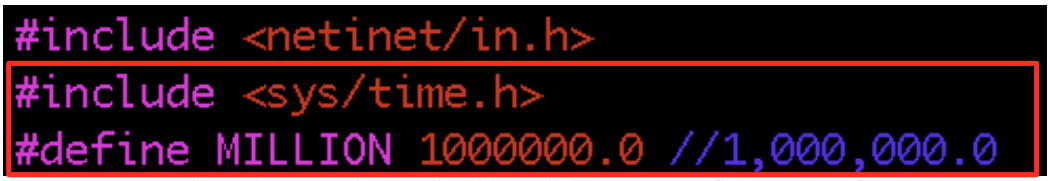


Figure 15: Including the "sys/time.h" library and defining MILLION

25. Declare the variables **now**, **server_epoch**, and **client_epoch**.

```
int main(int argc, char *argv[])
{
        int server_socket_fd, client_socket_fd, portno;
        socklen_t clilen;
        char buffer[256];
        struct sockaddr_in serv_addr, cli_addr;
        int n;

        struct timeval now;
        double server_epoch, client_epoch;
```

Figure 16: Declaring now, server_epoch and client_epoch

26. Get the current timestamp directly after reading a message from the client.

    Display the timestamp received from the client, and generated from the server.

```
        memset(buffer, 0, 256);
        n = read(client_socket_fd, buffer, 255); // read what the client sent too
the server and store it in "buffer"
        if (n < 0) {
                error("ERROR reading from socket");
        }
        gettimeofday(&now, NULL);
        server_epoch = now.tv_sec + now.tv_usec/MILLION;

        // print the message to console
        printf("CLIENT: %s || SERVER: %10.6lf\n", buffer, server_epoch);
```

Figure 17: Acquisition of the current time stamp and editing the call to printf

27. Save and quit the file.
28. Compile the C code source into an executable binary file.

    $ gcc –o server server.c

29. Repeat steps 11 and 12 above.

```
root@edison:~# !.
./server 8000
CLIENT: 1475172164.897662 || SERVER: 1475172164.875322
root@edison:~#
```

Figure 18: server output

30. Display the difference in timestamps by following the steps below.

Convert the message read from the socket descriptor by the server from a **string** to a **double**.

Edit the **printf** section of the code such that it matches the below screenshot.

```
// sscanf is the opposite of printf
// it reads items from the character array "buffer"
// grabs variables from the string and stores them into the appropriate
// variables as indicated by the function
sscanf(buffer, "%lf\n", &client_epoch);

// print the message to console
printf("CLIENT: %s || SERVER: %10.6lf\n", buffer, server_epoch);
printf("DIFFERENCE: %10.6lf\n", server_epoch - client_epoch);
```

Figure 19: Edits required to enable the server sided software to calculate the difference between the client recorded timestamp and the server recorded timestamp

For more information regarding **sscanf**, refer to the link below.

http://man7.org/linux/man-pages/man3/sscanf.3p.html

31. Save and quit the file.
32. Compile the C code source file into an executable binary file.

    $ gcc –o server server.c

33. Repeat steps 9 and 10 above.

```
root@edison:~# ./server 8000
CLIENT: 1475175464.548094 || SERVER: 1475175464.435218
DIFFERENCE:  -0.112876
root@edison:~#
```

Figure 20: Output from server showing the difference in recorded timestamps

34. Simulate an event where the clocks are misaligned due to drift.

    Issue the below commands to create an artificial drift of approximately 10 minutes on the client Intel Edison.

    $ timedatectl set-ntp false
    $ timedatectl set-time <TEN_MINUTES_AGO>

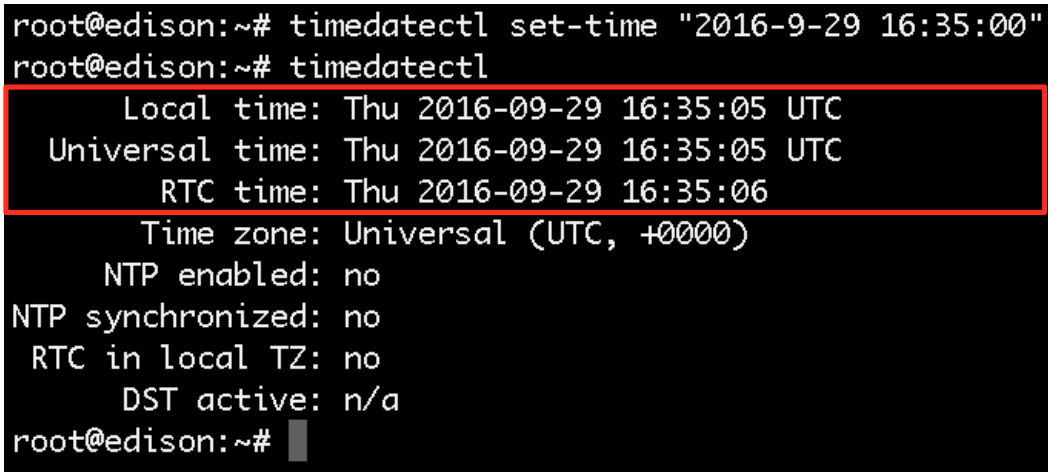


Figure 21: Editing the clock on the client such that there is a 10 minute simulate drift between the client and the server nodes

35. Repeat steps 9-10 above, notice the timestamp difference is now much larger.

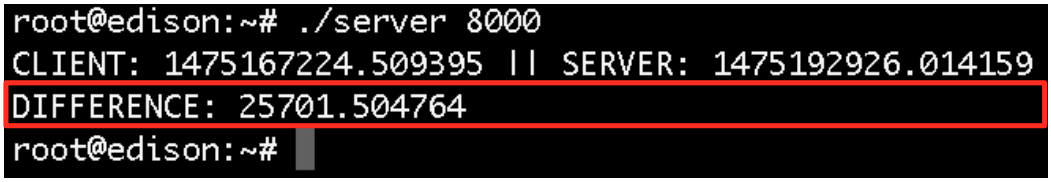


Figure 22: Output from server demonstrating the effect of misaligned clocks

36. Enable NTP on the client Intel Edison

$ timedatectl set-ntp true

Next Tasks

1. The difference in timestamps is due to **clock drift**, **network latency** and other factors. **Network latency** is likely to be the biggest cause of difference in timestamps.

   Design an experiment to determine an estimated value for network latency.

2. Repeat steps 9-10 above, examine the value of **DIFFERENCE**. Notice that it changes.

   Design a method to determine the average difference in timestamps between the server and the client.

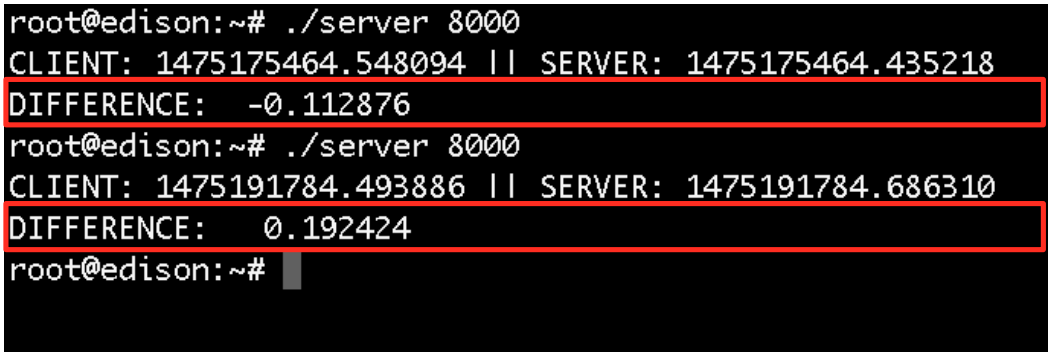


Figure 23: Output from the server after multiple runs showing that the difference in timestamps is not constant

3. **EXTENSION:** conduct the experiment designed in step 1. Record the empirically determined average network latency.