CMPT 431 Assignment 3

Oscar Smith-Sieger
Mayanka Medhe

1

Transient Synchronous Communication-
In this question, we used socket programming on python platform. The server is sending time using the time.time() function, which returns the time as a floating point number expressed in seconds since the epoch, in UTC. To start the server use command-
python server_TCP.py (for TCP) and
python server_UDP.py (for UDP)

The client is receiving the time using client.recv() function in case of TCP and client.recvfrom() function in case of UDP. To start the client use command-
python client_TCP.py (for TCP) and
python client_UDP.py (for UDP)
Using TCP -

```
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$ python client_TCP.py
Server time = 1550611552.74 Corrected Server time = 1550611552.74
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$
```

```
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$ python server_TCP.py
Current Server time is 1550611552.74. Server time sent!
Client disconnected!
```

Using UDP -

```
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$ python client.py
Server time = 1550611588.8 Corrected Server time = 1550611590.02
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$
```

```
mayanka@mayanka-Aspire-E5-522G:~/Documents/SEM_SFU/CMPT 431/A3$ python server.py
Current Server time is 1550611588.8. Server time sent!
Current Server time is 1550611588.8. Server time sent!
Current Server time is 1550611588.8. Server time sent!
```

2

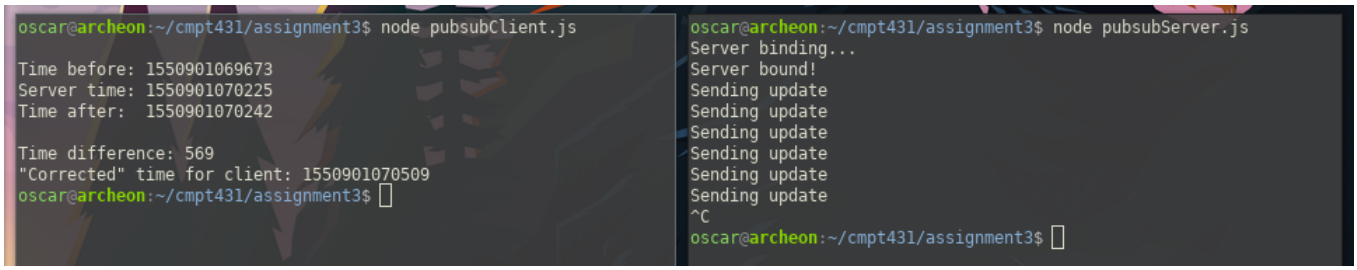
We utilized ZMQ under node.js. For the standard req-rep pattern, we implemented things exactly as we did for the previous assignments, as the paradigm is the exact same.

```
oscar@archeon:~/cmpt431/assignment3$ node repreqClient.js
Connecting to server...
Sending 'getTime_ms'
Received reply: '1550900956150'

Time before: 1550900956135
Server time: 1550900956150
Time after:  1550900956152

Time difference: 17
"Corrected" time for client: 1550900956158
oscar@archeon:~/cmpt431/assignment3$
```

```
oscar@archeon:~/cmpt431/assignment3$ node repreqServer.js
Server listening on 16540!
Received request: 'getTime_ms'
```

For the pub-sub pattern, things were more complicated. Isolating the "before" time is nigh impossible, given the delay of updates (which we set as 1 second). Instead, we stored the time before we subscribed, as

a sort of approximate. This still fits within the algorithm we were given and results in a time correctness resolution proportional to the frequency of updates (as mentioned, 1 second in this case).



3

Method	Ease of programming (developer's perspective) (1=easiest, 7=toughest)	Clock Synchronization Accuracy (1=highest, 7=lowest)	Adaptability: amount of work needed to interoperate with other systems (1=best, 7=worst)
Socket: UDP	1	6	4
Socket: TCP	2	5	5
ZeroMQ: ReqRep	5	1	6
ZeroMQ: PubSup	7	7	7
RPC or RMI	3	2	3
REST	4	4	1
SOAP	6	3	2

It was easiest to program using sockets as python has a very easy to use socket package. It was most difficult to program ZeroMQ: PubSub because it was difficult to isolate the "before" time, then comes the ReqRep method. RPC was moderately difficult given that it was written in C language. REST and SOAP were a little more difficult because of the javascript involved.

The clock synchronization accuracy is the worst in case of ZeroMQ: PubSub because of the aforementioned problem of isolating the "before" time. It is the highest in case of ZeroMQ: ReqRep, then RPC then REST. SOAP, TCP and UDP all are in python and are using same function time.time() to get the current time. Hence, its accuracy is similar.

Adaptability is the highest in case of REST because it is a web based service then comes SOAP. It is lowest in case of ZeroMQ: PubSub and ZeroMQ: ReqRep. While, adaptability is moderate in the rest of the methods.