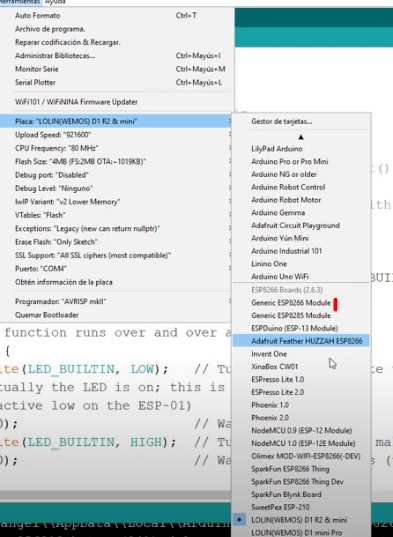
**TRABAJO FINAL PROGRAMACION DISTRIBUIDA**

Este trabajo final tiene como objetivo la investigación del funcionamiento del tiempo real a comparación del obtenido en la práctica de la materia, se utilizará una placa WEMOS D1, se explicaran los pasos para instalarla y usarla, los pasos para comunicarse con la pc a través de wifi, la realización de las comunicaciones y sus respectivas comparación con las de la práctica.

Para instalar el WEMOS D1 segui los pasos del video <https://www.youtube.com/watch?v=9mkPMXblPJI>, primero instalamos el driver para que la pc detecte la placa, luego en el arduino agregamos el paquete de ESP8266 e instalamos el mismo gestor de tarjeta.

En mi caso uso el Generic ESP8266 Module en vez del WEMOS D1.



Luego se probó el ejemplo que hay en el video para ver si funcionaba la placa correctamente.

Ahora ya llegamos a la parte de los experimentos de la práctica, los cuales trabajaban con cliente/servidor, enviando mensajes y calculando el tiempo que tardaba en enviarse y llegar.

Para hacer lo mismo con la placa y calcular lo que sería “tiempo real” de comunicación

Se utilizo la siguiente página para poder crear el codigo <https://circuits4you.com/2018/11/26/esp8266-nodemcu-tcp-socket-server-arduino-example/>

Ahí se aprendió a crear un cliente servidor mediante una comunicación por TCP/IP

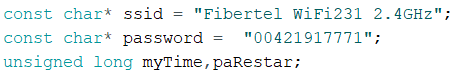
Primero incluimos los paquetes que necesitaremos 

Definimos el puerto, el cliente por wifi y servidor con el puerto 8888



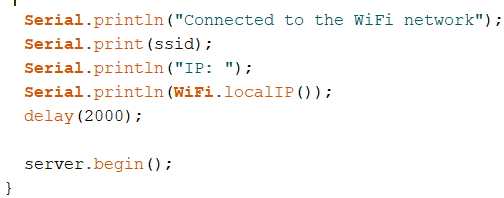
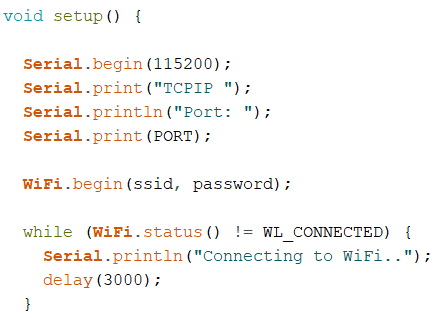
Por un lado tendremos las constantes que se necesitan para la conexión wifi, básicamente el wifi que se usa y la contraseña.

Por otro el buffer que es para el mensaje que enviaremos y los long que se usarán para calcular el tiempo que tarda desde el envío hasta que llega.

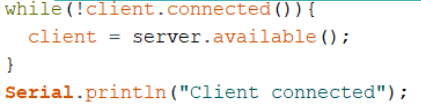


El Serial.begin(115200) es el upload speed que tenemos para poder ver el monitor serial.

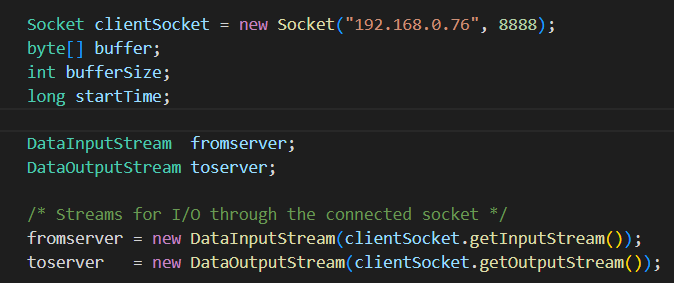
Lo más importante acá es el WiFi.begin que inicia la conexión por wifi con el WEMOS, una vez conectado lo imprimo y muestro la IP a la que está conectado para poder utilizarla en el cliente con el que se conectará.



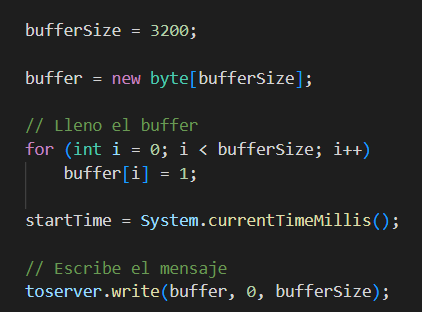
En el void loop se hace lo siguiente, primero se espera a conectar con el cliente, cuando se conecte el cliente se avisa y se procede.



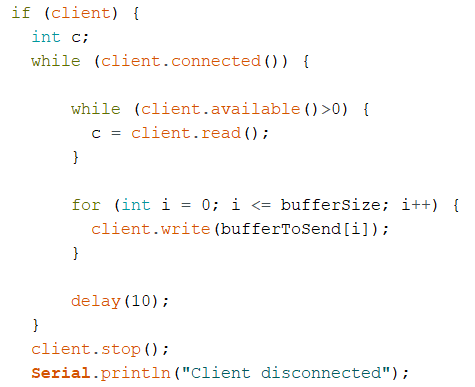
Para el cliente se creó un programa de java con sockets, primero conectamos al host (ip mostrada en la pantalla de arduino) y al puerto 8888



Una vez conectado el cliente se le envían 3200 bytes al servidor, también acá se toma el momento de inicio del tiempo de comunicación

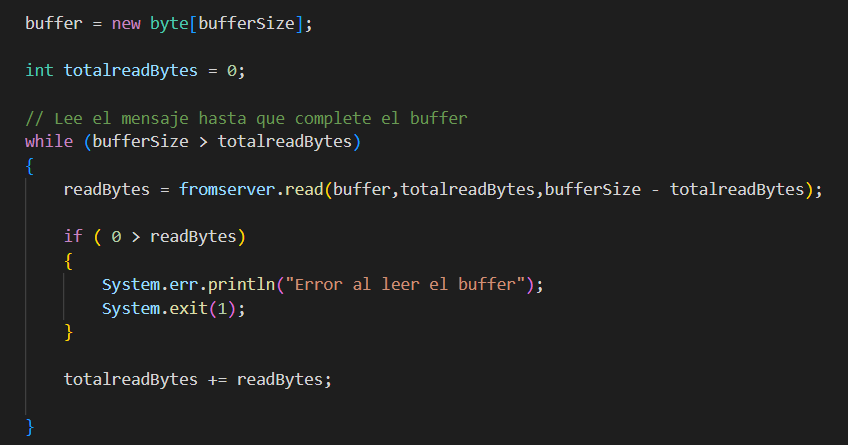


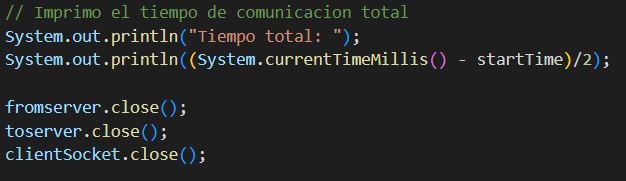
Desde el lado del servidor en el código de arduino una vez conectado con el cliente si este existe se comienza a recorrer hasta que termine de leer los bytes que recibe y una vez finaliza se envían la misma cantidad de bytes al servidor. BufferSize contiene 3200 bytes igual que en java para que los mensajes desde ambos lados sean relativamente idénticos.



Desde el cliente lo que se hace es leer los 3200 bytes que recibimos, como no se leen todos de golpe, hay que crear un while hasta que se hayan leído por completo. Esto no fue necesario desde el arduino porque se lee byte a byte y no se pierde nada de la comunicación.

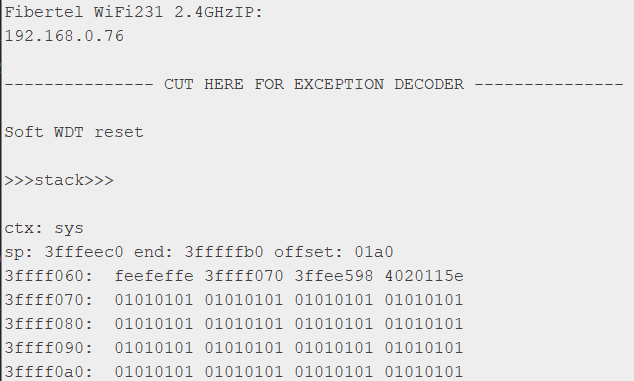
Por último se calcula el tiempo total de comunicación ahora que ya finalizó y se cierran las conexiones



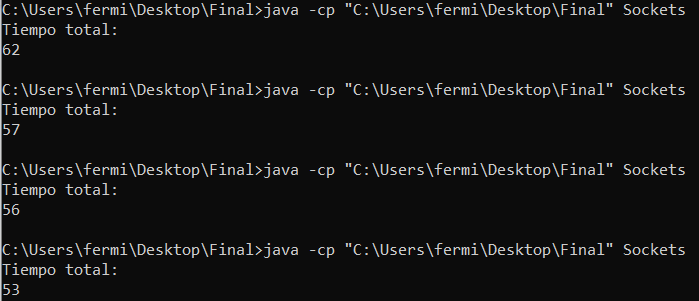
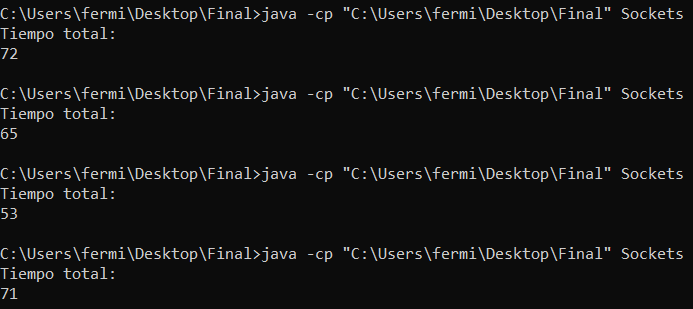


Como pudimos observar el código del cliente en java es muy similar a lo que se hizo en la práctica de la cursada, con pequeñas diferencias por el hecho de que debemos conectarnos a la placa por medio de wifi.

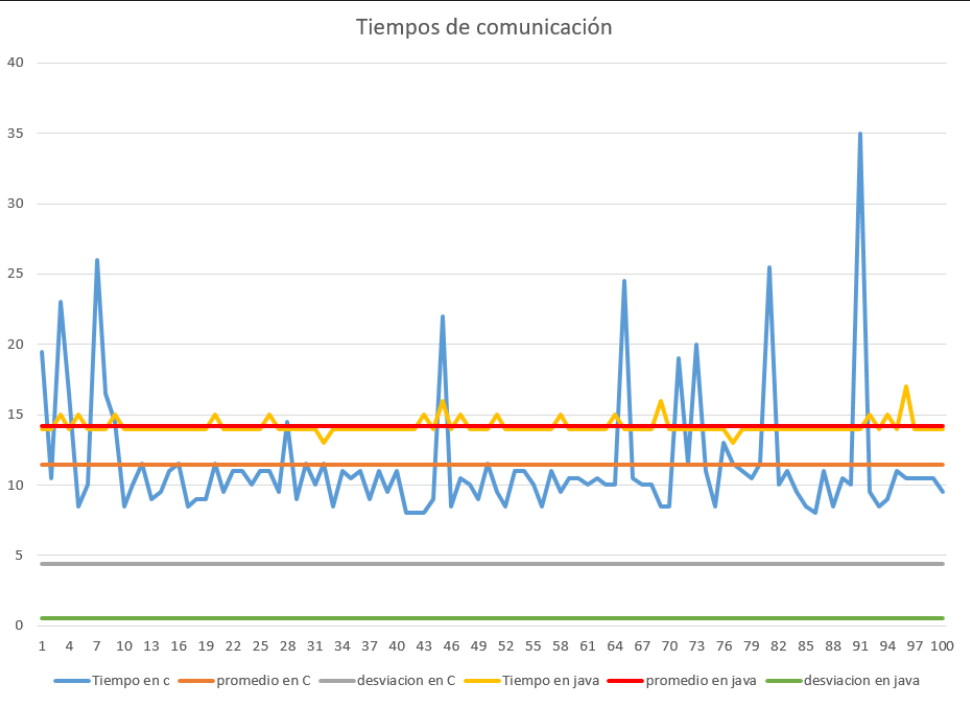
Se puede ver que el tiempo ronda los 50 a 80 milisegundos, hay que tener en cuenta esto se hizo con 3200 bytes, si se supera esa cantidad desde el lado de la placa arduino nunca terminaba de cargar el vector y la conexión a wifi quedaba colgada de esta manera

y seguía…

Aca los tiempos:



Es decir, si lo comparamos con los tiempos que habíamos obtenido en la práctica donde trabajamos sin una placa se puede ver que los obtenidos ahora son peores y con menos bytes por el hecho de que estamos trabajando con tiempo “real” y no en una misma máquina. (Tiempos con mensajes de 10000 bytes)

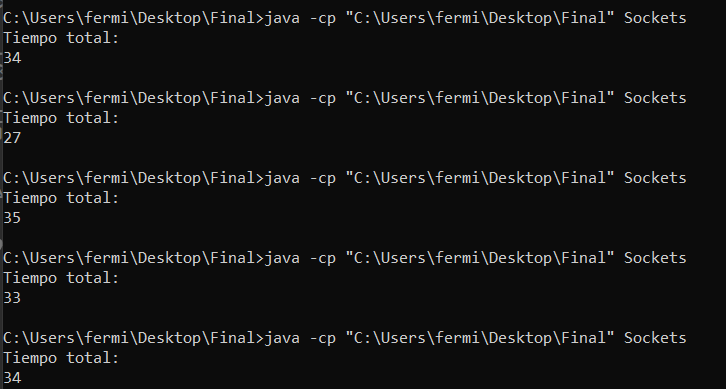


Se puede observar que con 50 o 100 bytes el tiempo total ronda de 30 hacia arriba, con 500 disminuye a 20 en promedio y con 1500 ya vuelve a tener 30 en promedio, con 3200 bytes es el tiempo total más grande rondando lo dicho anteriormente, así que se puede ver que en los primeros bytes hasta 1500 los bytes rondan en promedio los 30 con excepciones y despues de eso comienza a subir hasta llegar a superar los 3200 cuando no se puede transmitir más nada

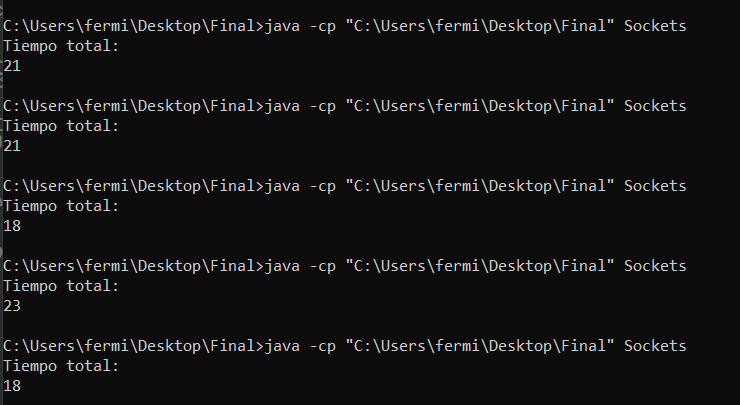
Con 50 bytes:



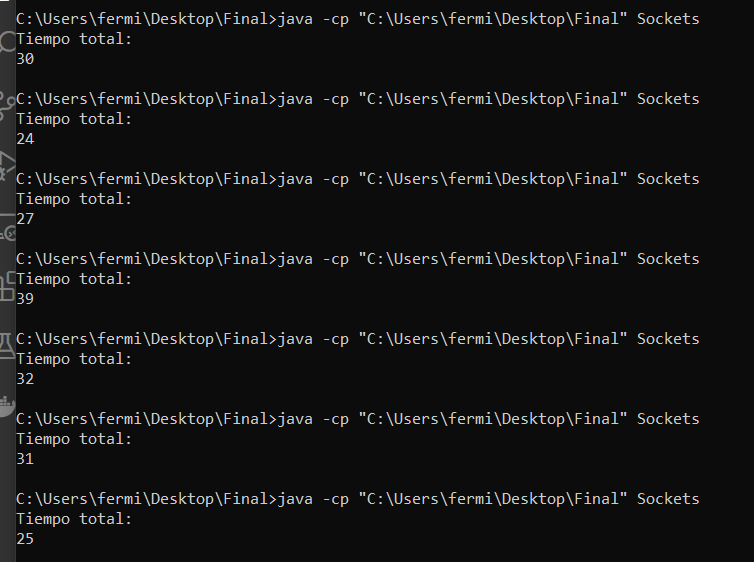
Con 100 bytes:



Con 500:



con 1500:



con 2500:



**Conclusiones**

Se pudo ver que la mayor diferencia a tener en cuenta es que al trabajar con tiempo real perdimos en velocidad a comparación de hacer todo en una misma maquina, la diferencia ya es grande y tan solo usamos 3200 bytes a comparación de los 10000 que se usaron en las de la práctica.

Otra diferencia es que a la hora de leer en arduino no se pierde ningún byte, cosa que en las de la práctica puede suceder y por eso se hizo uso de un checksum para ir verificando que los mensajes llegaran correctamente en las prácticas. Es decir desde java podríamos perder información en el ejemplo que se realizo en este tp, aunque eso no afecte en casi nada al tiempo total de comunicación.

Como última diferencia vimos que hasta cierto punto (1500 bytes de tamaño en mensajes) el tiempo de comunicación se comportaba de forma monótona, siendo casi siempre igual y después de eso comenzaba a aumentar y llegar al punto donde se cuelga con una cantidad más pequeña que la de la propia maquina, lo cual sucede porque la placa tiene una capacidad mucho más limitada.