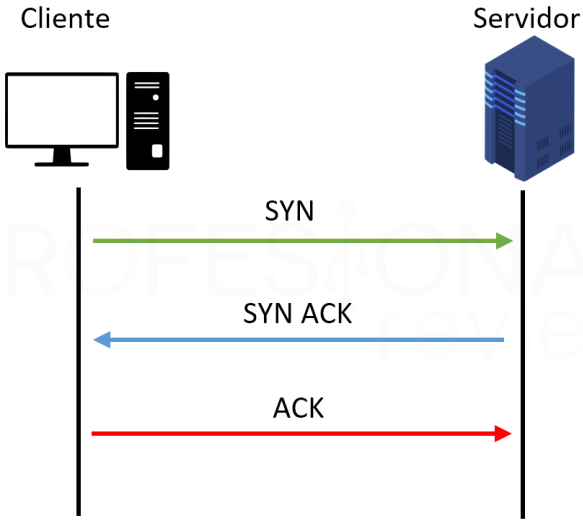
Servicio TCP de la capa de transporte

TCP (Transmission Control Protocol) es un protocolo de comunicación de datos utilizado en la capa de transporte del modelo OSI (Open Systems Interconnection) y en el modelo TCP/IP. Es un protocolo orientado a la conexión que proporciona una transmisión de datos fiable y en orden, asegurando que los datos enviados desde una máquina lleguen correctamente a su destino.

UDP (User Datagram Protocol) y TCP son los protocolos básicos de nivel de transporte para realizar conexiones entre sistemas principales de Internet. TCP y UDP permiten que los programas envíen mensajes a las aplicaciones de otros sistemas principales y reciban mensajes de dichas aplicaciones. Cuando una aplicación envía a la capa de transporte una petición de envío de un mensaje, UDP y TCP dividen la información en paquetes, añaden una cabecera de paquete incluida la dirección de destino y envían la información a la capa de red para su proceso adicional. TCP y UDP utilizan puertos de protocolo en el sistema principal para identificar el destino específico del mensaje.

Las aplicaciones y los protocolos de nivel superior utilizan UDP para realizar conexiones de datagrama y TCP para realizar conexiones de corriente. La interfaz de sockets de sistema operativo implementa estos protocolos.

- TCP proporciona entrega continua fiable de datos entre sistemas principales de Internet.
- Definiciones de campo de cabecera de TCP.



Protocolo TCP

Obviamente interviene en la capa de transporte tanto de OSI como TCP/IP y es un protocolo orientado a la conexión a diferencia de IP. Esto significa que los dos equipos que intervienen deben aceptar la conexión antes de efectuar el intercambio de datos. Esto es muy importante para asegurar que los datos llegan y toda la comunicación de estas características:

- La comunicación empieza y finaliza de forma cordial y sin roturas.
- Asegura el transporte fiable de datos para que todos lleguen al destino.
- Ordena los segmentos de datos cuando llevan a través del protocolo IP.
- Los datos pueden formar segmentos de longitud variable.

- Puede monitorizar el flujo de datos para no saturar la red, lo que vendría siendo un QoS (Quality of Service).
- Permite circular datos simultáneamente, aunque vengan de distintas funciones, lo que se llama multiplexación.

Desarrollo:

Inicialmente reamos un proyecto en Visual Studio Code, basta con crear una carpeta y dos scripts con extensión .js para esta práctica se nombraron de la siguiente manera:

- P_tcp_cliente.js
- P_tcp_servidor.js

Comenzaremos por comprender que conforma el servidor:

El módulo net en Node.js proporciona una API para crear y manejar tanto servidores como clientes de red. Permite trabajar con conexiones TCP (Transmission Control Protocol) y algunas funcionalidades básicas de conexiones en red, permite crear un clientes y servidores TCP.

El módulo net en Node.js ofrece soporte para IPC (Inter-Process Communication) y conexiones de red TCP. Proporciona clases y métodos para crear y manejar servidores (net.Server), clientes/sockets (net.Socket), y listas de bloqueo (net.BlockList). También incluye utilidades para gestionar conexiones, direcciones IP, y eventos relacionados con la red.

Puedes conocer las funcionalidades que ofrece el módulo en la documentación de node: https://nodejs.org/api/net.html#new-netserveroptions-connectionlistener

Ahora seguimos definiendo la lógica que implica la práctica comenzando con el servidor:

Primero, se importa el módulo net para utilizar las funcionalidades de red TCP. Luego, se crea un servidor TCP utilizando net.createServer(). Se define un manejador de eventos para la conexión ('connection'), donde se configuran manejadores para cuando se recibe un mensaje de un cliente ('data'), cuando la conexión se cierra ('close') y cuando ocurre un error ('error'). En el evento 'data', el servidor imprime el mensaje del cliente y responde con 'Mensaje recibido'. Finalmente, el servidor comienza a escuchar en el puerto 8080 y se imprime un mensaje confirmando que el servidor está escuchando.

Teniendo el siguiente fragmento de código en P_tcp_servidor.js:

```
const net = require('net');

const server = net.createServer();

server.on('connection', (socket)=>{
    socket.on('data', (data)=>{
        console.log('\nEl cliente ' + socket.remoteAddress + ":" +
socket.remotePort + "dice: " + data);
        socket.write('Mensaje recibido');
    });

    socket.on('close', ()=>{
        console.log('Comunicación finalizada');
```

```
    });

    socket.on('error', (err)=>{
        console.log(err.message);
    });
});

server.listen(8080, ()=>{
    console.log('servidor esta escuchando al puerto', server.address().port);
});
```

Ahora continuamos con el cliente primero conociendo lo que realiza nuestro código:

Primero, se importa el módulo net para utilizar las funcionalidades de red TCP y readline-sync para permitir la entrada del usuario desde la línea de comandos. Luego, se define la configuración del cliente en el objeto configuracion, especificando el puerto y la dirección del servidor. A continuación, se crea una conexión TCP utilizando net.createConnection(configuracion).

Antes de proceder al código primero debemos de instalar la biblioteca readline-sync para permitirnos la entrada en línea de comandos de forma síncrona con el siguiente comando en la terminal:

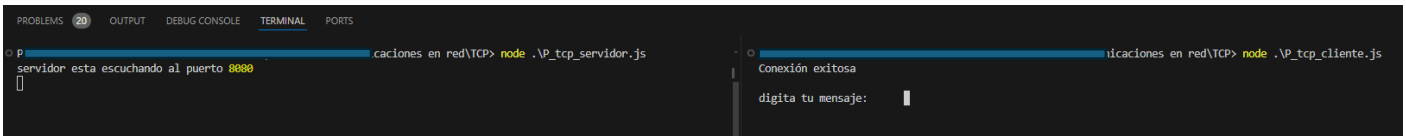
```
npm install readline-sync
```

Posterior a ello agregamos la lógica a nuestro archivo P_tcp_cliente.js:

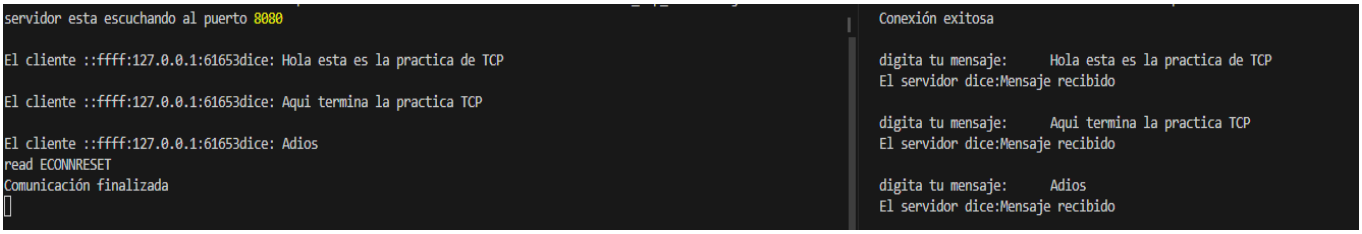
```
const net = require('net');
const readline = require('readline-sync');

const configuracion = {
    port: 8080,
    host: '127.0.0.1'
};

const cliente = net.createConnection(configuracion);

cliente.on('connect', ()=>{
    console.log('Conexión exitosa');
    sendLine();
});

cliente.on('data', (data)=>{
    console.log('El servidor dice:' + data);
    sendLine();
});

cliente.on('error', (err)=>{
    console.log(err.message);
});

function sendLine() {
    var line = readline.question('\ndigita tu mensaje: \t');
    if (line == "0") {
        cliente.end();
```

```
    }else{
        cliente.write(line);
    }
}
```

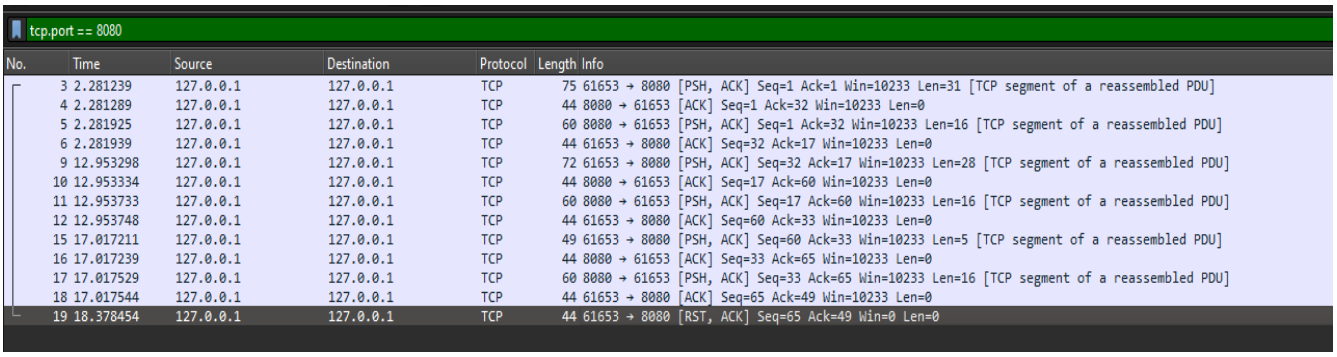
Una vez establecida la lógica se deben de abrir dos terminales, una para la ejecución del servidor TCP y otra para la ejecución del cliente TCP como se muestra a continuación:



Se debe tener una salida como la que se presenta a continuación:



También se puede hacer el análisis de los datos recibidos a través de wireshark como se muestra a continuación:



Teniendo una vista detallada de los mensajes enviados como se muestra a continuación: