Transmission Control Protocol (TCP)

What is TCP?

- TCP is connection based protocol
- No data is sent until the server and client perform initial connection(handshake)
- TCP connection present the data to application as a continuous stream of bytes
- **Why should you use TCP? The Transmission Control Protocol (TCP):**
  - **Is reliable:** Packets dropped in the network are detected and retransmitted by the sender.
  - **Has in-order data delivery:** Data is read by your application in the order it was written by the sender.

TCP Sockets

- Create stream socket

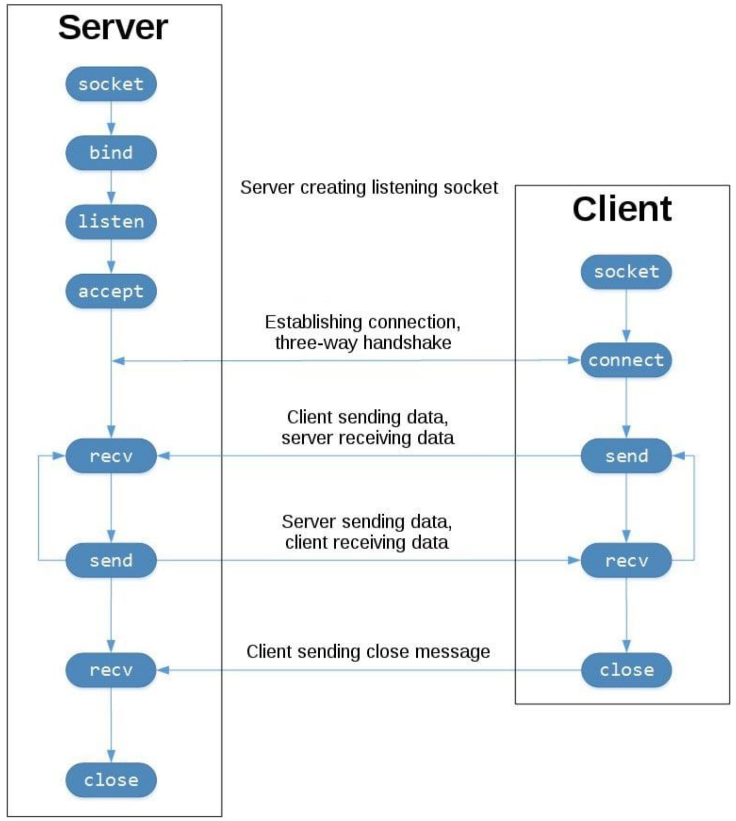
```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- **Server Methods**
  - sock.**bind**(address): Bind the socket to address.
  - sock.**listen**(*backlog*)
    - Listen to clients, **backlog** argument specifies the max no. of queued connections
  - conn, address =sock.**accept**()
    - Accept a connection. The socket must be bound to an address and listening for connections. **The return value** is a pair (conn, address).

TCP Socket con’t

- **Client Method**
  - sock.**connect**(server_*address*): Connect to a remote socket at server_*address*.
- **Server and client methods**
  - Data =conn.**recv**(*bufsize*) or sock.recv(bufsize)[from client]
  - Receive data from the socket. **The return value** (Data) is a bytes object representing the data received. The maximum amount of data to be received at once is specified by ***bufsize***.

TCP Client Server

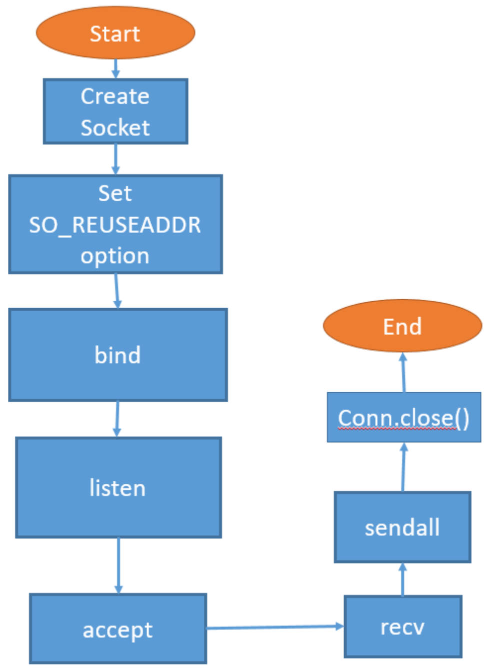

Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

TCP Client Server Methods

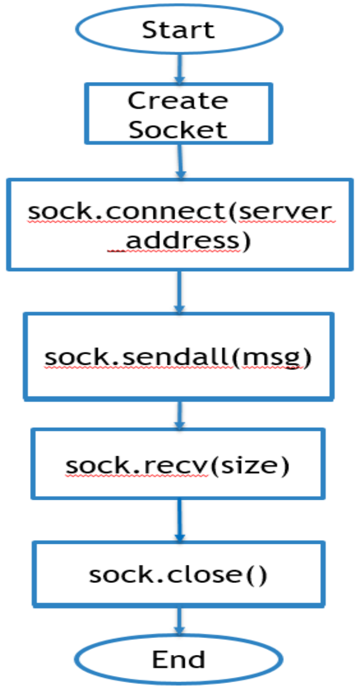
- **A listening** socket does just what its name suggests. It listens for connections from clients.
- When a client connects, the server calls **.accept()** to accept, or complete, the connection.
- The client calls **.connect()** to establish a connection to the server and initiate the three-way handshake.
- The handshake step is important because it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client, or server can reach the other.
- In the middle is the round-trip section, where data is exchanged between the client and server using calls to **.send() and .recv().**
- At the bottom, the client and server close their respective sockets.

TCP Client Server

Server



Client



TCP Client

```
def client(host, port):
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    sock.connect(server_address)
    print("Connecting to %s port %s" % server_address)
    # Send data
    sock.sendall(b'Hi there, server')
    #reply = recvall(sock, 16)
    # Receive  data
    reply=sock.recv(1024)
    print('The server said', reply.decode())
    sock.close()
    return
```

```
# Main
host=socket.gethostname()
port=2456
client(host,port)
```

recvall function

```
def recvall(sock, length):
    data = b''
    while len(data) < length:
        more = sock.recv(length - len(data))
        if not more:
        raise EOFError('was expecting %d bytes but only received'
        ' %d bytes before the socket closed'
        % (length, len(data)))
        data += more
    return data
```

TCP Server

```
import socket
def server(host, port):
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port and host
    sock.bind((host, port))
    # Listen to clients, backlog argument specifies
    # the   max   no.     of   queued   connections
    sock.listen(10)
    print('Listening at', sock.getsockname())
```

TCP Server Cont.

```
while True:
        conn, sockname = sock.accept()
        print('We have accepted a connection from', sockname)
        print(' Socket name:', conn.getsockname())
        print(' Socket peer:', conn.getpeername())
        message=conn.recv(1024)
        print("data recieved from client=",message.decode())
        #message = recvall(conn, 16)
        #print(' Incoming sixteen-octet message:', repr(message))
        conn.sendall('Farewell, client'.encode())
        conn.close()
    print(' Reply sent, socket closed')
    return
server(socket.gethostname(),2456)
```

Closed Connections, Half-Open Connections

- socket.shutdown(how)
- Shut down one or both halves of the connection.
- If how is SHUT_RD, further receives are disallowed.
- If how is SHUT_WR, further sends are disallowed.
- If how is SHUT_RDWR, further sends and receives are disallowed.
- Ex:

```
sock.shutdown(socket.SHUT_WR)
```

getpeername()

- getpeername(): Returns the remote address to which this socket is connected.
- For TCP :
  - Server side: conn. getpeername()
  - Client side : s. getpeername()
- For UDP???

Getsockname method

- getsockname() : retrieve the
  current IP and port to which the socket is bound.
- For TCP :
  - Server side: conn.getsockname()
  - Client side : s.getsockname()
- For UDP:
  - s.getsockname()

ANY QUESTIONS ?

Thank You